

**OGC® DOCUMENT: 23-001R0**

External identifier of this OGC® document: <http://www.opengis.net/doc/DIS/ogcapi-connectedsystems-1/1.0>



Open  
Geospatial  
Consortium

# OGC API - CONNECTED SYSTEMS - PART 1: FEATURE RESOURCES

---

**STANDARD**  
Implementation

**DRAFT**

**Version:** 1.0

**Submission Date:** yyyy-mm-dd

**Approval Date:** yyyy-mm-dd

**Publication Date:** yyyy-mm-dd

**Editor:** Alex Robin

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

Recipients of this document are invited to submit, with their comments, notification of any relevant patent rights of which they are aware and to provide supporting documentation.

### **License Agreement**

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

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

### **Copyright notice**

Copyright © 2024 Open Geospatial Consortium

To obtain additional rights of use, visit <https://www.ogc.org/legal>

### **Note**

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. The Open Geospatial Consortium shall not be held responsible for identifying any or all such patent rights.

Recipients of this document are requested to submit, with their comments, notification of any relevant patent claims or other intellectual property rights of which they may be aware that might be infringed by any implementation of the standard set forth in this document, and to provide supporting documentation.

# CONTENTS

I. ABSTRACT .....	xiii
II. KEYWORDS .....	xiv
III. PREFACE .....	xv
IV. SECURITY CONSIDERATIONS .....	xvi
V. SUBMITTING ORGANIZATIONS .....	xvii
VI. SUBMITTERS .....	xvii
1. SCOPE .....	2
2. CONFORMANCE .....	4
3. NORMATIVE REFERENCES .....	7
4. TERMS AND DEFINITIONS .....	9
6. CONVENTIONS .....	15
6.1. Identifiers .....	15
6.2. URL Templates .....	15
6.3. Abbreviated terms .....	15
7. OVERVIEW .....	19
7.1. General .....	19
7.2. Design Considerations .....	19
7.3. Resource Types .....	20
7.4. Resource Encodings .....	22
7.5. Resource Collections .....	22
7.6. API Endpoints .....	23
7.7. Paged Responses .....	25
7.8. Search & Filtering .....	25
7.9. Link Relation Types .....	26
7.10. Security Considerations .....	27
8. REQUIREMENTS CLASS “COMMON” .....	30
8.1. Overview .....	30
8.2. API Landing Page .....	30

8.3. API Definition .....	30
8.4. Resource IDs .....	31
8.5. Unique Identifiers (UID) .....	31
8.6. Coordinate Reference Systems .....	32
8.7. Date/Time Query Parameter .....	32
<b>9. REQUIREMENTS CLASS “SYSTEM FEATURES” .....</b>	<b>35</b>
9.1. Overview .....	35
9.2. System Resource .....	36
9.3. System Canonical URL .....	40
9.4. System Resources Endpoints .....	40
9.5. System Feature Collections .....	41
<b>10. REQUIREMENTS CLASS “SUBSYSTEMS” .....</b>	<b>44</b>
10.1. Overview .....	44
10.2. Types of System/Subsystem Associations .....	44
10.3. Subsystem Resource .....	45
10.4. Subsystem Canonical URL .....	46
10.5. Subsystem Resources Endpoint .....	46
10.6. System Recursive Search .....	47
10.7. System Associations .....	48
<b>11. REQUIREMENTS CLASS “DEPLOYMENT FEATURES” .....</b>	<b>50</b>
11.1. Overview .....	50
11.2. Deployment Resource .....	51
11.3. Deployment Canonical URL .....	53
11.4. Deployment Resources Endpoints .....	53
11.5. Deployment Feature Collections .....	54
<b>12. REQUIREMENTS CLASS “SUBDEPLOYMENTS” .....</b>	<b>57</b>
12.1. Overview .....	57
12.2. Subdeployment Resource .....	58
12.3. Subdeployment Canonical URL .....	58
12.4. Subdeployment Resources Endpoint .....	58
12.5. Deployment Recursive Search .....	59
12.6. Deployment Associations .....	60
<b>13. REQUIREMENTS CLASS “PROCEDURE FEATURES” .....</b>	<b>63</b>
13.1. Overview .....	63
13.2. Procedure Resource .....	64
13.3. Procedure Canonical URL .....	66
13.4. Procedure Resources Endpoints .....	66
13.5. Procedure Feature Collections .....	67
<b>14. REQUIREMENTS CLASS “SAMPLING FEATURES” .....</b>	<b>70</b>
14.1. Overview .....	70
14.2. Features of Interest .....	71

14.3. Sampling Feature Resource .....	72
14.4. Sampling Feature Canonical URL .....	74
14.5. Sampling Feature Resources Endpoints .....	74
14.6. Sampling Feature Collections .....	76
14.7. Dynamic properties .....	76
<b>15. REQUIREMENTS CLASS “PROPERTY DEFINITIONS” .....</b>	<b>79</b>
15.1. Overview .....	79
15.2. Property Resource .....	80
15.3. Property Canonical URL .....	81
15.4. Property Resources Endpoints .....	81
15.5. Property Collections .....	82
<b>16. REQUIREMENTS CLASS “ADVANCED FILTERING” .....</b>	<b>85</b>
16.1. Overview .....	85
16.2. Definitions .....	86
16.3. Common Resource Query Parameters .....	87
16.4. Common Feature Query Parameters .....	89
16.5. System Resources Endpoint Query Parameters .....	90
16.6. Deployment Resources Endpoint Query Parameters .....	93
16.7. Procedure Resources Endpoint Query Parameters .....	97
16.8. Sampling Feature Resources Endpoint Query Parameters .....	98
16.9. Property Resources Endpoint Query Parameters .....	100
16.10. Combination of Filter Parameters .....	102
16.11. Indirect Associations .....	102
<b>17. REQUIREMENTS CLASS “CREATE/REPLACE/DELETE” .....</b>	<b>105</b>
17.1. Overview .....	105
17.2. Systems .....	106
17.3. Subsystems .....	107
17.4. Deployments .....	107
17.5. Subdeployments .....	108
17.6. Procedures .....	108
17.7. Sampling Features .....	109
17.8. Property Definitions .....	109
17.9. Custom Collections .....	110
<b>18. REQUIREMENTS CLASS “UPDATE” .....</b>	<b>113</b>
18.1. Overview .....	113
18.2. Systems .....	113
18.3. Deployments .....	114
18.4. Procedures .....	114
18.5. Sampling Features .....	115
18.6. Derived Properties .....	115
<b>19. REQUIREMENTS CLASSES FOR ENCODINGS .....</b>	<b>117</b>
19.1. Requirements Class “GeoJSON Format” .....	117

19.2. Requirements Class “SensorML Format” .....	128
<b>ANNEX A (NORMATIVE) CONFORMANCE CLASS ABSTRACT TEST SUITE .....</b>	<b>142</b>
A.1. Supporting Tests .....	142
A.2. Conformance Class “Common” .....	143
A.3. Conformance Class “System Features” .....	145
A.4. Conformance Class “Subsystems” .....	147
A.5. Conformance Class “Deployment Features” .....	150
A.6. Conformance Class “Subdeployments” .....	153
A.7. Conformance Class “Procedure Features” .....	156
A.8. Conformance Class “Sampling Features” .....	159
A.9. Conformance Class “Property Definitions” .....	161
A.10. Conformance Class “Advanced Filtering” .....	163
A.11. Conformance Class “Create/Replace/Delete” .....	177
A.12. Conformance Class “Update” .....	183
A.13. Conformance Class “GeoJSON” .....	185
A.14. Conformance Class “SensorML” .....	191
<b>ANNEX B (INFORMATIVE) EXAMPLES .....</b>	<b>199</b>
<b>ANNEX C (INFORMATIVE) RELATIONSHIP WITH OTHER OGC/ISO STANDARDS</b>	
<b>(INFORMATIVE) .....</b>	<b>201</b>
C.1. W3C Semantic Sensor Network Ontology .....	201
C.2. OGC Sensor Modeling Language (SensorML) Standard .....	202
C.3. OGC/ISO Observations, Measurements and Samples (OMS) Standard .....	203
C.4. IETF GeoJSON .....	204
C.5. OGC Features and Geometries JSON (JSON-FG) .....	204
C.6. OGC API – Features Standard .....	204
C.7. OGC API – Moving Features Standard .....	205
C.8. OGC API – Environmental Data Retrieval (EDR) Standard .....	205
C.9. OGC SensorThings API Standard .....	206
C.10. Coverages .....	208
C.11. 3D Features .....	208
C.12. OGC Sensor Observation Service (SOS) Standard .....	209
C.13. OGC Sensor Planning Service (SPS) Standard .....	210
<b>ANNEX D (INFORMATIVE) REVISION HISTORY .....</b>	<b>212</b>
<b>BIBLIOGRAPHY .....</b>	<b>214</b>

## LIST OF TABLES

---

Table 1 – Overview of resource types defined by this Standard .....	21
Table 2 – Query Parameters .....	26

Table 3 – Link Relation Types .....	26
Table 4 – System Attributes .....	37
Table 5 – System Associations .....	37
Table 6 – System Types .....	38
Table 7 – Asset Types .....	38
Table 8 – Subsystem Associations .....	46
Table 9 – System Associations .....	48
Table 10 – Deployment Attributes .....	51
Table 11 – Deployment Associations .....	52
Table 12 – Subdeployment Associations .....	58
Table 13 – Deployment Associations .....	61
Table 14 – Procedure Attributes .....	64
Table 15 – Procedure Associations .....	65
Table 16 – Procedure Types .....	65
Table 17 – Common Sampling Feature Attributes .....	73
Table 18 – Sampling Features Associations .....	73
Table 19 – Property Definition Attributes .....	80
Table 20 – GeoJSON Mappings of Common Attributes .....	119
Table 21 – GeoJSON Encoding of System Attributes .....	120
Table 22 – GeoJSON Encoding of System Associations .....	120
Table 23 – GeoJSON Encoding of Deployment Attributes .....	122
Table 24 – GeoJSON Encoding of Deployment Associations .....	122
Table 25 – GeoJSON Encoding of Procedure Attributes .....	124
Table 26 – GeoJSON Encoding of Procedure Associations .....	124
Table 27 – GeoJSON Encoding of Sampling Feature Attributes .....	126
Table 28 – GeoJSON Encoding of Sampling Feature Associations .....	126
Table 29 – SensorML Mappings of Common Attributes .....	130
Table 30 – SensorML Mappings of System Attributes .....	131
Table 31 – SensorML Mappings of System Associations .....	132
Table 32 – SensorML Mappings of Deployment Attributes .....	134
Table 33 – SensorML Mappings of Deployment Associations .....	134
Table 34 – SensorML Mappings of Procedure Attributes .....	137
Table 35 – SensorML Mappings of Procedure Associations .....	137
Table 36 – SensorML Mappings of Property Attributes .....	140

## LIST OF FIGURES

Figure 1 – Class diagram of API resources .....	21
---	----

# LIST OF RECOMMENDATIONS

---

- REQUIREMENTS CLASS 1 ..... 30
- REQUIREMENTS CLASS 2 ..... 35
- REQUIREMENTS CLASS 3 ..... 44
- REQUIREMENTS CLASS 4 ..... 50
- REQUIREMENTS CLASS 5 ..... 57
- REQUIREMENTS CLASS 6 ..... 63
- REQUIREMENTS CLASS 7 ..... 70
- REQUIREMENTS CLASS 8 ..... 79
- REQUIREMENTS CLASS 9 ..... 85
- REQUIREMENTS CLASS 10 ..... 105
- REQUIREMENTS CLASS 11 ..... 113
- REQUIREMENTS CLASS 12 ..... 117
- REQUIREMENTS CLASS 13 ..... 128
- REQUIREMENT 1 ..... 31
- REQUIREMENT 2 ..... 31
- REQUIREMENT 3 ..... 32
- REQUIREMENT 4 ..... 39
- REQUIREMENT 5 ..... 40
- REQUIREMENT 6 ..... 40
- REQUIREMENT 7 ..... 41
- REQUIREMENT 8 ..... 42
- REQUIREMENT 9 ..... 46
- REQUIREMENT 10 ..... 47
- REQUIREMENT 11 ..... 47
- REQUIREMENT 12 ..... 48
- REQUIREMENT 13 ..... 48
- REQUIREMENT 14 ..... 53
- REQUIREMENT 15 ..... 53
- REQUIREMENT 16 ..... 54
- REQUIREMENT 17 ..... 54
- REQUIREMENT 18 ..... 55

REQUIREMENT 19 .....	58
REQUIREMENT 20 .....	59
REQUIREMENT 21 .....	59
REQUIREMENT 22 .....	60
REQUIREMENT 23 .....	60
REQUIREMENT 24 .....	66
REQUIREMENT 25 .....	66
REQUIREMENT 26 .....	67
REQUIREMENT 27 .....	67
REQUIREMENT 28 .....	68
REQUIREMENT 29 .....	74
REQUIREMENT 30 .....	74
REQUIREMENT 31 .....	75
REQUIREMENT 32 .....	75
REQUIREMENT 33 .....	76
REQUIREMENT 34 .....	81
REQUIREMENT 35 .....	81
REQUIREMENT 36 .....	82
REQUIREMENT 37 .....	83
REQUIREMENT 38 .....	86
REQUIREMENT 39 .....	87
REQUIREMENT 40 .....	87
REQUIREMENT 41 .....	89
REQUIREMENT 42 .....	90
REQUIREMENT 43 .....	91
REQUIREMENT 44 .....	91
REQUIREMENT 45 .....	92
REQUIREMENT 46 .....	93
REQUIREMENT 47 .....	94
REQUIREMENT 48 .....	94
REQUIREMENT 49 .....	95
REQUIREMENT 50 .....	96
REQUIREMENT 51 .....	96

REQUIREMENT 52 .....	97
REQUIREMENT 53 .....	98
REQUIREMENT 54 .....	99
REQUIREMENT 55 .....	99
REQUIREMENT 56 .....	100
REQUIREMENT 57 .....	101
REQUIREMENT 58 .....	101
REQUIREMENT 59 .....	102
REQUIREMENT 60 .....	106
REQUIREMENT 61 .....	106
REQUIREMENT 62 .....	107
REQUIREMENT 63 .....	107
REQUIREMENT 64 .....	108
REQUIREMENT 65 .....	108
REQUIREMENT 66 .....	109
REQUIREMENT 67 .....	110
REQUIREMENT 68 .....	110
REQUIREMENT 69 .....	110
REQUIREMENT 70 .....	111
REQUIREMENT 71 .....	111
REQUIREMENT 72 .....	113
REQUIREMENT 73 .....	114
REQUIREMENT 74 .....	114
REQUIREMENT 75 .....	115
REQUIREMENT 76 .....	115
REQUIREMENT 77 .....	118
REQUIREMENT 78 .....	118
REQUIREMENT 79 .....	118
REQUIREMENT 80 .....	119
REQUIREMENT 81 .....	119
REQUIREMENT 82 .....	119
REQUIREMENT 83 .....	121
REQUIREMENT 84 .....	121

REQUIREMENT 85 .....	124
REQUIREMENT 86 .....	124
REQUIREMENT 87 .....	125
REQUIREMENT 88 .....	125
REQUIREMENT 89 .....	129
REQUIREMENT 90 .....	129
REQUIREMENT 91 .....	129
REQUIREMENT 92 .....	130
REQUIREMENT 93 .....	130
REQUIREMENT 94 .....	131
REQUIREMENT 95 .....	131
REQUIREMENT 96 .....	131
REQUIREMENT 97 .....	133
REQUIREMENT 98 .....	133
REQUIREMENT 99 .....	136
REQUIREMENT 100 .....	137
REQUIREMENT 101 .....	137
REQUIREMENT 102 .....	139
REQUIREMENT 103 .....	139
RECOMMENDATION 1 .....	32
RECOMMENDATION 2 .....	39
RECOMMENDATION 3 .....	88
RECOMMENDATION 4 .....	102
RECOMMENDATION 5 .....	102
CONFORMANCE CLASS A.1 .....	143
CONFORMANCE CLASS A.2 .....	145
CONFORMANCE CLASS A.3 .....	147
CONFORMANCE CLASS A.4 .....	150
CONFORMANCE CLASS A.5 .....	153
CONFORMANCE CLASS A.6 .....	156
CONFORMANCE CLASS A.7 .....	159
CONFORMANCE CLASS A.8 .....	161
CONFORMANCE CLASS A.9 .....	163

CONFORMANCE CLASS A.10 .....	177
CONFORMANCE CLASS A.11 .....	183
CONFORMANCE CLASS A.12 .....	185
CONFORMANCE CLASS A.13 .....	191



# 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. The OGC API – Connected Systems Standard (aka “this Standard” or “CS API”) 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 Standard, including the systems themselves, are also features, the OGC API – Connected Systems Standard is logically written as an extension of the OGC API – Features Standard (Parts 1 and 4).

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/actuators 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 resources defined by other OGC Standards, such as 3D Tiles, Coverages, EDR, SensorThings, Processes, and other instances of OGC API – Features. Among other things, this linking capability enables retrieving 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 CS API is comprised of multiple parts, each of them being a separate standard.

“Part 1 – Feature Resources” (this Part) defines resource types and encodings for providing metadata about systems and their deployments, as well as the procedures and sampling strategies used by these systems. Resource types defined in Part 1 are modeled on concepts from the Semantic Sensor Network Ontology (SOSA/SSN). They are all feature types except for the `Property` resource that is used to describe feature properties. Part 1 also defines additional filtering capabilities and requirements for the Create/Replace/Delete/Update operations.

“Part 2 – Dynamic Data” defines additional resource types and encodings that implement the SSN concepts needed for exchanging dynamic data related to the features defined in Part 1. It defines efficient ways of encoding this dynamic (time-varying) information (including observations, commands and system events), and mechanisms allowing bi-directional streaming of real-time data as well as access to historical data. Part 2 also defines a snapshot mechanism for dynamic feature properties.

Other parts will be developed to define additional functionality such as pub/sub protocols, binary encodings, and concrete sampling feature types.



## 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 1: Feature 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

---

Security considerations are detailed in Clause 7.10.

# 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 Initiative for Geospatial Open Source Software 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:

NAME	AFFILIATION
Alex Robin (Editor)	GeoRobotix, Inc.
Christian Autermann	52° North Initiative
Chuck Heazel	Heazeltech (for NGA)
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.

1

# SCOPE

---

The CS API Standard defines extensions to the OGC API – Features Standard for exposing metadata regarding all kinds of observing systems and associated resources. The CS API provides an actionable implementation of concepts defined in the Semantic Sensor Network Ontology (SOSA/SSN) and also complies with OGC API – Common.

More specifically, Part 1 of the CS API Standard provides an implementation of some of the SOSA/SSN concepts (namely Systems, Platforms, Sensors, Actuators, Samplers, Procedures, Deployments and Samples) as regular features, thus making them serializable to GeoJSON format, and easily accessible by any feature API client. Advanced SensorML encodings are also defined by this standard, allowing the provision of more advanced metadata (such as complete system datasheets).

The following types of resources are defined by Part 1 of the CS API Standard:

- **Systems** are features that represent instances of observing systems, platforms, sensors, actuators and samplers.
- **Deployments** are features that provide information about the deployment of one or more Systems. They typically have a temporal and spatial extent.
- **Procedures** are non-spatial features describing the procedure implemented by one or more System instances (e.g. specs/datasheets and methodologies).
- **Sampling Features** are features used to describe the sampling geometry and/or methodology of a given observing System.
- **Property Definitions** provide semantic information about feature properties, which can be observable properties, controllable properties or simply asserted properties (e.g. certain system characteristics and capabilities).

2

# CONFORMANCE

---

## 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 “System Features” defines requirements for System resources.
- Clause 10, Requirements Class “Subsystems” defines requirements for Subsystem resources.
- Clause 11, Requirements Class “Deployment Features” defines requirements for Deployment resources.
- Clause 12, Requirements Class “Subdeployments” defines requirements for Subdeployment resources.
- Clause 13, Requirements Class “Procedure Features” defines requirements for Procedure resources.
- Clause 14, Requirements Class “Sampling Features” defines requirements for Sampling Feature resources.
- Clause 15, Requirements Class “Property Definitions” defines requirements for Property resources.
- Clause 16, Requirements Class “Advanced Filtering” defines requirements for additional filters that can be used to query *CS resources*\*.
- Clause 17, Requirements Class “Create/Replace/Delete” defines requirements for creating, replacing, and deleting *CS resources*\*.
- Clause 18, Requirements Class “Update” defines requirements for updating *CS resources*\*.

- Clause 19.1, Requirements Class “GeoJSON Format” defines requirements for encoding *CS resources\** as GeoJSON.
- Clause 19.2, Requirements Class “SensorML Format” defines requirements for encoding *CS resources\** as SensorML-JSON.

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 the resource types defined in Clauses 9, 13, 11, 14, and 15.

3

# NORMATIVE REFERENCES

---

## 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).

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>.

Clements Portele, Panagiotis (Peter) A. Vretanos: OGC 18-058, *OGC API – Features – Part 2: Coordinate Reference Systems by Reference*. Open Geospatial Consortium (2020). <http://www.opengis.net/doc/IS/ogcapi-features-2/1.0.0>.

OGC API – Features – Part 4: Create, Replace, Update and Delete, version 1.0.0-DRAFT. <https://docs.ogc.org/DRAFTS/20-002.html>

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>

OGC SensorML Encoding Standard, version 3.0, <https://docs.ogc.org/DRAFTS/23-000.html>

John Herring: OGC 06-103r4, *OpenGIS Implementation Specification for Geographic information – Simple feature access – Part 1: Common architecture*. Open Geospatial Consortium (2011). <http://www.opengis.net/doc/is/sfa/1.2.1>.

ISO: ISO 8601:2019, *Date and time – Representations for information interchange – Part 1: Basic rules*. International Organization for Standardization, Geneva (2019). .. ISO (2019).

ISO: ISO 8601:2019, *Date and time – Representations for information interchange – Part 2: Extensions*. International Organization for Standardization, Geneva (2019). .. 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>.

H. Butler, M. Daly, A. Doyle, S. Gillies, S. Hagen, T. Schaub: IETF RFC 7946, *The GeoJSON Format*. RFC Publisher (2016). <https://www.rfc-editor.org/info/rfc7946>.

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>



4

# TERMS AND DEFINITIONS

---

## 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. Actuator

---

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

[SOURCE: SOSA-SSN, [Actuator Class](#)]

### 4.3. Connected Systems

---

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

## 4.4. Deployment

---

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

[SOURCE: SOSA/SSN, [Deployment Class](#)]

## 4.5. Feature

---

Abstraction of real-world phenomena.

**Note 1 to entry:** More details about the term 'feature' may be found in the W3C/OGC Spatial Data on the Web Best Practice in the section 'Spatial Things, Features and Geometry'.

[SOURCE: ISO-19101, definition 4.11]

## 4.6. Feature Collection

---

A set of features from a dataset.

[SOURCE: OGC API – Features, definition 4.1.4]

## 4.7. 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, [FeatureOfInterest Class](#)]

## 4.8. Observation

---

Act of observing a property

[SOURCE: ISO-19156, definition 4.10]

## 4.9. Platform

---

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

[SOURCE: SOSA/SSN, [Platform Class](#)]

## 4.10. 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, [Procedure Class](#)]

## 4.11. 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]

## 4.12. Sample

---

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

[SOURCE: SOSA/SSN, [Sample Class](#)]

## 4.13. Sampler

---

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

[SOURCE: SOSA/SSN, [Sampler Class](#)]

## 4.14. 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.

## 4.15. 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, [Sensor Class](#)]

## 4.16. Sensor Network

---

A collection of sensors and processing nodes, in which information on properties observed by the sensors may be transferred and processed.

Note: A particular type of a sensor network is an ad-hoc sensor network.

## 4.17. 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, [System Class](#)]

6

# CONVENTIONS

---

## 6.1. Identifiers

---

The normative provisions in this standard are denoted by the URI

<http://www.opengis.net/spec/ogcapi-connectedsystems-1/1.0>

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

## 6.2. URL Templates

---

URL template notation is used in various places in this document.

Fixed parts of the URL are written as plain string while the variable parts of the URL (i.e. the parameters) are surrounded by curly brackets { }.

In particular, the following parameters are used in various places:

- {api\_root} denotes the base URL of the API, which corresponds to the landing page
- {id} denotes the local ID of a resource

## 6.3. Abbreviated terms

---

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

- API: Application Programming Interface
- CPU: Central Processing Unit
- CRS: Coordinate Reference System
- CSML: Climate Science Modeling Language
- ENU: East North Up
- GPS: Global Positioning System

- HTTP: Hypertext Transfer Protocol
- ISO: International Organization for Standardization
- LTP: Local Tangent Plane
- M2M: Machine to Machine
- MISB: Motion Imagery Standards Board
- NED: North East Down
- OGC: Open Geospatial Consortium
- O&M: Observations and Measurements
- OMS: Observations, Measurements and Samples
- RBAC: Role Based Access Control
- 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)
- UAS: Unmanned Aerial System
- UAV: Unmanned Aerial Vehicle
- UGV: Unmanned Ground Vehicle
- UID: Unique Identifier
- USV: Unmanned Surface Vehicle
- UUV: Unmanned Underwater Vehicle
- UML: Unified Modeling Language
- URL: Uniform Resource Locator
- URI: Uniform Resource Identifier
- UTC: Coordinated Universal Time
- WKT: Well-Known Text

- XML: eXtended Markup Language
- 1D: One Dimensional
- 2D: Two Dimensional
- 3D: Three Dimensional

7

# OVERVIEW

---

## 7.1. General

---

OGC Web API Standards enable access to resources using the HTTP protocol and its associated operations (GET, PUT, POST, DELETE, etc.).

OGC API – Connected Systems Standard – Part 1 (aka “this Standard” or “CS API”) defines resource types that allow the provision of metadata 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.

This Standard is an extension of the OGC API – Features Standard and defines specific feature collections, feature types and filtering mechanisms that are also dependent on building blocks from the OGC API – Common Standard. Therefore, an implementation of this Standard shall first satisfy the appropriate Requirements Classes from these two Standards. In addition, this Standard has dependencies on some OGC and non-OGC encoding standards. All dependencies are clearly identified in each Requirements Class.

## 7.2. Design Considerations

---

While this is the first version of the OGC API – Connected Systems series, the fine-grained access to sensor related data, including sensor metadata, observations and tasking over the Web has been supported by the OGC Sensor Observation Service (SOS), OGC Sensor Planning Service (SPS) and SensorThings API Standards and their various implementations for many years.

SOS and SPS were designed in the early 2000s and use a Remote-Procedure-Call-over-HTTP architectural style and XML for any payloads, while the SensorThings API Standard is a newer OGC Standard that was the first to adopt the REST architecture style with JSON payloads.

Requirements in the OGC API – Connected Systems Standard (CS API) support all capabilities from these previous Standards, but using a modernized approach that follows the current Web architecture and in particular the W3C/OGC best practices for sharing Spatial Data on the Web as well as the latest OGC API guidelines.

The CS API is designed as an extension of the OGC API – Features Standard which makes it entirely compatible with Features API clients, while still allowing more advanced functionality to access dynamic data associated to features. A clear goal of this approach is to better integrate the GIS and sensor/IoT communities.

Another key design decision was to allow linking to implementations of other OGC API Standards whenever possible, thus allowing a much better integration with the rest of the OGC API ecosystem.

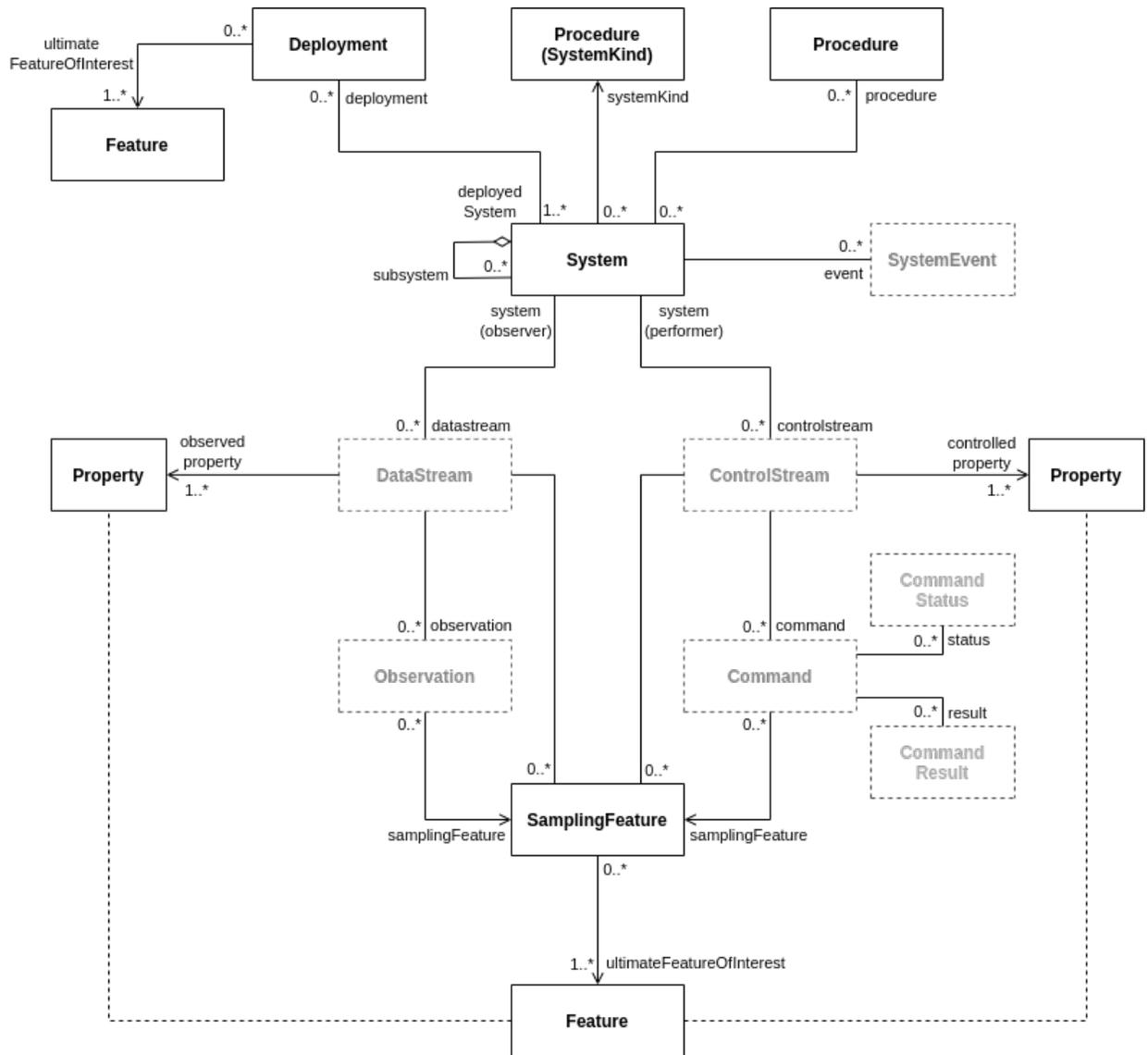
Models from the Semantic Sensor Network Ontology (SOSA/SSN) are the basis for the design of the CS API, and the SensorML language is used as an implementation model to provide concrete implementations of the SOSA/SSN concepts. An alternative GeoJSON encoding of these concepts is also defined.

The CS API defines several resource/feature types in separate requirements classes, any combination of which can be implemented by the server. This allows resources to be distributed across several servers (potentially different implementations of the CS API) and connected via hyperlinks. For example, a given implementation can choose to implement only `System` and `Deployment` features and rely on other servers to host complementary metadata such as `Procedures`, `Property Definitions` and `Features of Interest` (i.e. domain features). This is a common use case since procedure descriptions (i.e. system datasheets) and domain features (e.g. geographical or hydrological features) can typically be shared by many different organizations.

## 7.3. Resource Types

---

Figure 1 shows a UML class diagram of the Connected Systems API resources (Part 2 resources are shown with a dashed outline).



**Figure 1** – Class diagram of API resources

All resources defined in Part 1 of this Standard are feature types, except for the Property Definition resource. Each resource type is defined in its own requirements class. The table below provides an overview of these resource types:

**Table 1** – Overview of resource types defined by this Standard

RESOURCE TYPE	REQUIREMENTS CLASS	DESCRIPTION	POSSIBLE ENCODINGS
System (Feature)	Clause 9	Description of system instances such as sensors, platforms, human observers, etc.	GeoJSON, SML-JSON
Deployment (Feature)	Clause 11	Description of deployments involving one or more systems for a particular purpose.	GeoJSON, SML-JSON

RESOURCE TYPE	REQUIREMENTS CLASS	DESCRIPTION	POSSIBLE ENCODINGS
Procedure (Feature)	Clause 13	Description of procedures implemented by systems such as datasheets or methods (e.g. system types).	GeoJSON, SML-JSON
Sampling Feature	Clause 14	Description of sampling strategies associated with specific systems (e.g. sampling geometry or method).	GeoJSON
Property Definition	Clause 15	Description of feature properties (observable properties, controllable properties, system properties).	SML-JSON

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

## 7.4. Resource Encodings

This Standard also defines encodings that can be used to encode the resource types listed above. Support for these encodings is not required as each encoding is specified in its own requirements class.

Encodings are specified in Clause 19 of this Standard. Each encoding requirements class provides schemas and examples for the supported resource types.

## 7.5. Resource Collections

The CS API Standard defines several resource types that are intended to be offered by the server via separate collections. These resource collections are governed by requirements from `ogcapi-features-1`. Such collections will be referred to as OGC API Collections in the rest of this document.

The CS API makes use of this collection mechanism to allow a server to provide a more organized view of its content, by grouping them into logical collections according to any criteria.

Note that a given resource can be made available through more than one collection endpoints (i.e. collections can overlap). This provides great flexibility for a server to organize resources according to multiple criteria and thus provide different view points to the client simultaneously. See the requirements classes of the different resource types for examples.

When exposing resources via a collection endpoint, the server must indicate the type of the items contained in the collection. This is done using the `itemType` attribute of the collection. For feature collections, the `itemType` is always set to `feature`, so this Standard defines the `featureType` attribute to further specify the type of features contained in the collection.

Heterogeneous feature collections (i.e. collections containing a mix of feature types) are allowed but this Standard does not define the behavior of such collections.

## 7.6. API Endpoints

---

As described in the previous clause, all resources defined in the CS API Standard are available through collection endpoints.

However, the CS API also defines different types of endpoints that are useful for the following use cases:

- Providing a canonical URL for a resource, independently of the collections it is part of,
- Providing a canonical endpoint to add resources of a given type, independently of the collections it will be added to,
- Searching for resources across all collections of a given resource type,
- Access resources as sub-resources of a parent resource (allows to provide a pre-filtered view of the resources)

### 7.6.1. Endpoint Types

The CS API Standard defines the behavior of “resources endpoints” and “resource endpoints” associated to each resource type defined in the Standard. These endpoints are defined in a way that is independent of the actual endpoint URL so that the same behavior can be reused at different API paths.

The terms “resources endpoint” and “resource endpoint” are used as defined by OGC API – Features – Part 4: Create, Replace, Update and Delete, that is:

- A “resource endpoint” is an API endpoint exposing a single resource.
- A “resources endpoint” is an API endpoint exposing a set of resources.

Several types of API endpoints are defined by the CS API Standard:

- Canonical resources endpoints (e.g. {api\_root}/systems)
- Canonical resource endpoints (e.g. {api\_root}/systems/{id})
- Nested resources endpoints (e.g. {api\_root}/systems/{id}/subsystems)
- Collection items resources endpoints (e.g. {api\_root}/collections/{id}/items)

## 7.6.2. Canonical Resources Endpoints

A canonical resources endpoint exposes all resources of a given type hosted by the server. It provides a default endpoint for creating new resources (using HTTP POST), and retrieving/searching resources (using HTTP GET) of this type. Canonical resources endpoint have simple URLs located directly at the API root.

The canonical resources endpoints for resource types defined in Part 1 of the CS API Standard are:

- `{api_root}/systems`
- `{api_root}/deployments`
- `{api_root}/procedures`
- `{api_root}/samplingFeatures`
- `{api_root}/properties`

## 7.6.3. Canonical Resource Endpoints

A canonical resource endpoint exposes a single resource. It provides a default endpoint for retrieving, replacing, updating or deleting (using HTTP GET, PUT, PATCH and DELETE, respectively) a given resource. Any change to the resource made at its canonical endpoint will be reflected in all collections that the resource is part of.

The canonical URL for a single resource is based on the URL of the canonical resources endpoint of the corresponding resource type. This leads to the following canonical URL templates for resource types defined in Part 1 of the CS API Standard:

- `{api_root}/systems/{id}`
- `{api_root}/deployments/{id}`
- `{api_root}/procedures/{id}`
- `{api_root}/samplingFeatures/{id}`
- `{api_root}/properties/{id}`

When a resource is retrieved from a URL that is NOT its canonical URL (e.g. through a collection), its canonical URL must be provided in the response.

An example of canonical link is provided in the following JSON snippet:

```
{
  "type": "Feature",
  "id": "123",
  ...
}
```

```

"links": [
  {
    "rel" : "self",
    "title" : "this document",
    "href" : "https://data.example.org/api/collections/uav_systems/123?f=
json",
    "type" : "application/geo+json"
  }, {
    "rel" : "canonical",
    "title" : "this resource canonical URL",
    "href" : "https://data.example.org/api/systems/123?f=json",
    "type" : "application/geo+json"
  }
]
}

```

**NOTE:** If the response format is not JSON based, the canonical link can still be provided in the HTTP response headers.

## 7.7. Paged Responses

---

All resource collections support paging via the `limit` query parameter and the next link, as specified by the OGC API – Features – Part 1: Core Standard.

## 7.8. Search & Filtering

---

The core search capability is based on the OGC API – Features – Part 1: Core Standard and thus supports:

- Bounding box searches using the `bbox` parameter,
- Time instant or time period searches using the `datetime` parameter,
- Equality predicates on feature properties (i.e. `property=value`).

The CS API Standard extends these core search capabilities to include:

- Search by resource local ID or UID using the `id` parameter.
- Geospatial searches using the `geom` parameter encoded as a WKT geometry,
- Full-text searches using the `q` parameter (prefix search only).

Additional filters are defined on a per resource type basis, as shown in the following table:

**Table 2** – Query Parameters

REQUIREMENTS CLASS	QUERY PARAMETERS
System Features	parent, procedure, foi, observedProperty, controlledProperty
Deployment Features	parent, system, foi, observedProperty, controlledProperty
Procedure Features	observedProperty, controlledProperty
Sampling Features	foi, observedProperty, controlledProperty
Property Definitions	baseProperty, objectType

See Clause 16, Requirements Class “Advanced Filtering” for more details.

## 7.9. Link Relation Types

The following link relation types are defined and used in this Standard:

**Table 3** – Link Relation Types

RELATION TYPE	USED IN RESOURCE	DESCRIPTION
ogc-rel:parentSystem	System (Subsystem), Sampling Feature	Link to the parent system of the entity.
ogc-rel:subsystems	System	Link to the subsystems of a parent system.
ogc-rel:samplingFeatures	System, Deployment	Link to the sampling features associated to the entity.
ogc-rel:deployments	System	Link to the deployments associated to the entity.
ogc-rel:procedures	System	Link to the procedures that can be implemented by a system.
ogc-rel:parentDeployment	Deployment (Subdeployment)	Link to the parent deployment of a subdeployment.
ogc-rel:subdeployments	Deployment	Link to the subdeployments of a parent deployment.
ogc-rel:featuresOfInterest	System, Deployment	Link to the ultimate features of interest associated to the entity.

RELATION TYPE	USED IN RESOURCE	DESCRIPTION
ogc-rel: implementingSystem	Procedure	Link to the systems that implement the procedure.
ogc-rel: sampledFeature	Sampling Feature	Link to the the ultimate feature of interest sampled by the sampling feature.
ogc-rel: sampleOf	Sampling Feature	Link to other sampling features that the sampling feature is a sample of.
ogc-rel: datastreams	System, Deployment, Sampling Feature	Link to the datastreams that are associated to the entity,
ogc-rel: controlStreams	System, Deployment, Sampling Feature	Link to the controlstreams that are associated to the entity,

## 7.10. Security Considerations

### 7.10.1. Authentication

The expectation is that certain functionality of the CS API will be protected by an access control mechanism (e.g. RBAC), which requires each user to authenticate.

This Standard does not mandate a particular authentication method, but the following methods are commonly used and supported by OpenAPI:

- HTTP authentication (basic, bearer),
- API key (either as a header or as a query parameter),
- OAuth2 Common Flows (implicit, password, application and access code) as defined in RFC6749, and
- OpenID Connect Discovery.

NOTE: Some of these authentication methods are only recommended over HTTPS.

### 7.10.2. Encryption

A CS API implementation will often be used to transmit confidential or sensitive data. Encryption in-transit using HTTPS (i.e. HTTP over TLS/SSL) is thus highly recommended and is now very common practice on the web.

In addition, implementations of this Standard may also store confidential or sensitive data (e.g. in a database) for extended periods of time. In this case, encryption at rest is also recommended, especially if data is hosted on a shared infrastructure (e.g. public clouds).

### **7.10.3. M2M Communications**

It is expected that clients implementing the CS API Standard will sometime be machines that connect to the API automatically without human intervention.

To mitigate data spoofing, it is highly recommended that this type of clients use a strong authentication method and digital signatures relying on asymmetric cryptography, and whose access can be easily revoked (e.g. PKI certificates).

### **7.10.4. Common Weaknesses**

Please see [Clause 11](#) of OGC API – Features – Part 1: Core for guidance regarding the mitigation of typical web APIs weaknesses.

8

# REQUIREMENTS CLASS “COMMON”

---

## 8.1. Overview

REQUIREMENTS CLASS 1	
IDENTIFIER	/req/api-common
TARGET TYPE	Web API
CONFORMANCE CLASS	Conformance class A.1: /conf/api-common
PREREQUISITES	<a href="http://www.opengis.net/spec/ogcapi-features-1/1.0/req/core">http://www.opengis.net/spec/ogcapi-features-1/1.0/req/core</a> <a href="http://www.opengis.net/spec/ogcapi-common-1/1.0/req/core">http://www.opengis.net/spec/ogcapi-common-1/1.0/req/core</a> <a href="http://www.opengis.net/spec/ogcapi-common-1/1.0/req/landing-page">http://www.opengis.net/spec/ogcapi-common-1/1.0/req/landing-page</a> <a href="http://www.opengis.net/spec/ogcapi-common-1/1.0/req/json">http://www.opengis.net/spec/ogcapi-common-1/1.0/req/json</a>
NORMATIVE STATEMENTS	Requirement 1: /req/api-common/resource-ids Requirement 2: /req/api-common/resource-uids Recommendation 1: /rec/api-common/resource-uids-types Requirement 3: /req/api-common/datetime

This requirements class regroups core dependencies that all other requirements class inherit. It also provides clarifications on the use of the OGC API – Common Standard constructs.

## 8.2. API Landing Page

The landing page provides links to start exploration of the resources offered by an OGC API implementation instance. The OGC API – Common Standard already requires some common links, sufficient for this Standard.

## 8.3. API Definition

Every OGC API implementation instance is required to provide a definition document that describes the capabilities of that instance. This definition document can be used by

developers to understand the API capabilities, by software clients to connect to the server, or by development tools to support the implementation of servers and clients.

Implementation must comply with requirements defined in the OGC API – Common Standard when generating the API definition document.

## 8.4. Resource IDs

Resource IDs are typically generated by the server and are not guaranteed to be globally unique. However, the server must ensure that IDs are unique within a given resource type and/or feature type.

### REQUIREMENT 1

**IDENTIFIER** /req/api-common/resource-ids

**INCLUDED IN** Requirements class 1: /req/api-common

**STATEMENT** The server SHALL ensure that resource IDs are unique across all resources of a given type (i.e. across all collections containing resources of that type).

## 8.5. Unique Identifiers (UID)

This Standard mandates that some resource types (e.g. feature resources) have a globally unique identifier that is independent of the resource URL. This is needed to carry the identity of the real-world object that a resource represents across services, as multiple servers may host different representations/descriptions of the same object.

### Example

A Connected Systems API server implementation may hold the summary representation of a building, while the feature API server of the land register contains its 2D footprint, and the emergency response server contains its detailed 3D structure. All 3 representations should refer to the same identifier so they can be related to each other.

### REQUIREMENT 2

**IDENTIFIER** /req/api-common/resource-uids

**INCLUDED IN** Requirements class 1: /req/api-common

## REQUIREMENT 2

**A** The server SHALL ensure that resource UIDs (Unique IDs) are unique across all its collections.

**B** The server SHALL ensure that all resource UIDs are valid URIs.

## RECOMMENDATION 1

**IDENTIFIER** /rec/api-common/resource-uids-types

**INCLUDED IN** Requirements class 1: /req/api-common

The server SHOULD ensure that resource UIDs are globally unique. The recommended URI types are:

- A**
- URNs for 128-bits Universally Unique Identifiers (UUID) (prefixed by urn:uuid:, see RFC4122)
  - URNs using a namespace registered with IANA

## 8.6. Coordinate Reference Systems

As the CS API Standard extends OGC API – Features – Part 1: Core, the server is only required to implement support for CRS:84 (longitude, latitude) and CRS:84h (longitude, latitude, height). However, if support for a CRS other than CRS:84 or CRS:84h is needed, the server can also implement requirements from the OGC API – Features – Part 2: Coordinate Reference Systems by Reference Standard.

## 8.7. Date/Time Query Parameter

### REQUIREMENT 3

**IDENTIFIER** /req/api-common/datetime

**INCLUDED IN** Requirements class 1: /req/api-common

**A** When the datetime query parameter is used to filter a collection of feature types defined in this Standard, the server SHALL use the validTime attribute of the features to determine their temporal extent.

## REQUIREMENT 3

**B**

Only features with a `validTime` period that intersects the value of the `datetime` query parameter, or features that don't report any temporal validity (i.e. `validTime` attribute is null or not set), SHALL be included in the result set.

9

# REQUIREMENTS CLASS “SYSTEM FEATURES”

---

## 9.1. Overview

REQUIREMENTS CLASS 2	
IDENTIFIER	/req/system
TARGET TYPE	Web API
CONFORMANCE CLASS	Conformance class A.2: /conf/system
PREREQUISITE	Requirements class 1: /req/api-common
NORMATIVE STATEMENTS	Requirement 4: /req/system/location-time Requirement 5: /req/system/canonical-url Requirement 6: /req/system/resources-endpoint Requirement 7: /req/system/canonical-endpoint Requirement 8: /req/system/collections

The “System Features” requirements class specifies how system descriptions are provided using the CS API.

The `System` resource implements the *System* concept defined in the Semantic Sensor Network Ontology (SOSA/SSN). `System` resources are used to expose metadata about several kinds of systems and their components, including sensors, actuators, samplers, processes, platforms, etc.

**NOTE 1:** `System` resources describe **system instances** (e.g. a sensor device, a UAV platform) while `Procedure` resources are used to describe **system types** (e.g. a particular model of sensor or vehicle, specified using the manufacturer datasheet). Several `System` instances can implement the same `Procedure` (i.e. be of the same model).

**NOTE 2:** `System` resources are used to model instances of *sosa:System* (including its subclasses: *sosa:Sensor*, *sosa:Actuator* and *sosa:Sampler*), as well as *sosa:Platform*. Semantic tagging is used to provide the exact type.

**NOTE 3:** The Semantic Sensor Network Ontology (SOSA/SSN) does not define `sosa:Platform` as a subclass of `sosa:System`. However, the ontology allows an individual to be both a `sosa:System` and a `sosa:Platform`. This is the specific case that is modeled in this Standard as a `System` resource tagged with `systemType=sosa:Platform`. However, not all types of platforms need be modeled as systems (i.e. trees, buildings, etc. can also serve as platforms). Thus, this Standard does not put any restriction as to what type of Features can be used as platforms. This enables

a Deployment to refer to any Feature as the platform, which includes linking to external feature stores (e.g. building databases, etc.).

### System Examples

- A digital temperature probe (type: Sensor, assetType: Equipment)
- A GPS receiver (type: Sensor, assetType: Equipment)
- A video camera (type: Sensor, assetType: Equipment)
- A weather forecasting system (type: Sensor, assetType: Simulation)
- A human bird watcher (type: Sensor, assetType: Human)
- An electric motor (type: Actuator, assetType: Equipment)
- A motorized window blind (type: Actuator, assetType: Equipment)
- A field technician collecting water samples (type: Sampler, assetType: Human)
- An unmanned vehicle (type: Platform, assetType: Equipment)
- An aircraft (type: Platform, assetType: Equipment)
- A bulldozer (type: Platform, assetType: Equipment)
- The Nexrad radar network (type: System, assetType: Group)

## 9.2. System Resource

---

### 9.2.1. Introduction

In the CS API Standard, System resources are a special kind of feature resource that implements the *ssn:System* concept.

This section defines the attributes and associations composing a System 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 resource as GeoJSON
- System resource as SensorML

## 9.2.2. Properties

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

All System resource representations provided by encoding requirements classes map to these properties.

**Table 4 – System Attributes**

NAME	SOSA/SSN PROPERTY	DEFINITION	DATA TYPE	USAGE
uniqueIdentifier	RDF concept URI	The unique identifier of the system in the form of a URI (preferably a URN). This identifier should be persistent and independent of the actual resource URL.	URI	Required
name	rdfs:label	The human readable name of the system.	String	Required
description	rdfs:comment	A human readable description for the system.	String	Optional
systemType	rdfs:type	The type of system (see Table 6).	URI or CURIE	Required
assetType	-	The type of asset involved in the system (see Table 7).	Enum	Optional
validTime	-	The validity period of the system's description.	Date Time	Optional
location	geo:lat/geo:lon	The current location of the system.	Point	Optional

**Table 5 – System Associations**

NAME	SOSA/SSN PROPERTY	DEFINITION	TARGET CONTENT	USAGE
systemKind	ssn:hasSystemKind	The description of the kind of System (e.g. datasheet).	A single Procedure resource (inline or by-reference).	Optional
subsystems	sosa:hasSubSystem	The list of subsystems (i.e. components) attached to the System, if any.	A list of System resources (inline or by-reference).	Required
samplingFeatures	-	The Sampling Features associated to the System, if any.	A list of SamplingFeature resources (inline or by-reference).	Required
deployments	sosa:hasDeployment	The Deployments that the System is part of, if any.	A list of Deployment resources (inline or by-reference).	Optional

NAME	SOSA/SSN PROPERTY	DEFINITION	TARGET CONTENT	USAGE
procedures	sosa:implements	The Procedures that can be implemented by the System, if any.	A list of Procedure resources (inline or by-reference).	Optional
datastreams	-	The DataStreams generated by the System, if any.	A list of DataStream resources (inline or by-reference).	Required
controlstream	-	The ControlStreams receiving commands for the System, if any.	A list of ControlStream resources (inline or by-reference).	Required

**Table 6 – System Types**

SYSTEM TYPE	URI	CURIE	USAGE
Sensor	<a href="http://www.w3.org/ns/sosa/Sensor">http://www.w3.org/ns/sosa/Sensor</a>	sosa:Sensor	When the system's primary activity is 'sensing' or 'observing'.
Actuator	<a href="http://www.w3.org/ns/sosa/Actuator">http://www.w3.org/ns/sosa/Actuator</a>	sosa:Actuator	When the system's primary activity is 'actuating'.
Sampler	<a href="http://www.w3.org/ns/sosa/Sampler">http://www.w3.org/ns/sosa/Sampler</a>	sosa:Sampler	When the system's primary activity is 'sampling'.
Platform	<a href="http://www.w3.org/ns/sosa/Platform">http://www.w3.org/ns/sosa/Platform</a>	sosa:Platform	When the system's primary activity is 'carrying' other systems.
System	<a href="http://www.w3.org/ns/sosa/System">http://www.w3.org/ns/sosa/System</a>	sosa:System	For all other system types.

NOTE: Tagging a System resource with a particular type only carries semantic meaning and does not imply any API functionality. All types of systems can receive commands and generate datastreams.

**Table 7 – Asset Types**

ASSET TYPE VALUE	USAGE
Equipment	The system is composed of one or more hardware devices or pieces of equipment, that interact directly with the real-world and can be either automated or manually operated (e.g. sensor or actuator device, vehicle, robot, etc.).
Human	The system consists of one or more human beings that follow well defined procedures (e.g. conducting polls or surveys, collecting samples, carrying sensors, etc.).
LivingThing	The system consists of one or more animals or other living organisms (most often used with systemType=Platform when it carries sensors).
Simulation	The system is a software algorithm that simulates or predicts the state of the real-world (e.g. weather forecasts, mathematical models, training simulations, etc.).

ASSET TYPE VALUE	USAGE
Process	The system is a process or process chain that transforms data coming from or going to other systems.
Group	The system represents a group of similar systems (e.g. sensor network, vehicle fleet, etc.). Such system usually has subsystems that provide detailed information about each member in the group.
Other	Any other type of asset not accounted for in this list. In this case, an application specific property can be used to provide the type.

NOTE: Deployments can also be used to document how different types of systems are used together. For instance, a deployment may describe how a “human” (e.g. system of type platform) uses a piece of “equipment” (e.g. system of type sensor) according to a well defined procedure.

### 9.2.3. Location

It is recommended that the System resource representation includes the location of the system. If the implementation decides to report the location, it must be its latest known location.

RECOMMENDATION 2	
IDENTIFIER	/rec/system/location
STATEMENT	A System feature resource SHOULD include the system location.

REQUIREMENT 4	
IDENTIFIER	/req/system/location-time
INCLUDED IN	Requirements class 2: /req/system
A	If the implementation chooses to report the location of a system, it SHALL be the latest known location of the system, unless a specific snapshot date/time is requested (see Part 2).

NOTE: If the system is a virtual asset, such as a simulation or process, reporting its location is not always desired or possible. If an implementation wishes to report such location, the recommendation is the following:

- If the location of the computing hardware that executes the process is known, it should be used as the feature location (e.g. the location of the datacenter).

- If the process can be run in multiple locations (e.g. distributed or redundant process), a multi-point geometry or an entire geographic area should be used as the feature location.
- If the location is unknown, the location is not set.

## 9.3. System Canonical URL

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

### REQUIREMENT 5

**IDENTIFIER** /req/system/canonical-url

**INCLUDED IN** Requirements class 2: /req/system

**A** Every System resource exposed by the server SHALL be accessible through its canonical URL of the form {api\_root}/systems/{id} where id is the local identifier of the System resource.

**B** If a System 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. System Resources Endpoints

### 9.4.1. Definition

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

### REQUIREMENT 6

**IDENTIFIER** /req/system/resources-endpoint

**INCLUDED IN** Requirements class 2: /req/system

**A** The server SHALL support the HTTP GET operation at the path associated to the System resources endpoint.

## REQUIREMENT 6

B	The operation SHALL fulfill all requirements defined in <a href="#">Clause 7.15.2 to 7.15.8</a> of OGC API – Features – Part 1: Core.
C	All features in the result set SHALL be System resources.

Clause 16.5 defines additional query parameters applicable to System resources endpoint.

### 9.4.2. Canonical System Resources Endpoint

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

## REQUIREMENT 7

**IDENTIFIER** /req/system/canonical-endpoint

**INCLUDED IN** Requirements class 2: /req/system

**A** The server SHALL expose a System resources endpoint at the path {api\_root}/systems.

**B** The endpoint SHALL expose all System resources available on the server.

## 9.5. System Feature Collections

Any number of feature collections containing System features can be available at a CS API endpoint, but the server must at least expose one. Members of System feature collections are identified with the feature type `sosa:System`.

System resources can be grouped into collections according to any arbitrary criteria, as shown in the following examples.

### Examples of System Collections

- All unmanned systems at URL {api\_root}/collections/uxs\_systems
- All UAV systems at URL {api\_root}/collections/uav\_systems
- All systems operated by organization X at URL {api\_root}/collections/orgx\_systems
- All currently deployed systems at URL {api\_root}/collections/deployed\_systems

Note that a given system can be part of all 4 collections at the same time.

REQUIREMENT 8	
<b>IDENTIFIER</b>	/req/system/collections
<b>INCLUDED IN</b>	Requirements class 2: /req/system
<b>A</b>	The server SHALL expose at least one Feature collection containing System resources.
<b>B</b>	The server SHALL identify all Feature collections containing System resources by setting the itemType attribute to feature and the featureType attribute to sosa:System in the Collection metadata.
<b>C</b>	For any feature collection with featureType set to sosa:System, the HTTP GET operation at the path /collections/{collectionId}/items SHALL support the query parameters and response of a System resources endpoint.

10

# REQUIREMENTS CLASS “SUBSYSTEMS”

---

## 10.1. Overview

### REQUIREMENTS CLASS 3

IDENTIFIER	/req/subsystem
TARGET TYPE	Web API
CONFORMANCE CLASS	Conformance class A.3: /conf/subsystem
PREREQUISITE	Requirements class 2: /req/system
NORMATIVE STATEMENTS	Requirement 9: /req/subsystem/collection Requirement 10: /req/subsystem/recursive-param Requirement 11: /req/subsystem/recursive-search-systems Requirement 12: /req/subsystem/recursive-search-subsystems Requirement 13: /req/subsystem/recursive-assoc

Describing complex systems in terms of a hierarchy of components (or subsystems) is often useful. The “Subsystems” requirements class specifies how such hierarchical systems are described and made discoverable in an implementation of the CS API Standard.

Subsystems (i.e. system components) are regular `System` features that are made available through a sub-collection of a parent `System` resource. Subsystems can be any kind of `System` and can be indefinitely nested.

## 10.2. Types of System/Subsystem Associations

Subsystems can be associated to their parent system either by **Composition** or **Aggregation**.

**Composition** means that a subsystem is an integral part of a parent system (i.e. the subsystem cannot or is not usually taken apart from its parent system during its lifetime, except for maintenance). Composition is implemented in the API by creating subsystems as nested resources under their parent system.

**Aggregation** means that a subsystem is not permanently attached to its parent system (i.e. it can be taken apart and can be attached to different parent systems at different times). Aggregation

is implemented in the API using Deployment resources that describe what subsystems are mounted on a parent system at any given time.

**NOTE:** This definition is slightly looser than the UML definition for **Composition** associations. In UML, composition is used if a child cannot exist independently from the parent. In the context of the CS API Standard, composition can be used by choice if it is known that the component is rarely taken apart from its parent, even though, in principle the component could exist independently.

#### Examples of systems modeled by composition

- A UAV with built-in IMU, GPS and camera that cannot be changed
- A weather station with built-in sensors

#### Examples of systems modeled by aggregation

- A weather station with interchangeable sensors
- A human operator using different sensors according to the mission

#### Examples of systems modeled by combining both composition and aggregation

- A UAV with IMU, GPS, flight controller (composition since they are built-in) and interchangeable payloads such as RGB camera, thermal camera, LiDAR, etc. (aggregation since they can be changed for each mission).
- A mobile phone with integrated sensors (composition since they are built-in) and additional sensors connected via Bluetooth (aggregation).

## 10.3. Subsystem Resource

---

### 10.3.1. Introduction

Resources representing subsystems are regular System resources that are nested under their parent System.

See Clause 9.2 for the requirements applying to System resources.

### 10.3.2. Properties

A System resource that is a subsystem of a parent system also includes the following associations:

**Table 8** – Subsystem Associations

NAME	SOSA/SSN PROPERTY	DEFINITION	TARGET CONTENT	USAGE
parentSystem	inverse Of sosa: hasSubSystem	The parent System that this subsystem is part of.	A single System resource.	Required

## 10.4. Subsystem Canonical URL

Since subsystems are also System resources, they are required to have a canonical URL as specified by Requirement 5: `/req/system/canonical-url`.

## 10.5. Subsystem Resources Endpoint

Subsystems are accessible as sub-resources of their parent System.

REQUIREMENT 9	
IDENTIFIER	<code>/req/subsystem/collection</code>
INCLUDED IN	Requirements class 3: <code>/req/subsystem</code>
A	The server SHALL expose a System resources endpoint at path <code>{api_root}/systems/{parentId}/subsystems</code> .
B	The endpoint SHALL only expose the System resources that are subsystems of the parent System with ID <code>parentId</code> .
C	The endpoint SHALL expose all subsystems that are permanently attached to the parent system, and CAN also expose the subsystems that are currently deployed on the parent.

Some systems have interchangeable components that are often called “payloads” (e.g. UAV platform carrying different sensors at different times). Although such payloads can be listed as subsystems by the server, the best way to describe this is by creating a new Deployment resource every time the payloads are changed (the deployment description allows one to explicitly list which payloads are mounted on a platform during a given time period).

## 10.6. System Recursive Search

By default, the canonical System resources endpoint only exposes top-level systems (i.e. subsystems are not visible).

Likewise, Subsystem resources endpointw only expose the direct subsystems, but not subsystems nested at deeper levels (i.e. the subsystems of the subsystems and so on).

The recursive query parameter changes the default behavior by instructing the server to process all subsystems recursively (at all levels below the current level).

### REQUIREMENT 10

**IDENTIFIER** /req/subsystem/recursive-param

**INCLUDED IN** Requirements class 3: /req/subsystem

**STATEMENT** The server SHALL support a query parameter recursive with the following characteristics (using an OpenAPI Specification 3.0 fragment):  
name: recursive  
in: query  
required: false  
schema:  
  type: boolean

### REQUIREMENT 11

**IDENTIFIER** /req/subsystem/recursive-search-systems

**INCLUDED IN** Requirements class 3: /req/subsystem

**A** HTTP GET operations at the canonical System resources endpoint {api\_root}/systems SHALL support the parameter recursive.

**B** If the recursive parameter is omitted or set to false, the response SHALL only include the top-level systems, and not their subsystems.

**C** If the recursive parameter is set to true, the response SHALL include top-level systems as well as their subsystems, recursively.

**D** Other query string parameters SHALL be applied to all processed System resources, regardless of whether they are top-level systems or subsystems.

## REQUIREMENT 12

**IDENTIFIER** /req/subsystem/recursive-search-subsystems

**INCLUDED IN** Requirements class 3: /req/subsystem

**A** HTTP GET operations at subsystems resources endpoints {api\_root}/systems/{id}/subsystems SHALL support the parameter recursive.

**B** If the recursive parameter is omitted or set to false, the response SHALL only include the system's direct subsystems.

**C** If the recursive parameter is set to true, the response SHALL include all nested subsystems, recursively.

**D** Other query string parameters SHALL be applied to all processed System resources, regardless of whether they are direct subsystems, or transitive subsystems.

## 10.7. System Associations

If a System has subsystems, the associations listed below must reference a resource set that includes resources associated to the main system, as well as all its subsystems.

## REQUIREMENT 13

**IDENTIFIER** /req/subsystem/recursive-assoc

**INCLUDED IN** Requirements class 3: /req/subsystem

**A** Whenever a System resource has subsystems, the target content of its associations SHALL be adjusted as indicated in Table 9.

**Table 9** – System Associations

ASSOCIATION NAME	TARGET CONTENT
samplingFeatures	The Sampling Features associated to the System and all its subsystems, recursively.
datastreams	The DataStreams generated by the System and all its subsystems, recursively.
controlstreams	The ControlStreams receiving commands for the System and all its subsystems, recursively.

11

# REQUIREMENTS CLASS “DEPLOYMENT FEATURES”

---

# REQUIREMENTS CLASS “DEPLOYMENT FEATURES”

## 11.1. Overview

REQUIREMENTS CLASS 4	
IDENTIFIER	/req/deployment
TARGET TYPE	Web API
CONFORMANCE CLASS	Conformance class A.4: /conf/deployment
PREREQUISITE	Requirements class 1: /req/api-common
NORMATIVE STATEMENTS	Requirement 14: /req/deployment/canonical-url Requirement 15: /req/deployment/resources-endpoint Requirement 16: /req/deployment/canonical-endpoint Requirement 17: /req/deployment/ref-from-system Requirement 18: /req/deployment/collections

The “Deployment Features” requirements class specifies how Deployment descriptions are provided using the CS API.

The Deployment resource implements the *Deployment* concept defined in the Semantic Sensor Network Ontology (SOSA/SSN) and is used to provide information about the deployment of one or more Systems (with or without payloads) for a specific purpose (often at a specific location and time).

### Deployment Examples

- An in-situ sensor deployed at a given location
- A network of in-situ sensors deployed along a river
- A network of security cameras deployed in a city
- A mission involving one or more unmanned systems (platforms) with payloads
- A team deployed to collect survey responses
- A sample collection campaign involving field personnel

- A forecast model run

## 11.2. Deployment Resource

### 11.2.1. Introduction

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

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

- Deployment resource as GeoJSON
- Deployment resource as SensorML

### 11.2.2. Properties

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

All Deployment resource representations provided by encoding requirements classes map to these properties.

**Table 10** – Deployment Attributes

NAME	SOSA/SSN PROPERTY	DEFINITION	DATA TYPE	USAGE
uniqueIdentif	RDF concept URI	The unique identifier of the deployment in the form of a URI (preferably a URN). This identifier should be persistent and independent of the actual resource URL.	URI	Required
name	rdfs:label	The human readable name of the deployment.	String	Required
description	rdfs:comment	A human readable description for the deployment.	String	Optional
deploymentTyp	rdfs:type	The type of deployment.	URI	Optional
validTime	-	The time period during which the systems are deployed.	Date Time	Required

NAME	SOSA/SSN PROPERTY	DEFINITION	DATA TYPE	USAGE
location	geo:lat/geo:lon	The location or area where the systems are deployed.	Geometry	Optional

NOTE 1: The deployment location is not to be confused with the features of interest or sampling features location/geometry. For in-situ and short-range remote sensors, the deployment location may include the sampling features, but it is usually not true for long range remote sensing such as space based earth observation. If the deployed systems are mobile, the deployment location should be the entire area where the systems can be moved to.

NOTE 2: For deployments of models or processes, the location would usually not be provided, and the user should query the area covered by the sampling feature(s) instead (i.e. the geographic area covered by the model). If a deployment location is provided, it should be the location where the hardware executing the process is located, not the location of the sampling features.

**Table 11 – Deployment Associations**

NAME	SOSA/SSN PROPERTY	DEFINITION	TARGET CONTENT	USAGE
platform	sosa: deployedOnPla	The platform on which the systems are deployed.	A single Feature resource (inline or by-reference).	Optional
deployedSystem	sosa: deployedSystem	The list of Systems deployed during the Deployment, if any.	A list of System resources (inline or by-reference).	Required
subdeployment -		The list of subdeployments that are part of the Deployment, if any.	A list of Deployment resources (inline or by-reference).	Required
featuresOfInterest -		The ultimate features of interest that are observed and/or controlled during the Deployment.	A list of Feature resources (inline or by-reference).	Optional
samplingFeature -		The Sampling Features associated to Systems deployed during the Deployment.	A list of Sampling Feature resources (inline or by-reference).	Optional
datastreams -		The Data Streams containing observations collected during the Deployment.	A list of DataStream resources (inline or by-reference).	Optional
controlstream -		The Control Streams that received commands issued during the Deployment.	A list of ControlStream resources (inline or by-reference).	Optional

NOTE: The platform can be another System (e.g. a UAV or a station), but can also be any feature (e.g. a building or a tree).

## 11.3. Deployment Canonical URL

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

### REQUIREMENT 14

**IDENTIFIER** /req/deployment/canonical-url

**INCLUDED IN** Requirements class 4: /req/deployment

**A** Every Deployment resource exposed by the server SHALL be accessible through its canonical URL of the form {api\_root}/deployments/{id} where id is the local identifier of the Deployment resource.

**B** If a Deployment 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. Deployment Resources Endpoints

### 11.4.1. Definition

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

### REQUIREMENT 15

**IDENTIFIER** /req/deployment/resources-endpoint

**INCLUDED IN** Requirements class 4: /req/deployment

**A** The server SHALL support the HTTP GET operation at the path associated to the Deployment resources endpoint.

**B** The operation SHALL fulfill all requirements defined in [Clause 7.15.2 to 7.15.8](#) of OGC API – Features – Part 1: Core.

**C** All features in the result set SHALL be Deployment resources.

Clause 16.6 defines additional query parameters applicable to Deployment resources endpoint.

## 11.4.2. Canonical Deployment Resources Endpoint

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

### REQUIREMENT 16

**IDENTIFIER** /req/deployment/canonical-endpoint

**INCLUDED IN** Requirements class 4: /req/deployment

**A** The server SHALL expose a Deployment resources endpoint at the path {api\_root}/deployments.

**B** The endpoint SHALL expose all Deployment resources available on the server.

## 11.4.3. Nested Deployment Resources Endpoint

### REQUIREMENT 17

**IDENTIFIER** /req/deployment/ref-from-system

**INCLUDED IN** Requirements class 4: /req/deployment

**CONDITIONS**

- The server implements the requirements class Requirements Class “System Features”
- The server provides the deployments association as part of System resource representations

**A** The deployments association in a System resource representation SHALL be implemented as a link to a Deployment resources endpoint at path {api\_root}/systems/{sysId}/deployments.

**B** The endpoint SHALL only expose the Deployment resources where the System with ID sysId was deployed.

## 11.5. Deployment Feature Collections

Any number of feature collections containing Deployment features can be available at a CS API endpoint, but the server must at least expose one. Members of Deployment feature collections are identified with the feature type `sosa:Deployment`.

Deployment resources can be grouped into collections according to any arbitrary criteria, as shown in the following examples.

### Examples of Deployment Collections

- All saildrone deployments at URL {api\_root}/collections/saildrone\_missions
- All special forces deployments at URL {api\_root}/collections/sof\_missions

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

## REQUIREMENT 18

**IDENTIFIER** /req/deployment/collections

**INCLUDED IN** Requirements class 4: /req/deployment

**A** The server SHALL expose at least one Feature collection containing Deployment resources.

**B** The server SHALL identify all Feature collections containing Deployment resources by setting the itemType attribute to feature and the featureType attribute to sosa:Deployment in the Collection metadata.

**C** For any feature collection with featureType set to sosa:Deployment, the HTTP GET operation at the path /collections/{collectionId}/items SHALL support the query parameters and response of a Deployment resources endpoint.

12

# REQUIREMENTS CLASS “SUBDEPLOYMENTS”

---

## 12.1. Overview

### REQUIREMENTS CLASS 5

IDENTIFIER	/req/subdeployment
TARGET TYPE	Web API
CONFORMANCE CLASS	Conformance class A.5: /conf/subdeployment
PREREQUISITE	Requirements class 4: /req/deployment
NORMATIVE STATEMENTS	<p>Requirement 19: /req/subdeployment/collection</p> <p>Requirement 20: /req/subdeployment/recursive-param</p> <p>Requirement 21: /req/subdeployment/recursive-search-deployments</p> <p>Requirement 22: /req/subdeployment/recursive-search-subdeployments</p> <p>Requirement 23: /req/subdeployment/recursive-assoc</p>

Describing complex deployments in terms of a hierarchy of deployments is often useful. This requirements class specifies how such hierarchical deployments are described and made discoverable using the CS API Standard.

Subdeployments are regular Deployment features that are made available through a sub-collection of a parent Deployment resource.

#### Examples of deployments with subdeployments

- A deployment of many monitoring sensors along a river, where each monitoring site is described as a subdeployment.
- A deployment of multiple unmanned systems to cover a large area divided into smaller regions, with subdeployments describing which platforms/payloads are deployed in each region.
- A tactical military operation, where each unit is described as a subdeployment.

## 12.2. Subdeployment Resource

### 12.2.1. Introduction

Resources representing subdeployments are regular Deployment resources that are nested under their parent deployment.

See Clause 11.2 for the requirements applying to Deployment resources.

### 12.2.2. Properties

A Deployment resource that is a subdeployment of a parent deployment also includes the following associations:

**Table 12** – Subdeployment Associations

NAME	SOSA/SSN PROPERTY	DEFINITION	TARGET CONTENT	USAGE
parentDeploym -		The parent Deployment that this subdeployment is part of.	A single Deployment resource.	Required

## 12.3. Subdeployment Canonical URL

Since subdeployments are also Deployment resources, they are required to have a canonical URL as specified by Requirement 14: `/req/deployment/canonical-url`.

## 12.4. Subdeployment Resources Endpoint

Subdeployments are accessible as sub-resources of their parent Deployment.

### REQUIREMENT 19

**IDENTIFIER** `/req/subdeployment/collection`

**INCLUDED IN** Requirements class 5: `/req/subdeployment`

## REQUIREMENT 19

A	The server SHALL expose subdeployments as a collection of Deployment resources at path <code>{api_root}/deployments/{parentId}/subdeployments</code> .
B	The content of the collection SHALL be the list of Deployment resources that are part of the parent Deployment with ID <code>parentId</code> .
C	The collection SHALL support the same query parameters as the ones supported by the catch-all collection <code>{api_root}/deployments</code> .

## 12.5. Deployment Recursive Search

By default, the canonical Deployment resources endpoint only exposes top-level deployments (i.e. subdeployments are not visible).

Likewise, Subdeployment resources endpoints only expose the direct subdeployments, but not subdeployments nested at deeper levels.

The recursive query parameter changes the default behavior by instructing the server to process all subdeployments recursively (at all levels below the current level).

## REQUIREMENT 20

**IDENTIFIER** `/req/subdeployment/recursive-param`

**INCLUDED IN** Requirements class 5: `/req/subdeployment`

**STATEMENT** The server SHALL support a query parameter `recursive` with the following characteristics (using an OpenAPI Specification 3.0 fragment):

```
name: recursive
in: query
required: false
schema:
  type: boolean
```

## REQUIREMENT 21

**IDENTIFIER** `/req/subdeployment/recursive-search-deployments`

**INCLUDED IN** Requirements class 5: `/req/subdeployment`

**A** HTTP GET operations at the canonical Deployment resources endpoint `{api_root}/deployments` SHALL support the parameter `recursive`.

## REQUIREMENT 21

B	If the <code>recursive</code> parameter is omitted or set to <code>false</code> , the response SHALL only include the top-level deployments, and not their subdeployments.
C	If the <code>recursive</code> parameter is set to <code>true</code> , the response SHALL include top-level deployments as well as their subdeployments, recursively.
D	Other query string parameters SHALL be applied to all processed Deployment resources, regardless of whether they are root deployments or subdeployments.

## REQUIREMENT 22

**IDENTIFIER** `/req/subdeployment/recursive-search-subdeployments`

**INCLUDED IN** Requirements class 5: `/req/subdeployment`

A	HTTP GET operations at subdeployments resources endpoints <code>{api_root}/deployments/{id}/subdeployments</code> SHALL support the parameter <code>recursive</code> .
B	If the <code>recursive</code> parameter is omitted or set to <code>false</code> , the response SHALL only include the deployment's direct subdeployments.
C	If the <code>recursive</code> parameter is set to <code>true</code> , the response SHALL include all nested subdeployments, recursively.
D	Other query string parameters SHALL be applied to all processed Deployment resources, regardless of whether they are direct subdeployments, or transitive subdeployments.

## 12.6. Deployment Associations

If a Deployment has subdeployments, the associations listed below must reference a resource set that includes resources associated to the main deployment, as well as all its subdeployments.

## REQUIREMENT 23

**IDENTIFIER** `/req/subdeployment/recursive-assoc`

**INCLUDED IN** Requirements class 5: `/req/subdeployment`

A	Whenever a Deployment resource has subdeployments, the target content of its associations SHALL be adjusted as indicated in Table 13.
---	---

**Table 13 – Deployment Associations**

ASSOCIATION NAME	TARGET CONTENT
deployedSystems	The Systems deployed during the Deployment and all its subdeployments, recursively.
samplingFeatures	The Sampling Features associated to Systems deployed during the Deployment and all its subdeployments, recursively.
featuresOfInterest	The ultimate features of interest that are observed and/or controlled during the Deployment or any of its subdeployments, recursively.
datastreams	The DataStreams generated by systems deployed in the Deployment and all its subdeployments, recursively.
controlstreams	The ControlStreams receiving commands for systems deployed in the Deployment and all its subdeployments, recursively.

13

# REQUIREMENTS CLASS “PROCEDURE FEATURES”

---

# REQUIREMENTS CLASS “PROCEDURE FEATURES”

## 13.1. Overview

REQUIREMENTS CLASS 6	
IDENTIFIER	/req/procedure
TARGET TYPE	Web API
CONFORMANCE CLASS	Conformance class A.6: /conf/procedure
PREREQUISITE	Requirements class 1: /req/api-common
NORMATIVE STATEMENTS	Requirement 24: /req/procedure/location Requirement 25: /req/procedure/canonical-url Requirement 26: /req/procedure/resources-endpoint Requirement 27: /req/procedure/canonical-endpoint Requirement 28: /req/procedure/collections

The “Procedure Features” requirements class specifies how Procedure descriptions are provided using the CS API.

A Procedure resource implements the *Procedure* concept defined in the Semantic Sensor Network Ontology (SOSA/SSN). Procedure resources are used to provide the specifications or methodology implemented by a System to accomplish its task(s).

**NOTE:** Procedure resources describe types of systems (e.g. a particular model of sensor) or procedures implemented by systems (e.g. followed by a human operator or programmed into a piece of equipment), while System resources describe system instances (e.g. a sensor device, a human observer). Several System instances can be associated to the same Procedure.

### Procedure Examples

For hardware equipment operating automatically, a procedure is used to describe the equipment’s specifications (i.e. datasheet).

For human sensing or sampling tasks, the procedure describes the steps or methodology that must be followed by the operator.

For cases involving both an instrument and a human operator, the procedure describes what instrument(s) is/are used by the operator and how. In some cases, the human acts as the Platform that carries one or more sensors (e.g. a mobile phone, a portable infrared thermometer).

## 13.2. Procedure Resource

### 13.2.1. Introduction

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

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

- Procedure resource as GeoJSON
- Procedure resource as SensorML

### 13.2.2. Properties

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

All Procedure resource representations provided by encoding requirements classes map to these properties.

**Table 14** – Procedure Attributes

NAME	SOSA/SSN PROPERTY	DEFINITION	DATA TYPE	USAGE
uniqueIdentif	RDF concept URI	The unique identifier of the procedure in the form of a URI (preferably a URN). This identifier should be persistent and independent of the actual resource URL	URI	Required
name	rdfs:label	The human readable name of the procedure	String	Required
description	rdfs:comment	A human readable description for the procedure	String	Optional
procedureType	rdfs:type	The type of procedure (see Table 16)	URI	Required

NAME	SOSA/SSN PROPERTY	DEFINITION	DATA TYPE	USAGE
validTime	-	The validity period of the procedure description.	Date Time	Optional

NOTE: The validity time period of a Procedure description starts on the date that the model of the equipment was released. No System can implement the Procedure before this date. The end date should be set to a date after which no instance of the equipment is in use, or unbounded.

**Table 15 – Procedure Associations**

NAME	SOSA/SSN PROPERTY	DEFINITION	TARGET CONTENT	USAGE
implementingS	implementedBy	The Systems that implement the Procedure.	A list of System resources (inline or by-reference).	Optional

**Table 16 – Procedure Types**

PROCEDURE TYPE	URI	CURIE	IMPLEMENTED BY	USAGE
Observing	<a href="http://www.w3.org/ns/sosa/ObservingProcedure">http://www.w3.org/ns/sosa/ObservingProcedure</a>	sosa:ObservingProcedure	Sensor	An observation method.
Actuating	<a href="http://www.w3.org/ns/sosa/ActuatingProcedure">http://www.w3.org/ns/sosa/ActuatingProcedure</a>	sosa:ActuatingProcedure	Actuator	An actuation method.
Sampling	<a href="http://www.w3.org/ns/sosa/SamplingProcedure">http://www.w3.org/ns/sosa/SamplingProcedure</a>	sosa:SamplingProcedure	Sampler	A sampling method.
Other Procedure	<a href="http://www.w3.org/ns/sosa/Procedure">http://www.w3.org/ns/sosa/Procedure</a>	sosa:Procedure	Any System	Any other type of procedure or methodology.
Sensor Kind	<a href="http://www.w3.org/ns/ssn-system/SensorKind">http://www.w3.org/ns/ssn-system/SensorKind</a>	ssn:SensorKind	Sensor	A sensor datasheet.
Actuator Kind	<a href="http://www.w3.org/ns/ssn-system/ActuatorKind">http://www.w3.org/ns/ssn-system/ActuatorKind</a>	ssn:ActuatorKind	Actuator	An actuator datasheet.
Sampler Kind	<a href="http://www.w3.org/ns/ssn-system/SamplerKind">http://www.w3.org/ns/ssn-system/SamplerKind</a>	ssn:SamplerKind	Sampler	A sampler datasheet.
Platform Kind	<a href="http://www.w3.org/ns/ssn-system/PlatformKind">http://www.w3.org/ns/ssn-system/PlatformKind</a>	ssn:PlatformKind	Platform	A platform datasheet.
Other System Kind	<a href="http://www.w3.org/ns/ssn-system/SystemKind">http://www.w3.org/ns/ssn-system/SystemKind</a>	ssn:SystemKind	Any System	Any other system datasheet.

### 13.2.3. Location

A Procedure feature represents a datasheet or a methodology. It is thus a non-spatial entity.

#### REQUIREMENT 24

**IDENTIFIER** /req/procedure/location

**INCLUDED IN** Requirements class 6: /req/procedure

**STATEMENT** A Procedure feature resource SHALL not include a location or geometry.

## 13.3. Procedure Canonical URL

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

#### REQUIREMENT 25

**IDENTIFIER** /req/procedure/canonical-url

**INCLUDED IN** Requirements class 6: /req/procedure

**A** Every Procedure resource exposed by the server SHALL be accessible through its canonical URL of the form {api\_root}/procedures/{id} where id is the local identifier of the Procedure resource.

**B** If a Procedure 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.

## 13.4. Procedure Resources Endpoints

### 13.4.1. Definition

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

## REQUIREMENT 26

**IDENTIFIER** /req/procedure/resources-endpoint

**INCLUDED IN** Requirements class 6: /req/procedure

**A** The server SHALL support the HTTP GET operation at the path associated to the Procedure resources endpoint.

**B** The operation SHALL fulfill all requirements defined in [Clause 7.15.2 to 7.15.8](#) of OGC API – Features – Part 1: Core.

**C** All features in the result set SHALL be Procedure resources.

Clause 16.7 defines additional query parameters applicable to Procedure resources endpoint.

### 13.4.2. Canonical Procedure Resources Endpoint

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

## REQUIREMENT 27

**IDENTIFIER** /req/procedure/canonical-endpoint

**INCLUDED IN** Requirements class 6: /req/procedure

**A** The server SHALL expose a Procedure resources endpoint at the path {api\_root}/procedures.

**B** The endpoint SHALL expose all Procedure resources available on the server.

## 13.5. Procedure Feature Collections

Any number of feature collections containing Procedure features can be available at a CS API endpoint, but the server must at least expose one. Members of Procedure feature collections are identified with the feature type `sosa:Procedure`.

Procedure resources can be grouped into collections according to any arbitrary criteria, as shown in the following examples.

### Examples of Procedure Collections

- All weather station datasheets at URL {api\_root}/collections/wx\_stations\_datasheets
- All bird watching procedure at URL {api\_root}/collections/bird\_watching\_procedures

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

## REQUIREMENT 28

**IDENTIFIER** /req/procedure/collections

**INCLUDED IN** Requirements class 6: /req/procedure

- A** The server SHALL expose at least one Feature collection containing Procedure resources.
- B** The server SHALL identify all Feature collections containing Procedure resources by setting the `itemType` attribute to `feature` and the `featureType` attribute to `sosa:Procedure` in the Collection metadata.
- C** For any feature collection with `featureType` set to `sosa:Procedure`, the HTTP GET operation at the path `/collections/{collectionId}/items` SHALL support the query parameters and response of a Procedure resources endpoint.

14

# REQUIREMENTS CLASS “SAMPLING FEATURES”

---

# REQUIREMENTS CLASS “SAMPLING FEATURES”

## 14.1. Overview

REQUIREMENTS CLASS 7	
IDENTIFIER	/req/sf
TARGET TYPE	Web API
CONFORMANCE CLASS	Conformance class A.7: /conf/sf
PREREQUISITES	Requirements class 1: /req/api-common Requirements class 2: /req/system
NORMATIVE STATEMENTS	Requirement 29: /req/sf/canonical-url Requirement 30: /req/sf/resources-endpoint Requirement 31: /req/sf/canonical-endpoint Requirement 32: /req/sf/ref-from-system Requirement 33: /req/sf/collections

The “Sampling Features” requirements class specifies how Sampling Feature descriptions are provided using the CS API.

A Sampling Feature resource implements the *Sample* concept defined in the SSN Ontology. Sampling Features provide information about the sampling strategy used when observing the property of a larger feature (the Feature of Interest).

By analogy, this Standard also uses the concept of Sampling Feature to more precisely identify the part of a Feature of Interest that is being affected by an actuator or process in response to a command.

While Features of Interest exist independently from any system observing or controlling them, Sampling Features are always associated to a specific System resource (which can be either a Sensor, an Actuator or a Sampler) because they define this particular system’s sampling strategy.

### Sampling Feature Examples

- A sampling point along a river
- A trajectory at the ocean surface

- A satellite image footprint
- A profile of the atmosphere
- The viewing frustum of a video camera
- The area covered by a weather radar
- A part in a complex machine

## 14.2. Features of Interest

---

In the CS API Standard, the term *Feature of Interest* is used to specifically mean the *Ultimate Feature of Interest* whose properties are observed by a sensing system or changed by a controlling system. Features of interest are real-world features from an application domain and they exist independently from any sampling strategy.

While many features of interest represent physical entities or are geospatial in nature, they can also be used to model more abstract concepts.

### Features of Interest Examples

- Man-built entities (e.g. a building, a room, a window, a road, a bridge)
- Natural bodies (e.g. the earth atmosphere, a river, a water body, an aquifer, a geological layer, a forest)
- Living organisms (e.g. a person, an animal, a tree, a cell)
- Technological systems (e.g. a vehicle, a robot, a computer, a tool, a machine)
- Conceptual things (e.g. an administrative area, a legal entity)

**NOTE 1:** A *System* feature made available through the CS API can also take the role of the *Feature of Interest* of some observations. For example, an aircraft platform modeled as a *System* using the CS API may carry a GPS sensor to measure its location. In this case, the feature of interest of the GPS observations is the platform itself because the GPS observes the location of the platform. The *Sampling Feature* would typically be a sampling point where the GPS antenna is located.

**NOTE 2:** Application domain features of interest can be hosted at the same API endpoint as the other feature types defined in this Standard since they are just another type of feature; but in many cases they will be hosted by third party servers. This API is designed to allow linking to such external entities (although implementations are encouraged to cache some of this data locally to allow for faster join queries).

The sampling feature model used in the CS API also supports “sampling chains”, where several samples are linked via sub-sampling relationships. This is often done when analyzing specimens

(or material samples) in a lab, but can also be applied to other types of sampling features. A few examples are provided below:

### Sampling Chains Examples

- **Geology:** When performing rock analysis, the ultimate feature of interest may be an entire rock formation. A large core sample (13cm in diameter) is collected in the field, then smaller core plug samples (2cm diameter) are extracted from the core. In this case, the sampling chain is specimen (core plug) → specimen (entire core) → rock formation.
- **Video Surveillance:** A video camera has for ultimate feature of interest a road intersection. A viewing sector (portion of a sphere) is used to model the entire area viewable by the camera. The subset of this entire volume being viewed at any given instant is modeled by a frustum. In this case, the sampling chain is frustum → viewing sector → crossroad.
- **Oceanic Observations:** A vessel measures water parameters along its trajectory. The ultimate feature of interest is the Atlantic Ocean. The path corresponding to the trajectory of the vessel is a sampling curve, and each observation point along the way is a sub-sample of the path. In this case, the sampling chain is sampling point → sampling curve (path) → ocean

In this API, observations and commands are always associated to a `Sampling Feature`, never directly to the ultimate `Feature of Interest`. Each `Sampling Feature` resource has an association to the sampled feature (i.e. the feature being sampled) which can be the ultimate `Feature of Interest` or another (larger) `Sampling Feature` in the case of a sampling chain.

## 14.3. Sampling Feature Resource

---

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

This section defines the attributes and associations that are common to all `Sampling Feature` resources, but the exact way attributes and associations are encoded in the payload is defined by each encoding. For encodings defined in this document, please see:

- `SamplingFeature` resource as GeoJSON

Future parts of this Standard will define concrete types of `Sampling Features` that can be used to document different sampling strategies.

### 14.3.1. Common Properties

The following tables describe the attributes and associations that are common to all `Sampling Feature` resource and their mapping to SOSA/SSN.

All Sampling Feature resource representations provided by encoding requirements classes map to these properties.

**Table 17 – Common Sampling Feature Attributes**

NAME	SOSA/SSN PROPERTY	DEFINITION	DATA TYPE	USAGE
uniqueIdentif	RDF concept URI	The unique identifier of the sampling feature in the form of a URI (preferably a URN). This identifier must be persistent and independent of the actual resource URL.	URI	Required
name	rdfs:label	The human readable name of the sampling feature.	String	Required
description	rdfs:comment	A human readable description for the sampling feature.	String	Optional
featureType	rdfs:type	The type of sampling feature.	URI	Required
validTime	-	The validity period of the sampling feature.	Date Time	Optional

NOTE: Valid time is used when the sampling feature is not meaningful or usable outside of a given time period, for example:

- Biological samples are usually analyzed during a short time window after being extracted from the patient.
- Image footprints are synthetic features that are only valid at the time the image was collected.

**Table 18 – Sampling Features Associations**

RELATION NAME	SOSA/SSN PROPERTY	DEFINITION	TARGET CONTENT	USAGE
parentSystem	-	The System that created or uses this Sampling Feature.	A single System resource.	Required
sampledFeatur	sosa:hasSampledFea	The ultimate feature of interest that is being sampled or controlled.	A single Feature resource.	Required
sampleOf	sosa:isSampleOf	Other Sampling Features related to this Sampling Feature via sub-sampling.	A list of Sampling Feature resources (inline or by-reference).	Optional
datastreams	-	The Data Streams that contain observations of this Sampling Feature.	A list of DataStream resources (inline or by-reference).	Optional
controlstream	-	The Control Streams that received commands impacting this Sampling Feature.	A list of ControlStream resources (inline or by-reference).	Optional

## 14.4. Sampling Feature Canonical URL

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

### REQUIREMENT 29

**IDENTIFIER** /req/sf/canonical-url

**INCLUDED IN** Requirements class 7: /req/sf

**A** Every Sampling Feature resource exposed by the server SHALL be accessible through its canonical URL of the form {api\_root}/samplingFeatures/{id} where id is the local identifier of the Sampling Feature resource.

**B** If a Sampling Feature 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.

## 14.5. Sampling Feature Resources Endpoints

### 14.5.1. Definition

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

### REQUIREMENT 30

**IDENTIFIER** /req/sf/resources-endpoint

**INCLUDED IN** Requirements class 7: /req/sf

**A** The server SHALL support the HTTP GET operation at the path associated to the Sampling Feature resources endpoint.

**B** The operation SHALL fulfill all requirements defined in [Clause 7.15.2 to 7.15.8](#) of OGC API – Features – Part 1: Core.

**C** All features in the result set SHALL be Sampling Feature resources.

Clause 16.8 defines additional query parameters applicable to Sampling Feature resources endpoint.

### 14.5.2. Canonical Sampling Feature Resources Endpoint

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

#### REQUIREMENT 31

IDENTIFIER `/req/sf/canonical-endpoint`

INCLUDED IN Requirements class 7: `/req/sf`

A The server SHALL expose a Sampling Feature resources endpoint at the path `{api_root}/samplingFeatures`.

B The endpoint SHALL expose all Sampling Feature resources available on the server.

### 14.5.3. Nested Sampling Feature Resources Endpoint

#### REQUIREMENT 32

IDENTIFIER `/req/sf/ref-from-system`

INCLUDED IN Requirements class 7: `/req/sf`

A For each System resource, the server SHALL expose a Sampling Feature resources endpoint at the path `{api_root}/systems/{sysId}/samplingFeatures`.

B This Sampling Feature resources endpoint SHALL only list Sampling Feature resources that are associated to the parent System resource with local ID `sysId`.

C The `samplingFeatures` association in the parent System resource representation SHALL be implemented as a link to this Sampling Feature resources endpoint.

D The parameter `sysId` SHALL be the local identifier of the parent System resource.

## 14.6. Sampling Feature Collections

Any number of feature collections containing `Sampling Features` can be available at a CS API endpoint, but the server must at least expose one. Members of `Sampling Feature` collections are identified with the feature type `sosa:Sample`.

`Sampling Feature` resources can be grouped into collections according to any arbitrary criteria, as shown in the following examples.

### Examples of Sampling Feature Collections

- All river sampling stations at URL `{api_root}/collections/hydro_sampling_points`
- All satellite image footprints at URL `{api_root}/collections/img_footprints`

*Note that a given sampling feature can be part of multiple collections at the same time.*

### REQUIREMENT 33

**IDENTIFIER** `/req/sf/collections`

**INCLUDED IN** Requirements class 7: `/req/sf`

- A** The server SHALL expose at least one `Feature collection` containing `Sampling Feature` resources.
- B** The server SHALL identify all `Feature collections` containing `Sampling Feature` resources by setting the `itemType` attribute to `feature` and the `featureType` attribute to `sosa:Sample` in the `Collection` metadata.
- C** For any `feature collection` with `featureType` set to `sosa:Sample`, the HTTP GET operation at the path `/collections/{collectionId}/items` SHALL support the query parameters and response of a `Sampling Feature` resources endpoint.

## 14.7. Dynamic properties

When some of the `sampling feature` properties are dynamic, they are also modeled as `observations` (just like any other property observed on the feature of interest). An association with the `datastream` containing these `observations` can optionally be provided along with the `sampling feature` resource. The way this association is provided is encoding specific.

When the `datetime` parameter is included in the request, it is also possible to include a “snapshot” of these dynamic properties (i.e. the value of the property that is valid at the requested time) in the `feature` resource.

**NOTE:** Dynamic properties can also be exposed using the OGC API – Moving Features standard.

15

# REQUIREMENTS CLASS “PROPERTY DEFINITIONS”

---

# REQUIREMENTS CLASS “PROPERTY DEFINITIONS”

## 15.1. Overview

REQUIREMENTS CLASS 8	
IDENTIFIER	/req/property
TARGET TYPE	Web API
CONFORMANCE CLASS	Conformance class A.8: /conf/property
PREREQUISITE	Requirements class 1: /req/api-common
NORMATIVE STATEMENTS	Requirement 34: /req/property/canonical-url Requirement 35: /req/property/resources-endpoint Requirement 36: /req/property/canonical-endpoint Requirement 37: /req/property/collections

The “Property Definitions” requirements class specifies how property definitions are provided using the CS API.

The Property resource implements the *Property* concept defined in the SSN Ontology. Properties in the scope of the CS API are features of interest properties, including:

- Observable properties (i.e. subject of an observation, see *sosa:ObservableProperty*)
- Controllable properties (i.e. subject of an actuation, see *sosa:ActuatableProperty*)
- System properties (i.e. system characteristics and capabilities, which are sometimes asserted, see *ssn-system:SystemCapability*)

NOTE: The same property can be used in all 3 contexts simultaneously (e.g. the same “engine temperature” property can be measured by a sensor, controlled by the cooling system, and asserted in a min/max specification).

The definition is provided by deriving the property from a well known entity referenced in an ontology such as QUDT Quantity Kinds.

### Derived Property Examples

- Combustion Chamber Temperature (derived from qudt:Temperature)

- Hourly Average CPU Temperature (derived from qudt:Temperature)
- Engine Output Power (derived from qudt:Power)
- Received X-Band RF Power (derived from qudt:Power)

## 15.2. Property Resource

### 15.2.1. Introduction

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

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

- *Property* resource as SensorML

### 15.2.2. Properties

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

All *Property* resource representations provided by encoding requirements classes map to these properties.

**Table 19** – Property Definition Attributes

NAME	SOSA/SSN PROPERTY	DEFINITION	DATA TYPE	USAGE
uniqueIdentif	RDF concept URI	The unique identifier of the property in the form of a URI (preferably a URN). This identifier should be persistent and independent of the actual resource URL.	URI	Required
name	rdfs:label	The human readable name of the property.	String	Required
description	rdfs:comment	A human readable description for the property.	String	Optional
baseProperty	-	Reference to the definition of the base property this property is derived from.	URI or CURIE	Required

NAME	SOSA/SSN PROPERTY	DEFINITION	DATA TYPE	USAGE
objectType	-	Reference to the type of entity that the base property applies to.	URI or CURIE	Optional
statistic	-	Reference to the definition of the statistic applied to the base property values.	URI or CURIE	Optional

## 15.3. Property Canonical URL

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

### REQUIREMENT 34

**IDENTIFIER** /req/property/canonical-url

**INCLUDED IN** Requirements class 8: /req/property

**A** Every Property resource exposed by the server SHALL be accessible through its canonical URL of the form {api\_root}/properties/{id} where id is the local identifier of the property.

**B** If a Property 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.

## 15.4. Property Resources Endpoints

### 15.4.1. Definition

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

### REQUIREMENT 35

**IDENTIFIER** /req/property/resources-endpoint

**INCLUDED IN** Requirements class 8: /req/property

## REQUIREMENT 35

A	The server SHALL support the HTTP GET operation at the path associated to the Property resources endpoint.
B	The operation SHALL fulfill all requirements defined in <a href="#">Clause 7.15.2 to 7.15.8</a> 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	All resources in the result set SHALL be Property resources.

Clause 16.9 defines additional query parameters applicable to Property resources endpoint.

### 15.4.2. Canonical Property Resources Endpoint

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

## REQUIREMENT 36

IDENTIFIER /req/property/canonical-endpoint

INCLUDED IN Requirements class 8: /req/property

A	The server SHALL expose a Property resources endpoint at the path {api_root}/properties.
B	The endpoint SHALL expose all Property resources available on the server.

## 15.5. Property Collections

Any number of resource collections containing Property resources can be available at a CS API endpoint, but the server must at least expose one. Members of Property resource collections are identified with the item type `sosa:Property`.

Property resources can be grouped into collections according to any arbitrary criteria, as shown in the following examples.

### Examples of Property Collections

- All standard WMO properties at URL {api\_root}/collections/wmo\_properties
- All chemical properties at URL {api\_root}/collections/chemical\_properties

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

## REQUIREMENT 37

**IDENTIFIER** /req/property/collections

**INCLUDED IN** Requirements class 8: /req/property

**A** The server SHALL expose at least one resource collection containing Property resources.

**B** The server SHALL identify all resource collections containing Property resources by setting the `itemType` attribute to `sosa:Property` in the Collection metadata.

**C** For any resource collection with `itemType` set to `sosa:Property`, the HTTP GET operation at the path `/collections/{collectionId}/items` SHALL support the query parameters and response of a Property resources endpoint.

16

# REQUIREMENTS CLASS “ADVANCED FILTERING”

---

# REQUIREMENTS CLASS “ADVANCED FILTERING”

## 16.1. Overview

### REQUIREMENTS CLASS 9

<b>IDENTIFIER</b>	<code>/req/advanced-filtering</code>
<b>TARGET TYPE</b>	Web API
<b>CONFORMANCE CLASS</b>	Conformance class A.9: <code>/conf/advanced-filtering</code>
<b>PREREQUISITE</b>	Requirements class 1: <code>/req/api-common</code>
<b>INDIRECT PREREQUISITE</b>	OGC 06-103r4 – Clause 7: Well-known Text Representation for Geometry
<b>NORMATIVE STATEMENTS</b>	<p>Requirement 38: <code>/req/advanced-filtering/id-list-schema</code>            Requirement 39: <code>/req/advanced-filtering/resource-by-id</code>            Requirement 40: <code>/req/advanced-filtering/resource-by-keyword</code>            Requirement 41: <code>/req/advanced-filtering/feature-by-geom</code>            Requirement 42: <code>/req/advanced-filtering/system-by-parent</code>            Requirement 43: <code>/req/advanced-filtering/system-by-procedure</code>            Requirement 44: <code>/req/advanced-filtering/system-by-foi</code>            Requirement 45: <code>/req/advanced-filtering/system-by-obsprop</code>            Requirement 46: <code>/req/advanced-filtering/system-by-controlprop</code>            Requirement 52: <code>/req/advanced-filtering/procedure-by-obsprop</code>            Requirement 53: <code>/req/advanced-filtering/procedure-by-controlprop</code>            Requirement 47: <code>/req/advanced-filtering/deployment-by-parent</code>            Requirement 48: <code>/req/advanced-filtering/deployment-by-system</code>            Requirement 49: <code>/req/advanced-filtering/deployment-by-foi</code>            Requirement 50: <code>/req/advanced-filtering/deployment-by-obsprop</code>            Requirement 51: <code>/req/advanced-filtering/deployment-by-controlprop</code>            Requirement 54: <code>/req/advanced-filtering/sf-by-foi</code>            Requirement 55: <code>/req/advanced-filtering/sf-by-obsprop</code>            Requirement 56: <code>/req/advanced-filtering/sf-by-controlprop</code>            Requirement 57: <code>/req/advanced-filtering/prop-by-baseprop</code>            Requirement 58: <code>/req/advanced-filtering/prop-by-object</code>            Requirement 59: <code>/req/advanced-filtering/combined-filters</code></p>

The “Advanced Filtering” 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 can be used in addition to the ones required by Clause 8.

## 16.2. Definitions

### 16.2.1. ID List Schema

The following requirement defines a schema for an “identifier list” that is used by several query parameters. Identifiers in the list can be either local resource IDs or UIDs (URI) but the two types of identifiers cannot be mixed in the same request.

REQUIREMENT 38	
IDENTIFIER	/req/advanced-filtering/id-list-schema
INCLUDED IN	Requirements class 9: /req/advanced-filtering
	A query parameter of type ID_List SHALL conform to the following OpenAPI 3.0 schema: description: <b>List of resource local IDs or unique identifiers (URIs).</b> oneOf: <ul style="list-style-type: none"><li>- type: <b>array</b> title: <b>Local IDs</b> minItems: 1 items:<ul style="list-style-type: none"><li>A type: <b>string</b> minLength: 1</li></ul></li><li>- type: <b>array</b> title: <b>Unique IDs</b> minItems: 1 items:<ul style="list-style-type: none"><li>type: <b>string</b> format: <b>uri</b></li></ul></li></ul>
B	Items in the list SHALL be valid resource IDs or resource UIDs.
C	All items in the list SHALL be of the same ID type (either all resource IDs or all resource UIDs).

## 16.3. Common Resource Query Parameters

### 16.3.1. Overview

The filtering options defined in this section are common to all resource types defined in this Standard, and may also be used with other resource types that are hosted at the same CS API endpoint.

When this requirements class is implemented, the requirements in this class are applicable to all HTTP GET operations used to retrieve items from any collection offered by the server (i.e. it is not allowed to implement this class only for certain collections).

### 16.3.2. ID Filter

The ID filter is used to select resources that match one of the requested identifiers.

#### REQUIREMENT 39

**IDENTIFIER** /req/advanced-filtering/resource-by-id

**INCLUDED IN** Requirements class 9: /req/advanced-filtering

**A** The HTTP GET operation at any resources endpoint defined by this Standard SHALL support a parameter `id` of type `ID_List`.

**B** Only resources that are assigned one of the specified identifiers SHALL be part of the result set.

**C** If a UID specified in the query ends with a trailing `*` character, all resources that have a UID starting with the specified prefix SHALL be included in the result set.

### 16.3.3. Keyword Filter

The keyword filter is used to select resources by doing a full-text search on textual content.

#### REQUIREMENT 40

**IDENTIFIER** /req/advanced-filtering/resource-by-keyword

**INCLUDED IN** Requirements class 9: /req/advanced-filtering

## REQUIREMENT 40

The HTTP GET operation at any resources endpoint defined by this Standard SHALL support a parameter `q` with the following characteristics (using an OpenAPI 3.0 fragment):

```
name: q
in: query
description: |-
  Comma separated list of keywords used for full-text search.
  Only resources that have textual fields that contain one of the specified
  keywords are selected.
  The resource `name` and `description` properties are always searched.
  It is up to the server to decide which other textual fields are searched.
required: false
schema:
  type: array
  items:
    type: string
    minLength: 1
    maxLength: 50
  explode: false

examples:
  case1:
    summary: One keyword
    value: 'temp'
  case2:
    summary: Several keywords
    value: 'gps,imu'
```

A

B Only resources that have human readable content that contains one the specified keywords SHALL be part of the result set.

C At least the name and description attributes of the resource SHALL be included in the full-text search. It is the decision of the server to choose if other resource attributes are also searched.

D The server is allowed to run the search using the canonical form of the provided keywords rather than their exact value (lemmatization).

### 16.3.4. Simple Property Filter

The property filter is used to select resources that have specific property values.

## RECOMMENDATION 3

IDENTIFIER `/rec/advanced-filtering/resource-by-property`

A Filtering on resource properties SHOULD be supported as specified by requirement [/rec/core/fc-filters](#) of OGC API – Features.

### Example queries by property value

Systems by 'name' `{api_root}/systems?name=Weather%20Station`

Sampling features by  
'featureType' {api\_root}/samplingFeatures?featureType=om:Specimen

## 16.4. Common Feature Query Parameters

### 16.4.1. Geometry Filter

The geometry filter is used to select features with a spatial geometry intersecting the query geometry.

#### REQUIREMENT 41

**IDENTIFIER** /req/advanced-filtering/feature-by-geom

**INCLUDED IN** Requirements class 9: /req/advanced-filtering

**PREREQUISITE** OGC 06-103r4 – Clause 7: Well-known Text Representation (WKT) for Geometry

The HTTP GET operation at any resources endpoint defined by this Standard that expose features with geometries SHALL support a parameter geom with the following characteristics (using an OpenAPI 3.0 fragment):

**A**

```
name: geom
in: query
description: |-
  WKT geometry and operator to filter resources on their location or
  geometry.
  Only features that have a geometry that intersects the value of `geom`
  are selected.
required: false
schema:
  type: string
examples:
  point:
    value: 'LINESTRING((-86.53 12.45), (-86.54 12.46), (-86.55 12.47))'
  polygon:
    value: 'POLYGON((0 0,4 0,4 4,0 4,0 0))'
```

**B** The value of the geom parameter SHALL be a valid WKT geometry (as defined by BNF syntax provided in OGC 06-103r4).

**C** Only features that have a spatial geometry that intersects the filter geometry SHALL be part of the result set.

**D** If a feature has multiple spatial geometry properties, it is the decision of the server whether only a single spatial geometry property is used to determine the extent or all relevant geometries.

**E** Features with no spatial geometry SHALL be excluded from the result set.

## REQUIREMENT 41

F	The coordinate values SHALL be within the extent specified for the coordinate reference system (CRS).
---	---

## 16.5. System Resources Endpoint Query Parameters

### 16.5.1. Introduction

This section specifies query parameters that must be supported by the server at all System resources endpoints, including:

- The canonical System resources endpoint at `{api_root}/systems`
- The subsystems resources endpoints at `{api_root}/systems/{id}/subsystems`
- The items resources endpoints of System Feature collections at `{api_root}/collections/{colId}/items`

### 16.5.2. Parent System Filter

This filter is used to select subsystems of one or more parent systems.

## REQUIREMENT 42

**IDENTIFIER** `/req/advanced-filtering/system-by-parent`

**INCLUDED IN** Requirements class 9: `/req/advanced-filtering`

**A** The HTTP GET operation at a System resources endpoint SHALL support a parameter `parent` of type `ID_List`.

**B** Only subsystems of a parent system that has one of the requested identifiers SHALL be part of the result set.

### Example queries to find Systems by parent

Subsystems by parent ID `{api_root}/systems?parent=4g4ds54vv`

Subsystems by parent UID `{api_root}/systems?parent=urn:uuid:31f6865e-f438-430e-9b57-f965a21ee255`

### 16.5.3. Procedure Filter

This filter is used to select systems that implement specific procedures.

#### REQUIREMENT 43

**IDENTIFIER** `/req/advanced-filtering/system-by-procedure`

**INCLUDED IN** Requirements class 9: `/req/advanced-filtering`

**A** The HTTP GET operation at a System resources endpoint SHALL support a parameter procedure of type ID\_List.

**B** Only systems that implement a procedure that has one of the requested identifiers SHALL be part of the result set.

#### Example queries to find systems by procedure

Systems by procedure ID `{api_root}/systems?procedure=11gsd654g`

Systems by procedure ID (multiple) `{api_root}/systems?procedure=11gsd654g, fsv4dg62`

Systems by procedure UID `{api_root}/systems?procedure=urn:example:procedure:451585`

### 16.5.4. Feature of Interest Filter

This filter is used to select systems that observe or control specific features of interest.

#### REQUIREMENT 44

**IDENTIFIER** `/req/advanced-filtering/system-by-foi`

**INCLUDED IN** Requirements class 9: `/req/advanced-filtering`

**A** The HTTP GET operation at a System resources endpoint SHALL support a parameter foi of type ID\_List.

## REQUIREMENT 44

B	Only systems that observe or control 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.
D	If a system has subsystems, it SHALL be included in the result set if any of its subsystems (evaluated recursively) observes or controls one of the listed features of interest.

### Example queries to find Systems by feature of interest

Systems by feature of interest ID	<code>{api_root}/systems?foi=11gsd654g</code>
Systems by feature of interest UID	<code>{api_root}/systems?foi=urn:mrn:itu:mmsi:538070999</code>
Systems by feature of interest UID	<code>{api_root}/systems?foi=http://dbpedia.org/resource/Seawater</code>

In the third example, any system associated to a sampling feature with the `sampledFeature` association referencing <http://dbpedia.org/resource/Seawater> (either directly or transitively via other sampling features) would be included in the result set.

## 16.5.5. Observed Property Filter

This filter is used to select systems that can observe specific properties.

## REQUIREMENT 45

IDENTIFIER	<code>/req/advanced-filtering/system-by-obsprop</code>
INCLUDED IN	Requirements class 9: <code>/req/advanced-filtering</code>
A	The HTTP GET operation at a System resources endpoint SHALL support a parameter <code>observedProperty</code> of type <code>ID_List</code> .
B	Only systems that can observe a property that has one of the requested identifiers SHALL be part of the result set.
C	If a system has subsystems, it SHALL be included in the result set if any of its subsystems (evaluated recursively) observes one of the listed properties.

### Example queries to find Systems by observed property

Systems by observed property ID	{api_root}/systems?observedProperty=4578441
Systems by observed property UID	{api_root}/systems?observedProperty=http://qudt.org/vocab/quantitykind/Temperature

## 16.5.6. Controlled Property Filter

This filter is used to select systems that control specific properties.

REQUIREMENT 46	
<b>IDENTIFIER</b>	/req/advanced-filtering/system-by-controlprop
<b>INCLUDED IN</b>	Requirements class 9: /req/advanced-filtering
<b>A</b>	The HTTP GET operation at a System resources endpoint SHALL support a parameter controlledProperty of type ID_List.
<b>B</b>	Only systems that can control a property that has one of the requested identifiers SHALL be part of the result set.
<b>C</b>	If a system has subsystems, it SHALL be included in the result set if any of its subsystems (evaluated recursively) controls one of the listed properties.

### Example queries to find systems by controlled property

Systems by controlled property ID	{api_root}/systems?controlledProperty=4578441
Systems by controlled property UID	{api_root}/systems?controlledProperty=http://qudt.org/vocab/quantitykind/PH

## 16.6. Deployment Resources Endpoint Query Parameters

### 16.6.1. Introduction

This section specifies query parameters that must be supported by the server at all Deployment resources endpoints, including:

- The canonical Deployment resources endpoint at {api\_root}/deployments

- The subdeployments resources endpoints at {api\_root}/deployments/{id}/subdeployments
- The items resources endpoints of Deployment Feature collections at {api\_root}/collections/{colId}/items

## 16.6.2. Parent Deployment Filter

This filter is used to select subdeployments of a specific parent deployment.

REQUIREMENT 47	
IDENTIFIER	/req/advanced-filtering/deployment-by-parent
INCLUDED IN	Requirements class 9: /req/advanced-filtering
A	The HTTP GET operation at a Deployment resources endpoint SHALL support a parameter parent of type ID_List.
B	Only deployments that are part of a parent deployment that has one of the requested identifiers SHALL be part of the result set.

### Example queries to find Deployments by parent

Subdeployments by parent ID	{api_root}/deployments?parent=4g4ds54vv
Subdeployments by parent UID	{api_root}/deployments?parent=urn:uuid:31f6865e-f438-430e-9b57-f965a21ee255

## 16.6.3. Deployed System Filter

This filter is used to select deployments during which certain systems are deployed.

REQUIREMENT 48	
IDENTIFIER	/req/advanced-filtering/deployment-by-system
INCLUDED IN	Requirements class 9: /req/advanced-filtering
A	The HTTP GET operation at a Deployment resources endpoint SHALL support a parameter system of type ID_List.

## REQUIREMENT 48

- B** Only deployments during which a system that has one of the requested identifiers is deployed SHALL be part of the result set.

### Example queries to find Deployments by deployed systems

Deployments by  
deployed system ID `{api_root}/deployments?system=b5bxc988rf`

Deployments by  
deployed system UID `{api_root}/deployments?system=urn:mrn:itu:mmsi:538070999`

## 16.6.4. Feature of Interest Filter

This filter is used to select deployments during which deployed systems observe or control specific features of interest.

## REQUIREMENT 49

**IDENTIFIER** `/req/advanced-filtering/deployment-by-foi`

**INCLUDED IN** Requirements class 9: `/req/advanced-filtering`

- A** The HTTP GET operation at a Deployment resources endpoint SHALL support a parameter `foi` of type `ID_List`.
- B** Only deployments during which a deployed system observe or control 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.

### Example queries to find Deployments by feature of interest

Deployments by feature  
of interest ID `{api_root}/deployments?foi=g4sd56ht41`

Deployments by feature  
of interest UID `{api_root}/deployments?foi=urn:example:river:41148`

## 16.6.5. Observed Property Filter

This filter is used to select deployments during which certain properties are observed.

## REQUIREMENT 50

**IDENTIFIER** /req/advanced-filtering/deployment-by-obsprop

**INCLUDED IN** Requirements class 9: /req/advanced-filtering

**A** The HTTP GET operation at a Deployment resources endpoint SHALL support a parameter `observedProperty` of type `ID_List`.

**B** Only deployments during which a deployed system observes a property that has one of the requested identifiers SHALL be part of the result set.

### Example queries to find Deployments by observed property

Deployments by observed property ID `{api_root}/deployments?observedProperty=4578441`

Deployments by observed property UID `{api_root}/deployments?observedProperty=http://mmisw.org/ont/cf/parameter/wind_speed`

## 16.6.6. Controlled Property Filter

This filter is used to select deployments during which certain properties are controlled.

## REQUIREMENT 51

**IDENTIFIER** /req/advanced-filtering/deployment-by-controlprop

**INCLUDED IN** Requirements class 9: /req/advanced-filtering

**A** The HTTP GET operation at a Deployment resources endpoint SHALL support a parameter `controlledProperty` of type `ID_List`.

**B** Only deployments during which a deployed system controls a property that has one of the requested identifiers SHALL be part of the result set.

### Example queries to find Deployments by controlled property

Deployments by controlled property ID `{api_root}/deployments?controlledProperty=146687`

Deployments by controlled property UID `{api_root}/deployments?controlledProperty=http://qudt.org/vocab/quantitykind/Velocity`

## 16.7. Procedure Resources Endpoint Query Parameters

### 16.7.1. Introduction

This section specifies query parameters that must be supported by the server at all Procedure resources endpoints, including:

- The canonical Procedure resources endpoint at `{api_root}/procedures`
- The items resources endpoints of Procedure Feature collections at `{api_root}/collections/{colId}/items`

### 16.7.2. Observed Property Filter

This filter is used to select procedures that are designed to observe certain properties.

#### REQUIREMENT 52

**IDENTIFIER** `/req/advanced-filtering/procedure-by-obsprop`

**INCLUDED IN** Requirements class 9: `/req/advanced-filtering`

**A** The HTTP GET operation at a Procedure resources endpoint SHALL support a parameter `observedProperty` of type `ID_List`.

**B** Only procedures that can be used to observe a property that has one of the requested identifiers SHALL be part of the result set.

#### Example queries to find Procedures by observed property

Procedures by observed property ID `{api_root}/procedures?observedProperty=4578441`

Procedures by observed property UID `{api_root}/procedures?observedProperty=http://mmisw.org/ont/cf/parameter/air_pressure`

### 16.7.3. Controlled Property Filter

This filter is used to select procedures that are designed to control certain properties.

## REQUIREMENT 53

**IDENTIFIER** /req/advanced-filtering/procedure-by-controlprop

**INCLUDED IN** Requirements class 9: /req/advanced-filtering

**A** The HTTP GET operation at a Procedure resources endpoint SHALL support a parameter controlledProperty of type ID\_List.

**B** Only procedures that can be used to control a property that has one of the requested identifiers SHALL be part of the result set.

### Example queries to find Procedures by controlled property

Procedures by controlled property ID {api\_root}/procedures?controlledProperty=146687

Procedures by controlled property UID {api\_root}/procedures?controlledProperty=urn:example:prop:fuelmixratio

## 16.8. Sampling Feature Resources Endpoint Query Parameters

### 16.8.1. Introduction

This section specifies query parameters that must be supported by the server at all Sampling Feature resources endpoints, including:

- The canonical Sampling Feature resources endpoint at {api\_root}/samplingFeatures
- The items resources endpoints of Sampling Feature collections at {api\_root}/collections/{colId}/items

### 16.8.2. Feature of Interest Filter

This filter is used to select sampling features that sample specific features of interest.

## REQUIREMENT 54

**IDENTIFIER** /req/advanced-filtering/sf-by-foi

**INCLUDED IN** Requirements class 9: /req/advanced-filtering

**A** The HTTP GET operation at a Sampling Feature resources endpoint SHALL support a parameter foi of type ID\_List.

**B** Only sampling features that are associated to a feature of interest that has one of the requested identifiers SHALL be part of the result set.

### Example queries to find Sampling Features by feature of interest

Sampling Features by feature of interest ID {api\_root}/samplingFeatures?foi=g4sd56ht41

Sampling Features by feature of interest UID {api\_root}/samplingFeatures?foi=urn:example:river:41148

## 16.8.3. Observed Property Filter

This filter is used to select sampling features with certain observed properties.

## REQUIREMENT 55

**IDENTIFIER** /req/advanced-filtering/sf-by-obsprop

**INCLUDED IN** Requirements class 9: /req/advanced-filtering

**A** The HTTP GET operation at a Sampling Feature resources endpoint SHALL support a parameter observedProperty of type ID\_List.

**B** Only sampling features with an observed property that has one of the requested identifiers SHALL be part of the result set.

### Example queries to find Sampling Features by observed property

Sampling Features by observed property ID {api\_root}/samplingFeatures?observedProperty=221785

Sampling Features by observed property UID {api\_root}/samplingFeatures?observedProperty=http://qudt.org/vocab/quantitykind/Voltage

## 16.8.4. Controlled Property Filter

This filter is used to select sampling features with certain controlled properties.

### REQUIREMENT 56

**IDENTIFIER** /req/advanced-filtering/sf-by-controlprop

**INCLUDED IN** Requirements class 9: /req/advanced-filtering

**A** The HTTP GET operation at a Sampling Feature resources endpoint SHALL support a parameter controlledProperty of type ID\_List.

**B** Only sampling features with a controlled property that has one of the requested identifiers SHALL be part of the result set.

### Example queries to find Sampling Features by controlled property

Sampling Features by controlled property ID {api\_root}/samplingFeatures?controlledProperty=146687

Sampling Features by controlled property UID {api\_root}/samplingFeatures?controlledProperty=http://qudt.org/vocab/quantitykind/Velocity

## 16.9. Property Resources Endpoint Query Parameters

### 16.9.1. Introduction

This section specifies query parameters that must be supported by the server at all Property resources endpoints, including:

- The canonical Property resources endpoint at {api\_root}/samplingFeatures
- The items resources endpoints of Property resource collections at {api\_root}/collections/{colId}/items

### 16.9.2. Base Property Filter

This filter is used to select properties that are derived from a base property.

## REQUIREMENT 57

**IDENTIFIER** /req/advanced-filtering/prop-by-baseprop

**INCLUDED IN** Requirements class 9: /req/advanced-filtering

**A** The HTTP GET operation at a Property resources endpoint SHALL support a parameter baseProperty of type ID\_List.

**B** Only properties that are derived, directly or indirectly, from a base property that has one of the requested identifiers SHALL be part of the result set.

### Example queries to find Property Definitions by base property

Property by base property ID {api\_root}/properties?baseProperty=g4sd56ht41

Property by base property UID {api\_root}/properties?baseProperty=http://qudt.org/vocab/quantitykind/AmbientPressure

## 16.9.3. Object Type Filter

This filter is used to select properties that are associated to a specific kind of object.

## REQUIREMENT 58

**IDENTIFIER** /req/advanced-filtering/prop-by-object

**INCLUDED IN** Requirements class 9: /req/advanced-filtering

**A** The HTTP GET operation at a Property resources endpoint SHALL support a parameter objectType of type ID\_List.

**B** Only properties with the objectType attribute set to one of the requested identifiers SHALL be part of the result set.

### Example queries to find Property Definitions by object

Property by object kind UID {api\_root}/properties?object=https://dbpedia.org/page/Watercraft

Property by object kind UID {api\_root}/properties?object=https://dbpedia.org/page/Engine

## 16.10. Combination of Filter Parameters

### REQUIREMENT 59

**IDENTIFIER** /req/advanced-filtering/combined-filters

**INCLUDED IN** Requirements class 9: /req/advanced-filtering

**STATEMENT** When several filters are used in the same request, they SHALL be combined with the AND operator.

#### Example queries combining multiple filters

Systems by type and keyword `{api_root}/systems?featureType=sosa:Sensor&q=weather`

Deployments by deployed system ID and time `{api_root}/deployments?datetime=2021-04-01T00:00:00Z/2021-07-01T00:00:00Z&system=gs54v1fds6`

## 16.11. Indirect Associations

It is recommended that servers implement certain filtering capabilities in a way that does not require a “direct association” to exist between resources.

### RECOMMENDATION 4

**IDENTIFIER** /rec/advanced-filtering/indirect-prop

**STATEMENT** When evaluating `observedProperty` or `controlledProperty` filters, the server SHOULD also evaluate all properties that derive from the specified property, that is properties whose `baseProperty` attribute point to the specified property, either directly or transitively.

### RECOMMENDATION 5

**IDENTIFIER** /rec/advanced-filtering/indirect-foi

## RECOMMENDATION 5

**STATEMENT** When evaluating `foi` filters, the server **SHOULD** also evaluate all sampling features that are related to the specified feature of interest through a `samplingFeature` association. This allows a client to request resources by ultimate feature of interest even if it is not directly associated to the resource.

17

# REQUIREMENTS CLASS “CREATE/REPLACE/DELETE”

---

# REQUIREMENTS CLASS

## “CREATE/REPLACE/DELETE”

### 17.1. Overview

REQUIREMENTS CLASS 10	
IDENTIFIER	/req/create-replace-delete
TARGET TYPE	Web API
CONFORMANCE CLASS	Conformance class A.10: /conf/create-replace-delete
PREREQUISITES	Requirements class 1: /req/api-common <a href="http://www.opengis.net/spec/ogcapi-features-4/1.0/req/create-replace-delete">http://www.opengis.net/spec/ogcapi-features-4/1.0/req/create-replace-delete</a>
NORMATIVE STATEMENTS	Requirement 60: /req/create-replace-delete/system Requirement 61: /req/create-replace-delete/system-delete-cascade Requirement 62: /req/create-replace-delete/subsystem Requirement 63: /req/create-replace-delete/deployment Requirement 64: /req/create-replace-delete/subdeployment Requirement 65: /req/create-replace-delete/procedure Requirement 66: /req/create-replace-delete/sampling-feature Requirement 67: /req/create-replace-delete/property Requirement 68: /req/create-replace-delete/create-in-collection Requirement 69: /req/create-replace-delete/replace-in-collection Requirement 70: /req/create-replace-delete/delete-in-collection Requirement 71: /req/create-replace-delete/add-to-collection

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.

## 17.2. Systems

### REQUIREMENT 60

**IDENTIFIER** /req/create-replace-delete/system

**INCLUDED IN** Requirements class 10: /req/create-replace-delete

**CONDITIONS** The server implements Requirements Class “System Features”

**A** The server SHALL support the CREATE operation at the System resources endpoints defined by the following URI template:

- {api\_root}/systems

**B** The server SHALL support the REPLACE and DELETE operations at the System resources endpoints defined by the following URI template:

- {api\_root}/systems/{id}

**C** The id parameter SHALL be the local identifier of the System resource to replace or delete.

The following constraints must be implemented by the server.

### REQUIREMENT 61

**IDENTIFIER** /req/create-replace-delete/system-delete-cascade

**INCLUDED IN** Requirements class 10: /req/create-replace-delete

**CONDITIONS** The server implements Requirements Class “System Features”

**A** By default, the server SHALL reject a DELETE request on a System resource that has nested resources (i.e. subsystems, sampling features, datastreams, control streams) or that is associated with a deployment.

**B** If the System resource is not associated with any Deployment resource, and the request contains the cascade parameter, the server SHALL accept the DELETE request and delete the System resource as well as all its nested resources.

**C** If the System resource is associated with a Deployment resource, the Deployment resource SHALL be deleted first.

## 17.3. Subsystems

---

Subsystems (i.e. system components) can only be created as sub-resources of a parent system, but are updated/deleted at their canonical URL just like any other System resource.

### REQUIREMENT 62

**IDENTIFIER** /req/create-replace-delete/subsystem

**INCLUDED IN** Requirements class 10: /req/create-replace-delete

**CONDITIONS** The server implements Requirements Class “Subsystems”

**A** The server SHALL support the CREATE operation at the System resources endpoints defined by the following URI template:

- {api\_root}/systems/{parentId}/subsystems

**B** The operation SHALL result in the creation of a new System and its association with the System system with id parentId.

**NOTE:** There is no operation to “move” a subsystem from one parent to another. To achieve this, the client must first delete the subsystem at its canonical URI and recreate it under another parent system.

## 17.4. Deployments

---

### REQUIREMENT 63

**IDENTIFIER** /req/create-replace-delete/deployment

**INCLUDED IN** Requirements class 10: /req/create-replace-delete

**CONDITIONS** The server implements Requirements Class “Deployment Features”

**A** The server SHALL support the CREATE operation at the Deployment resources endpoints defined by the following URI template:

- {api\_root}/deployments

**B** The server SHALL support the REPLACE and DELETE operations at the Deployment resources endpoints defined by the following URI template:

- {api\_root}/deployments/{id}

## REQUIREMENT 63

**C** The id parameter SHALL be the local identifier of the Deployment resource to replace or delete.

## 17.5. Subdeployments

---

Subdeployments can only be created as sub-resources of a parent deployment, but are updated/deleted at their canonical URL just like any other Deployment resource.

## REQUIREMENT 64

**IDENTIFIER** /req/create-replace-delete/subdeployment

**INCLUDED IN** Requirements class 10: /req/create-replace-delete

**CONDITIONS** The server implements Requirements Class “Subdeployments”

**A** The server SHALL support the CREATE operation at the Deployment resources endpoints defined by the following URI template:

- {api\_root}/deployments/{parentId}/subdeployments

**B** The operation SHALL result in the creation of a new Deployment and its association with the parent Deployment with id parentId.

## 17.6. Procedures

---

## REQUIREMENT 65

**IDENTIFIER** /req/create-replace-delete/procedure

**INCLUDED IN** Requirements class 10: /req/create-replace-delete

**CONDITIONS** The server implements Requirements Class “Procedure Features”

**A** The server SHALL support the CREATE operation at the Procedure resources endpoints defined by the following URI template:

- {api\_root}/procedures

## REQUIREMENT 65

- B** The server SHALL support the REPLACE and DELETE operations at the Procedure resources endpoints defined by the following URI template:
- {api\_root}/procedures/{id}
- C** The id parameter SHALL be the local identifier of the Procedure resource to replace or delete.

## 17.7. Sampling Features

Sampling Features are created as sub-resources of a parent system.

## REQUIREMENT 66

**IDENTIFIER** /req/create-replace-delete/sampling-feature

**INCLUDED IN** Requirements class 10: /req/create-replace-delete

**CONDITIONS** The server implements Requirements Class “Sampling Features”

**A** The server SHALL support the CREATE operation at the Sampling Feature resources endpoints defined by the following URI template:

- {api\_root}/systems/{sysId}/samplingFeatures

**B** The server SHALL support the REPLACE and DELETE operations at the Sampling Feature resources endpoints defined by the following URI templates:

- {api\_root}/samplingFeatures/{id}
- {api\_root}/systems/{sysId}/samplingFeatures/{id}

**C** The sysId parameter SHALL be the local identifier of the parent System resource that the new sampling feature is or will be attached to.  
The id parameter SHALL be the local identifier of the Sampling Feature resource to replace or delete.

## 17.8. Property Definitions

Property resources are created, replaced and deleted using HTTP POST, PUT and DELETE operations, respectively.

## REQUIREMENT 67

**IDENTIFIER** /req/create-replace-delete/property

**INCLUDED IN** Requirements class 10: /req/create-replace-delete

**CONDITIONS** The server implements [clause-derived-properties]

**A** The server SHALL support the CREATE operation at the Property resources endpoints defined by the following URI template:

- {api\_root}/properties

**B** The server SHALL support the REPLACE and DELETE operations at the Property resources endpoints defined by the following URI template:

- {api\_root}/properties/{id}

**C** The id parameter SHALL be the local identifier of the Property resource to replace or delete.

## 17.9. Custom Collections

This clause defines the expected behavior of the server when Resource Collections other than the root collections are exposed by the server (this causes the same resource to be accessible via multiple collections simultaneously).

## REQUIREMENT 68

**IDENTIFIER** /req/create-replace-delete/create-in-collection

**INCLUDED IN** Requirements class 10: /req/create-replace-delete

**A** When a resource defined in the CS API Standard is successfully created by the server at any endpoint, the server SHALL always make it available in the root collection corresponding to the resource type.

## REQUIREMENT 69

**IDENTIFIER** /req/create-replace-delete/replace-in-collection

**INCLUDED IN** Requirements class 10: /req/create-replace-delete

**A** If a resource is successfully replaced by the server, the server SHALL reflect this change in all other collections that the resource is part of.

## REQUIREMENT 70

**IDENTIFIER** /req/create-replace-delete/delete-in-collection

**INCLUDED IN** Requirements class 10: /req/create-replace-delete

**A** If a resource is deleted from a root collection, the server SHALL also delete it from all other collections that the resource is part of.

**B** If a resource is deleted from a collection other than the root collection, the server SHALL only delete it from this collection.

Adding existing resources to one or more custom collections is done by posting a list of resource URIs to the collection endpoint.

## REQUIREMENT 71

**IDENTIFIER** /req/create-replace-delete/add-to-collection

**INCLUDED IN** Requirements class 10: /req/create-replace-delete

**A** The server SHALL support adding existing resources to a collection by reference.

**B** The Content-Type header of the request SHALL be set to text/uri-list.

**C** The body of the POST request SHALL contain the list of URIs of resources that need to be added to the collection, formatted with one URI per line (see <https://www.iana.org/assignments/media-types/text/uri-list>).

**D** All URIs included in the content body SHALL be canonical URLs or unique identifiers (UID) of resources available at the same API endpoint.

18

# REQUIREMENTS CLASS “UPDATE”

---

## 18.1. Overview

### REQUIREMENTS CLASS 11

IDENTIFIER	/req/update
TARGET TYPE	Web API
CONFORMANCE CLASS	Conformance class A.11: /conf/update
PREREQUISITES	Requirements class 10: /req/create-replace-delete <a href="http://www.opengis.net/spec/ogcapi-features-4/1.0/req/update">http://www.opengis.net/spec/ogcapi-features-4/1.0/req/update</a>
NORMATIVE STATEMENTS	Requirement 72: /req/update/system Requirement 73: /req/update/deployment Requirement 74: /req/update/procedure Requirement 75: /req/update/sampling-feature Requirement 76: /req/update/property

The “Update” requirements class specifies how instances of the feature types defined previously are updated 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 “resource endpoint” to identify the paths where the UPDATE operation is supported by the server. The following sections provide these endpoints for each resource type defined by the CS API Standard.

## 18.2. Systems

### REQUIREMENT 72

IDENTIFIER	/req/update/system
------------	--------------------

## REQUIREMENT 72

**INCLUDED IN** Requirements class 11: /req/update

**CONDITIONS** The server implements Requirements Class “System Features”

**A** The server SHALL support the UPDATE operation at the System resources endpoints defined by the following URI template:

- {api\_root}/systems/{id}

**B** The id parameter SHALL be the local identifier of the System resource to update.

## 18.3. Deployments

---

## REQUIREMENT 73

**IDENTIFIER** /req/update/deployment

**INCLUDED IN** Requirements class 11: /req/update

**CONDITIONS** The server implements Requirements Class “Deployment Features”

**A** The server SHALL support the UPDATE operation at the Deployment resources endpoints defined by the following URI template:

- {api\_root}/deployments/{id}

**B** The id parameter SHALL be the local identifier of the Deployment resource to update.

## 18.4. Procedures

---

## REQUIREMENT 74

**IDENTIFIER** /req/update/procedure

**INCLUDED IN** Requirements class 11: /req/update

**CONDITIONS** The server implements Requirements Class “Procedure Features”

## REQUIREMENT 74

- A The server SHALL support the UPDATE operation at the Procedure resources endpoints defined by the following URI template:
- {api\_root}/procedures/{id}
- B The id parameter SHALL be the local identifier of the Procedure resource to update.

## 18.5. Sampling Features

### REQUIREMENT 75

**IDENTIFIER** /req/update/sampling-feature

**INCLUDED IN** Requirements class 11: /req/update

**CONDITIONS** The server implements Requirements Class “Sampling Features”

- A The server SHALL support the UPDATE operation at the Sampling Feature resources endpoints defined by the following URI template:
- {api\_root}/samplingFeatures/{id}
- B The id parameter SHALL be the local identifier of the Sampling Feature resource to update.

## 18.6. Derived Properties

### REQUIREMENT 76

**IDENTIFIER** /req/update/property

**INCLUDED IN** Requirements class 11: /req/update

**CONDITIONS** The server implements [clause-derived-properties]

- A The server SHALL support the UPDATE operation at the Property resources endpoints defined by the following URI template:
- {api\_root}/properties/{id}

19

# REQUIREMENTS CLASSES FOR ENCODINGS

---

## 19.1. Requirements Class “GeoJSON Format”

### 19.1.1. Overview

REQUIREMENTS CLASS 12	
IDENTIFIER	/req/geojson
TARGET TYPE	Web API
CONFORMANCE CLASS	Conformance class A.12: /conf/geojson
PREREQUISITES	Requirements class 1: /req/api-common <a href="http://www.opengis.net/spec/ogcapi-features-1/1.0/req/geojson">http://www.opengis.net/spec/ogcapi-features-1/1.0/req/geojson</a>
NORMATIVE STATEMENTS	Requirement 77: /req/geojson/mediatype-read Requirement 78: /req/geojson/mediatype-write Requirement 79: /req/geojson/relation-types Requirement 80: /req/geojson/feature-attribute-mapping Requirement 81: /req/geojson/system-schema Requirement 82: /req/geojson/system-mappings Requirement 83: /req/geojson/deployment-schema Requirement 84: /req/geojson/deployment-mappings Requirement 85: /req/geojson/procedure-schema Requirement 86: /req/geojson/procedure-mappings Requirement 87: /req/geojson/sf-schema Requirement 88: /req/geojson/sf-mappings

The “GeoJSON Format” requirements class specifies how resources defined by the CS API Standard are encoded using the GeoJSON format. All feature types defined by this Standard can be encoded as GeoJSON.

The server must also implement Requirements Class “GeoJSON” as specified in the OGC API – Features – Part 1: Core Standard.

## 19.1.2. Media Type

The media type used for GeoJSON encoded resources is `application/geo+json`.

### REQUIREMENT 77

**IDENTIFIER** `/req/geojson/mediatype-read`

**INCLUDED IN** Requirements class 12: `/req/geojson`

**STATEMENT** The server SHALL support the media type `application/geo+json` in the HTTP Accept header and respond with a JSON document corresponding to the requested resource type.

### REQUIREMENT 78

**IDENTIFIER** `/req/geojson/mediatype-write`

**INCLUDED IN** Requirements class 12: `/req/geojson`

**CONDITIONS** The server implements Requirements Class "Create/Replace/Delete".

**STATEMENT** The server SHALL support the media type `application/geo+json` in the HTTP Content-Type header and parse the JSON body according to the resource type.

## 19.1.3. Link Relation Types

### REQUIREMENT 79

**IDENTIFIER** `/req/geojson/relation-types`

**INCLUDED IN** Requirements class 12: `/req/geojson`

**STATEMENT** For all associations encoded in the `links` member of the JSON response, the link relation type must be set to the association name.

## 19.1.4. Common Encoding Rules

## REQUIREMENT 80

**IDENTIFIER** /req/geojson/feature-attribute-mapping

**INCLUDED IN** Requirements class 12: /req/geojson

**STATEMENT** A GeoJSON document representing a Feature resource SHALL implement the mappings specified in Table 20.

**Table 20 – GeoJSON Mappings of Common Attributes**

ATTRIBUTE NAME	JSON MEMBER	USAGE
uniqueIdentifier	properties/uid	Value is a JSON string that is a valid URI.
name	properties/name	Value is a JSON string.
description	properties/description	Value is a JSON string.

## 19.1.5. System Representation

### REQUIREMENT 81

**IDENTIFIER** /req/geojson/system-schema

**INCLUDED IN** Requirements class 12: /req/geojson

**A** A GeoJSON document containing a single System feature SHALL be valid against the JSON schema system.json.

**B** A GeoJSON document containing a collection of System features SHALL be valid against the JSON schema systemCollection.json.

### REQUIREMENT 82

**IDENTIFIER** /req/geojson/system-mappings

**INCLUDED IN** Requirements class 12: /req/geojson

**STATEMENT** A GeoJSON document representing a System resource SHALL implement the mappings specified in Table 21 and Table 22.

**Table 21 – GeoJSON Encoding of System Attributes**

ATTRIBUTE NAME (SEE TABLE 4)	JSON MEMBER	USAGE
systemType	properties/ featureType	Use the URI or CURIE from Table 6 as the value.
assetType	properties/ assetType	Use the URI or CURIE from Table 7 as the value.
validTime	properties/ validTime	Value SHALL be a JSON array with min/max bounds encoded as a ISO8601 date/time string.
location	geometry	Value SHALL be a GeoJSON Point geometry.

**Table 22 – GeoJSON Encoding of System Associations**

ASSOCIATION NAME (SEE TABLE 5)	JSON MEMBER	USAGE
systemKind	properties/ systemKind@link	Value is a weblink resolving to a Procedure resource.
parentSystem	links	Value is a weblink* resolving to a System resource.
subsystems	links	Value is a weblink* resolving to a System resources endpoint.
samplingFeatures	links	Value is a weblink* resolving to a Sampling Feature resources endpoint.
deployments	links	Value is a weblink* resolving to a Deployment resources endpoint.
procedures	links	Value is a weblink* resolving to a Procedure resources endpoint.
datastreams	links	Value is a weblink* resolving to a DataStream resources endpoint**.
controlstreams	links	Value is a weblink* resolving to a ControlStream resources endpoint**.

\* The link relation type (rel property) must be set to the association name prefixed by ogc-rel:.

\*\* DataStream and ControlStream resources are defined in OGC API – Connected Systems – Part 2.

**Example – A System Feature in GeoJSON format:** This is a simple description of a fixed in-situ sensor in GeoJSON, with a link to its alternate SensorML representation.

```
{
  "type": "Feature",
  "id": "123",
  "geometry": {
    "type": "Point",
    "coordinates": [41.8781, -87.6298]
  }
}
```

```

},
"properties": {
  "uid": "urn:x-ogc:systems:001",
  "name": "Outdoor Thermometer 001",
  "description": "Digital thermometer located on first floor window 1",
  "featureType": "http://www.w3.org/ns/sosa/Sensor",
  "assetType": "http://www.opengis.net/def/x-OGC/TBD/Equipment",
  "systemKind@link": {
    "href": "https://data.example.org/api/procedures/TP60S?f=json",
    "uid": "urn:x-myorg:datasheets:ThermoPro:TP60S:v001",
    "title": "Thermo Pro TP60S",
    "type": "application/geo+json"
  }
},
"links": [
  {
    "rel": "self",
    "title": "this document",
    "href": "https://data.example.org/api/systems/123?f=json",
    "type": "application/geo+json"
  },
  {
    "rel": "alternate",
    "title": "this resource as SensorML",
    "href": "https://data.example.org/api/systems/123?f=sml",
    "type": "application/sml+json"
  }
]
}

```

## 19.1.6. Deployment Representation

### REQUIREMENT 83

**IDENTIFIER** /req/geojson/deployment-schema

**INCLUDED IN** Requirements class 12: /req/geojson

**A** A GeoJSON document containing a single Deployment feature SHALL be valid against the JSON schema [deployment.json](#).

**B** A GeoJSON document containing a collection of Deployment features SHALL be valid against the JSON schema [deploymentCollection.json](#).

### REQUIREMENT 84

**IDENTIFIER** /req/geojson/deployment-mappings

**INCLUDED IN** Requirements class 12: /req/geojson

## REQUIREMENT 84

**STATEMENT** A GeoJSON document representing a Deployment resource SHALL implement the mappings specified in Table 23 and Table 24.

**Table 23** – GeoJSON Encoding of Deployment Attributes

ATTRIBUTE NAME (SEE TABLE 10)	JSON MEMBER	USAGE
deploymentType	properties/ featureType	Use the URI or CURIE identifying the type of deployment as the value.
validTime	properties/ validTime	-
location	geometry	Value is a GeoJSON geometry.

**Table 24** – GeoJSON Encoding of Deployment Associations

ASSOCIATION NAME (SEE TABLE 11)	JSON MEMBER	USAGE
platform	properties/ platform@link	Value is a weblink resolving to a System resource.
deployedSystems	properties/ deployedSystems@ link	Value is a JSON Array of links to System resources.
parentDeployment	links	Value is a weblink* resolving to a Deployment resource.
subdeployments	links	Value is a weblink* resolving to a Deployment resources endpoint.
featuresOfInterest	links	Value is a weblink* resolving to a Feature resources endpoint.
samplingFeatures	links	Value is a weblink* resolving to a Sampling Feature resources endpoint.
datastreams	links	Value is a weblink* resolving to a DataStream resources endpoint**.
controlstreams	links	Value is a weblink* resolving to a ControlStream resources endpoint**.

\* The link relation type (rel property) must be set to the association name prefixed by ogc-rel:.

\*\* DataStream and ControlStream resources are defined in OGC API – Connected Systems – Part 2.

### Example – A Deployment Feature in GeoJSON format

```
{  
  "type": "Feature",
```

```

    "id": "iv3f2kcq27gfi",
    "geometry": {
      "type": "Polygon",
      "coordinates": [[
        [53.76,-173.7],
        [53.76,-155.07],
        [75.03,-155.07],
        [75.03,-173.7],
        [53.76,-173.7]
      ]]
    },
    "properties": {
      "uid": "urn:x-ogc:deployments:D001",
      "name": "Saildrone - 2017 Arctic Mission",
      "featureType": "http://www.w3.org/ns/sosa/Deployment",
      "description": "In July 2017, three saildrones were launched from Dutch Harbor, Alaska, in partnership with NOAA Research...",
      "validTime": ["2017-07-17T00:00:00Z", "2017-09-29T00:00:00Z"],
      "platform@link": {
        "href": "https://data.example.org/api/systems/27559?f=sml",
        "uid": "urn:x-saildrone:platforms:SD-1003",
        "title": "Saildrone SD-1003"
      },
      "deployedSystems@link": [
        {
          "href": "https://data.example.org/api/systems/41548?f=sml",
          "uid": "urn:x-saildrone:sensors:temp01",
          "title": "Air Temperature Sensor"
        },
        {
          "href": "https://data.example.org/api/systems/36584?f=sml",
          "uid": "urn:x-saildrone:sensors:temp02",
          "title": "Water Temperature Sensor"
        },
        {
          "href": "https://data.example.org/api/systems/47752?f=sml",
          "uid": "urn:x-saildrone:sensors:wind01",
          "title": "Wind Speed and Direction Sensor"
        }
      ]
    },
    "links": [
      {
        "rel" : "self",
        "href" : "https://data.example.org/api/deployments/iv3f2kcq27gfi?f=json",
        "type" : "application/geo+json",
        "title" : "this document"
      },
      {
        "rel" : "alternate",
        "href" : "https://data.example.org/api/deployments/iv3f2kcq27gfi?f=sml",
        "type" : "application/sml+json",
        "title" : "this resource as SensorML"
      }
    ]
  }
}

```

### 19.1.7. Procedure Representation

## REQUIREMENT 85

**IDENTIFIER** /req/geojson/procedure-schema

**INCLUDED IN** Requirements class 12: /req/geojson

**A** A GeoJSON document containing a single Procedure feature SHALL be valid against the JSON schema [procedure.json](#).

**B** A GeoJSON document containing a collection of Procedure features SHALL be valid against the JSON schema [procedureCollection.json](#).

## REQUIREMENT 86

**IDENTIFIER** /req/geojson/procedure-mappings

**INCLUDED IN** Requirements class 12: /req/geojson

**STATEMENT** A GeoJSON document representing a Procedure resource SHALL implement the mappings specified in Table 25 and Table 26.

**Table 25 – GeoJSON Encoding of Procedure Attributes**

ATTRIBUTE NAME (SEE TABLE 14)	JSON MEMBER	USAGE
procedureType	properties/ featureType	Use the URI or CURIE from Table 16 as the value.
validTime	properties/ validTime	Value SHALL be a JSON array with min/max bounds encoded as a ISO8601 date/time string.

**Table 26 – GeoJSON Encoding of Procedure Associations**

ASSOCIATION NAME (SEE TABLE 15)	JSON MEMBER	USAGE
implementingSyst	links	Value is a weblink* resolving to a System resources endpoint.

\* The link relation type (rel property) must be set to the association name prefixed by ogc-rel:.

### Example – A Procedure Feature in GeoJSON format

```
{  
  "type": "Feature",  
  "id": "iv3f2kcq27gfi",  
  "geometry": null,  
}
```

```

"properties": {
  "uid": "urn:x-gill:datasheets:windmaster:v1",
  "name": "Gill WindMaster",
  "description": "Precision 3-axis ultrasonic anemometer",
  "featureType": "http://www.w3.org/ns/ssn-system/SensorKind"
},
"links": [
  {
    "href" : "https://data.example.org/api/procedures/iv3f2kcq27gfi?f=json",
    "rel" : "self",
    "type" : "application/geo+json",
    "title" : "this document"
  }, {
    "href" : "https://data.example.org/api/procedures/iv3f2kcq27gfi?f=sml",
    "rel" : "alternate",
    "type" : "application/sml+json",
    "title" : "this resource as SensorML"
  }
]
}

```

### 19.1.8. Sampling Feature Representation

#### REQUIREMENT 87

**IDENTIFIER** /req/geojson/sf-schema

**INCLUDED IN** Requirements class 12: /req/geojson

**A** A GeoJSON document containing a single Sampling Feature SHALL be valid against the JSON schema [anySamplingFeature.json](#).

**B** A GeoJSON document containing a collection of Sampling Features SHALL be valid against the JSON schema [samplingFeatureCollection.json](#).

#### REQUIREMENT 88

**IDENTIFIER** /req/geojson/sf-mappings

**INCLUDED IN** Requirements class 12: /req/geojson

**STATEMENT** A GeoJSON document representing a Sampling Feature resource SHALL implement the mappings specified in Table 27 and Table 28.

**Table 27 – GeoJSON Encoding of Sampling Feature Attributes**

ATTRIBUTE NAME (SEE TABLE 17)	JSON MEMBER	USAGE
featureType	properties/ featureType	Use the URI or CURIE specific to a sampling feature type.
validTime	properties/ validTime	Value SHALL be a JSON array with min/max bounds encoded as a ISO8601 date/time string.

**Table 28 – GeoJSON Encoding of Sampling Feature Associations**

ASSOCIATION NAME (SEE TABLE 18)	JSON MEMBER	USAGE
sampledFeature	properties/ sampledFeature@ link	Value is a weblink resolving to a Feature resource.
parentSystem	links	Value is a weblink* resolving to a System resource.
sampleOf	links	Value is a weblink* resolving to a Sampling Feature resources endpoint.
datastreams	links	Value is a weblink* resolving to a DataStream resources endpoint**.
controlstreams	links	Value is a weblink* resolving to a ControlStream resources endpoint**.

\* The link relation type (rel property) must be set to the association name prefixed by ogc-rel:.

\*\* DataStream and ControlStream resources are defined in OGC API – Connected Systems – Part 2.

**Example 1 – A Sampling Point Feature in GeoJSON format**

```
{
  "type": "Feature",
  "id": "SP001",
  "geometry": {
    "type": "Point",
    "coordinates": [ 12.31, -86.98, -21]
  },
  "properties": {
    "uid": "urn:x-usgs:sites:301244087575701:sf:bottom",
    "name": "Bottom of Well - USGS Site #301244087575701",
    "description": "Sampling point is 2-4 inches above the bottom of the well",
    "featureType": "http://www.opengis.net/def/samplingFeatureType/OGC-OM/2.0/SF_SamplingPoint",
    "sampledFeature@link": {
      "href": "https://api.usgs.gov/collections/hydrological_features/items/112TRRC?f=json",
      "type": "application/geo+json",
      "title": "Aquifer 112TRRC"
    }
  }
}
```

## Example 2 – A Specimen Feature in GeoJSON format

```
{
  "type": "Feature",
  "id": "f6b464cf",
  "geometry": {
    "type": "Point",
    "coordinates": [ 30.706, -134.196, 272 ]
  },
  "properties": {
    "uid": "urn:x-csiro:samples:1114457888",
    "name": "Rock Sample CSIRO:1114457888",
    "description": "Rock sample collected on traverse",
    "featureType": "http://www.opengis.net/def/samplingFeatureType/OGC-OM/2.0/SF_Specimen",
    "samplingTime": "2007-01-24T12:14:50Z",
    "materialClass": "http://dbpedia.org/resource/Rock_(geology)",
    "sampledFeature@link": {
      "href": "https://api.usgs.gov/collections/geological_features/items/1458955?f=json",
      "type": "application/geo+json",
      "title": "Geological Unit 235"
    }
  },
  "links": [
    {
      "rel": "parentSystem",
      "href": "https://data.example.org/api/systems/2ad45f69?f=json",
      "type": "application/geo+json",
      "title": "Field Technician #123 (Rock Sampler)"
    }
  ]
}
```

## Example 3 – A System Part Feature in GeoJSON format

```
{
  "type": "Feature",
  "id": "1a0f80f9",
  "geometry": null,
  "properties": {
    "uid": "urn:x-ogc:sf:456",
    "name": "CPU 2",
    "description": "CPU 2 located in the robot chassis",
    "featureType": "http://www.opengis.net/def/samplingFeatureType/OGC-SML/2.0/FeaturePart",
    "sampledFeature@link": {
      "href": "https://data.example.org/api/systems/8624d054?f=json",
      "type": "application/geo+json",
      "title": "Tactical Ground Robot 457"
    }
  },
  "links": [
    {
      "rel": "parentSystem",
      "href": "https://data.example.org/api/systems/447845?f=json",
      "type": "application/geo+json",
      "title": "Water Level Sensor"
    }
  ]
}
```

## 19.2. Requirements Class “SensorML Format”

### 19.2.1. Overview

REQUIREMENTS CLASS 13	
IDENTIFIER	/req/sensorml
TARGET TYPE	Web API
CONFORMANCE CLASS	Conformance class A.13: /conf/sensorml
PREREQUISITES	Requirements class 1: /req/api-common <a href="http://www.opengis.net/spec/sensorML/3.0/req/json-simple-process">http://www.opengis.net/spec/sensorML/3.0/req/json-simple-process</a> <a href="http://www.opengis.net/spec/sensorML/3.0/req/json-physical-system">http://www.opengis.net/spec/sensorML/3.0/req/json-physical-system</a> <a href="http://www.opengis.net/spec/sensorML/3.0/req/json-deployment">http://www.opengis.net/spec/sensorML/3.0/req/json-deployment</a> <a href="http://www.opengis.net/spec/sensorML/3.0/req/json-derived-property">http://www.opengis.net/spec/sensorML/3.0/req/json-derived-property</a>
NORMATIVE STATEMENTS	Requirement 89: /req/sensorml/mediatype-read Requirement 90: /req/sensorml/mediatype-write Requirement 91: /req/sensorml/relation-types Requirement 92: /req/sensorml/resource-id Requirement 93: /req/sensorml/feature-attribute-mapping Requirement 94: /req/sensorml/system-schema Requirement 95: /req/sensorml/system-sml-class Requirement 96: /req/sensorml/system-mappings Requirement 97: /req/sensorml/deployment-schema Requirement 98: /req/sensorml/deployment-mappings Requirement 99: /req/sensorml/procedure-schema Requirement 100: /req/sensorml/procedure-sml-class Requirement 101: /req/sensorml/procedure-mappings Requirement 102: /req/sensorml/property-schema Requirement 103: /req/sensorml/property-mappings

The “SensorML Format” requirements class specifies how resources defined by the CS API Standard are encoded using the SensorML JSON format defined in SensorML 3.0.

All feature types defined by this Standard, except Sampling Features, can be encoded in SensorML. The SensorML format allows the provision of more advanced metadata such as application specific identifiers and classifiers, security and legal constraints, characteristics and capabilities, contact information, attached documents, reference frames, etc.

## 19.2.2. Media Type

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

The media type used for SensorML encoded resources is **application/sml+json**.

### REQUIREMENT 89

**IDENTIFIER** /req/sensorml/mediatype-read

**INCLUDED IN** Requirements class 13: /req/sensorml

**STATEMENT** The server SHALL support the media type **application/sml+json** in the HTTP Accept header and respond with a JSON document corresponding to the requested resource type.

### REQUIREMENT 90

**IDENTIFIER** /req/sensorml/mediatype-write

**INCLUDED IN** Requirements class 13: /req/sensorml

**CONDITIONS** The server implements /req/create-replace-delete.

**STATEMENT** The server SHALL support the media type **application/sml+json** in the HTTP Content-Type header and parse the JSON body according to the resource type.

## 19.2.3. Link Relation Types

### REQUIREMENT 91

**IDENTIFIER** /req/sensorml/relation-types

**INCLUDED IN** Requirements class 13: /req/sensorml

**STATEMENT** For all associations encoded in the links member of the JSON response, the link relation type must be set to the association name.

## 19.2.4. Common Encoding Rules

The following requirement provides the mapping between common resource properties and the corresponding JSON members when encoded in SensorML-JSON.

REQUIREMENT 92	
IDENTIFIER	/req/sensorml/resource-id
INCLUDED IN	Requirements class 13: /req/sensorml
A	The JSON member id SHALL be used to provide the local identifier of the resource.
B	The value of the id JSON member SHALL be the same as the {id} portion in the URL.

REQUIREMENT 93	
IDENTIFIER	/req/sensorml/feature-attribute-mapping
INCLUDED IN	Requirements class 13: /req/sensorml
STATEMENT	A SensorML JSON document representing a Feature resource SHALL implement the mappings specified in Table 29.

**Table 29** – SensorML Mappings of Common Attributes

ATTRIBUTE NAME	JSON MEMBER	USAGE
uniqueIdentifier	uniqueId	Value is a JSON string that is a valid URI.
name	label	Value is a JSON string.
description	description	Value is a JSON string.

## 19.2.5. System Representation

## REQUIREMENT 94

**IDENTIFIER** /req/sensorml/system-schema

**INCLUDED IN** Requirements class 13: /req/sensorml

**CONDITIONS** The server implements /req/system-features.

**A** A request or response body with media type `application/sml+json` containing a single System resource SHALL be valid against the JSON schema [system.json](#).

**B** A request or response body with media type `application/sml+json` containing a collection of System resources SHALL be valid against the JSON schema [systemCollection.json](#).

## REQUIREMENT 95

**IDENTIFIER** /req/sensorml/system-sml-class

**INCLUDED IN** Requirements class 13: /req/sensorml

**A** SensorML class `PhysicalComponent` or `PhysicalSystem` SHALL be used to describe hardware equipment or human observers.

**B** SensorML class `SimpleProcess` or `AggregateProcess` SHALL be used to describe a simulation or process.

## REQUIREMENT 96

**IDENTIFIER** /req/sensorml/system-mappings

**INCLUDED IN** Requirements class 13: /req/sensorml

**STATEMENT** A SensorML JSON document representing a System resource SHALL implement the mappings specified in Table 30 and Table 31.

**Table 30** – SensorML Mappings of System Attributes

ATTRIBUTE NAME (SEE TABLE 4)	JSON MEMBER	USAGE
systemType	definition	Use the URI or CURIE from Table 6 as the value.
assetType	classifiers	Use a classifier with definition <code>cs:AssetType</code> , and the URI or CURIE from Table 7 as the value.

ATTRIBUTE NAME (SEE TABLE 4)	JSON MEMBER	USAGE
validTime	validTime	-
location	position	Include location either as a GeoJSON geometry, or as part of a 3D Pose object (see OGC 23-000 for a description of the 3D Pose object).

**Table 31** – SensorML Mappings of System Associations

ASSOCIATION NAME (SEE TABLE 5)	JSON MEMBER	USAGE
systemKind	typeOf	Value is a weblink resolving to a Procedure resource.
parentSystem	attachedTo	Value is a weblink resolving to a System resource.
subsystems	links	Value is a weblink* resolving to a System resources endpoint.
samplingFeatures	links	Value is a weblink* resolving to a Sampling Feature resources endpoint.
deployments	links	Value is a weblink* resolving to a Deployment resources endpoint.
procedures	links	Value is a weblink* resolving to a Procedure resources endpoint.
datastreams	links	Value is a weblink* resolving to a DataStream resources endpoint**.
controlstreams	links	Value is a weblink* resolving to a ControlStream resources endpoint**.

\* The link relation type (rel property) must be set to the association name prefixed by ogc-rel:.

\*\* DataStream and ControlStream resources are defined in OGC API – Connected Systems – Part 2.

**Example – A System Feature in SensorML format:** This is a simple description of a fixed in-situ sensor with identification and contact information.

```
{
  "type": "PhysicalSystem",
  "id": "123",
  "definition": "http://www.w3.org/ns/sosa/Sensor",
  "uniqueId": "urn:x-ogc:systems:001",
  "label": "Outdoor Thermometer 001",
  "description": "Digital thermometer located on first floor window 1",
  "typeOf": {
    "href": "https://data.example.org/api/procedures/TP60S?f=sml",
    "uid": "urn:x-myorg:datasheets:ThermoPro:TP60S:v001",
    "title": "ThermoPro TP60S",
    "type": "application/sml+json"
  },
  "identifiers": [
    {
      "definition": "http://sensorml.com/ont/swe/property/SerialNumber",
```

```

    "label": "Serial Number",
    "value": "0123456879"
  },
  "contacts": [
    {
      "role": "http://sensorml.com/ont/swe/roles/Operator",
      "organisationName": "Field Maintenance Corp."
    }
  ],
  "position": {
    "type": "Point",
    "coordinates": [41.8781, -87.6298]
  }
}

```

## 19.2.6. Deployment Representation

### REQUIREMENT 97

**IDENTIFIER** /req/sensorml/deployment-schema

**INCLUDED IN** Requirements class 13: /req/sensorml

**CONDITIONS** The server implements /req/deployment-features.

**A** A request or response body with media type `application/sml+json` containing a single Deployment resource SHALL be valid against the JSON schema [deployment.json](#).

**B** A request or response body with media type `application/sml+json` containing a collection of Deployment resources SHALL be valid against the JSON schema [deploymentCollection.json](#).

### REQUIREMENT 98

**IDENTIFIER** /req/sensorml/deployment-mappings

**INCLUDED IN** Requirements class 13: /req/sensorml

**STATEMENT** A SensorML JSON document representing a Deployment resource SHALL implement the mappings specified in Table 32 and Table 33.

**Table 32** – SensorML Mappings of Deployment Attributes

ATTRIBUTE NAME (SEE TABLE 10)	JSON MEMBER	USAGE
deploymentType	definition	Use the URI or CURIE identifying the type of deployment as the value.
validTime	validTime	-
location	location	Value is a GeoJSON geometry

**Table 33** – SensorML Mappings of Deployment Associations

ASSOCIATION NAME (SEE TABLE 11)	JSON MEMBER	USAGE
platform	platform	Value is a weblink resolving to a System resource.
deployedSystems	deployedSystems	Value is a JSON Array of DeployedSystem objects, each of which contains a link to a System resource.
parentDeployment	links	Value is a weblink* resolving to a Deployment resource.
subdeployments	links	Value is a weblink* resolving to a Deployment resources endpoint.
featuresOfInterest	links	Value is a weblink* resolving to a Feature resources endpoint.
samplingFeatures	links	Value is a weblink* resolving to a Sampling Feature resources endpoint.
datastreams	links	Value is a weblink* resolving to a DataStream resources endpoint**.
controlstreams	links	Value is a weblink* resolving to a ControlStream resources endpoint**.

\* The link relation type (rel property) must be set to the association name prefixed by ogc-rel:.

\*\* DataStream and ControlStream resources are defined in OGC API – Connected Systems – Part 2.

**Example – A Deployment Feature in SensorML format**

```
{
  "type": "Deployment",
  "id": "iv3f2kcq27gfi",
  "definition": "http://www.w3.org/ns/sosa/Deployment",
  "uniqueId": "urn:x-saildrone:mission:2025",
  "label": "Saildrone - 2017 Arctic Mission",
  "description": "In July 2017, three saildrones were launched from Dutch Harbor, Alaska, in partnership with NOAA Research...",
  "classifiers": [
    {
      "definition": "https://schema.org/DefinedRegion",
      "label": "Region",
    }
  ]
}
```

```

    "value": "Arctic"
  }
],
"contacts": [
  {
    "role": "http://sensorml.com/ont/swe/property/Operator",
    "organisationName": "Saildrone, Inc.",
    "contactInfo": {
      "website": "https://www.saildrone.com/",
      "address": {
        "deliveryPoint": "1050 W. Tower Ave.",
        "city": "Alameda",
        "postalCode": "94501",
        "administrativeArea": "CA",
        "country": "USA"
      }
    }
  },
  {
    "role": "http://sensorml.com/ont/swe/property/DataProvider",
    "organisationName": "NOAA Pacific Marine Environmental Laboratory
(PMEL)",
    "contactInfo": {
      "website": "https://www.pmel.noaa.gov"
    }
  }
],
"validTime": [
  "2017-07-17T00:00:00Z",
  "2017-09-29T00:00:00Z"
],
"location": {
  "type": "Polygon",
  "coordinates": [[
    [-173.70, 53.76],
    [-173.70, 75.03],
    [-155.07, 75.03],
    [-155.07, 53.76],
    [-173.70, 53.76]
  ]]
},
"platform": {
  "system": {
    "href": "https://data.example.org/api/systems/27559?f=sml",
    "uid": "urn:x-saildrone:platforms:SD-1003",
    "title": "Saildrone SD-1003"
  }
},
"deployedSystems": [
  {
    "name": "air_temp_sensor",
    "description": "Air temperature sensor installed in the boom",
    "system": {
      "href": "https://data.example.org/api/systems/41548?f=sml",
      "uid": "urn:x-saildrone:sensors:temp01",
      "title": "Air Temperature Sensor"
    },
    "configuration": {
      "setValues": [{
        "ref": "parameters/sampling_rate",
        "value": 0.1
      }]
    }
  }
]

```

```

    },
    {
      "name": "water_temp_sensor",
      "description": "Water temperature sensor installed on the keel",
      "system": {
        "href": "https://data.example.org/api/systems/36584?f=sml",
        "uid": "urn:x-saildrone:sensors:temp02",
        "title": "Water Temperature Sensor"
      }
    },
    {
      "name": "wind_sensor",
      "description": "Wind sensor installed at the top of the mast",
      "system": {
        "href": "https://data.example.org/api/systems/47752?f=sml",
        "uid": "urn:x-saildrone:sensors:wind01",
        "title": "Wind Speed and Direction Sensor"
      }
    }
  ],
  "links": [
    {
      "rel" : "self",
      "href" : "https://data.example.org/api/deployments/iv3f2kcq27gfi?f=sml",
      "type" : "application/sml+json",
      "title" : "this document"
    }, {
      "rel" : "alternate",
      "href" : "https://data.example.org/api/deployments/iv3f2kcq27gfi?f=json",
      "type" : "application/geo+json",
      "title" : "this resource as GeoJSON"
    }
  ]
}

```

## 19.2.7. Procedure Representation

### REQUIREMENT 99

**IDENTIFIER** /req/sensorml/procedure-schema

**INCLUDED IN** Requirements class 13: /req/sensorml

**CONDITIONS** The server implements /req/procedure-features.

**A** A request or response body with media type `application/sml+json` containing a single Procedure resource SHALL be valid against the JSON schema [procedure.json](#).

**B** A request or response body with media type `application/sml+json` containing a collection of Procedure resources SHALL be valid against the JSON schema [procedureCollection.json](#).

## REQUIREMENT 100

**IDENTIFIER** /req/sensorml/procedure-sml-class

**INCLUDED IN** Requirements class 13: /req/sensorml

**A** SensorML class `PhysicalComponent` or `PhysicalSystem` SHALL be used to describe hardware equipment specifications (i.e. datasheet).

**B** SensorML class `SimpleProcess` or `AggregateProcess` SHALL be used to describe a procedure implemented by humans, such as a methodology or steps.

**C** No position information SHALL be provided as part of the procedure description.

## REQUIREMENT 101

**IDENTIFIER** /req/sensorml/procedure-mappings

**INCLUDED IN** Requirements class 13: /req/sensorml

**STATEMENT** A SensorML JSON document representing a Procedure resource SHALL implement the mappings specified in Table 34 and Table 35.

**Table 34** – SensorML Mappings of Procedure Attributes

ATTRIBUTE NAME (SEE TABLE 14)	JSON MEMBER	USAGE
procedureType	definition	Use the URI or CURIE from Table 16 as the value.
validTime	validTime	-

**Table 35** – SensorML Mappings of Procedure Associations

ASSOCIATION NAME (SEE TABLE 15)	JSON MEMBER	USAGE
implementingSyst	links	Value is a weblink* resolving to a System resources endpoint.

\* The link relation type (`rel` property) must be set to the association name prefixed by `ogc-rel:`.

### Example – A Procedure Feature (datasheet) in SensorML format

```
{
```

```

"type": "PhysicalComponent",
"id": "iv3f2kcq27gfi",
"definition": "http://www.w3.org/ns/ssn-system/SensorKind",
"uniqueId": "urn:osh:sensors:saildrone:S0004",
"label": "3D Ultrasonic Anemometer",
"description": "Precision 3-axis ultrasonic anemometer",
"identifiers": [
  {
    "definition": "http://sensorml.com/ont/swe/property/Manufacturer",
    "label": "Manufacturer Name",
    "value": "Gill"
  },
  {
    "definition": "http://sensorml.com/ont/swe/property/ModelNumber",
    "label": "Model Number",
    "value": "WindMaster"
  }
],
"classifiers": [
  {
    "definition": "http://sensorml.com/ont/swe/property/SensorType",
    "label": "Sensor Type",
    "value": "Anemometer"
  }
],
"capabilities": [
  {
    "definition": "http://www.w3.org/ns/ssn/systems/SystemCapability",
    "label": "Speed Measurement Capabilities",
    "capabilities": [
      {
        "name": "range",
        "type": "QuantityRange",
        "definition": "http://www.w3.org/ns/ssn/systems/MeasurementRange",
        "label": "Measurement Range",
        "uom": {
          "code": "m/s"
        },
        "value": [0.0,50.0]
      },
      {
        "name": "resolution",
        "type": "Quantity",
        "definition": "http://www.w3.org/ns/ssn/systems/Resolution",
        "label": "Resolution",
        "uom": {
          "code": "m/s"
        },
        "value": 0.01
      },
      {
        "name": "accuracy",
        "type": "Quantity",
        "definition": "http://sensorml.com/ont/swe/property/
RelativeAccuracy",
        "label": "Relative Accuracy",
        "uom": {
          "code": "%"
        },
        "value": 1.5
      }
    ]
  }
]
}

```

```

],
"links": [
  {
    "href" : "https://data.example.org/api/procedures/iv3f2kcq27gfi?f=sml",
    "rel" : "self",
    "type" : "application/sml+json",
    "title" : "this document"
  }, {
    "href" : "https://data.example.org/api/procedures/iv3f2kcq27gfi?f=json",
    "rel" : "alternate",
    "type" : "application/geo+json",
    "title" : "this resource as GeoJSON"
  }
]
}

```

## 19.2.8. Property Representation

REQUIREMENT 102	
IDENTIFIER	/req/sensorml/property-schema
INCLUDED IN	Requirements class 13: /req/sensorml
CONDITIONS	The server implements /req/property-definitions.
A	A request or response body with media type <code>application/sml+json</code> containing a single Property resource SHALL be valid against the JSON schema <a href="#">property.json</a> .
B	A request or response body with media type <code>application/sml+json</code> containing a collection of Property resources SHALL be valid against the JSON schema <a href="#">propertyCollection.json</a> .

REQUIREMENT 103	
IDENTIFIER	/req/sensorml/property-mappings
INCLUDED IN	Requirements class 13: /req/sensorml
STATEMENT	A SensorML JSON document representing a Property resource SHALL implement the mappings specified in Table 36.

**Table 36** – SensorML Mappings of Property Attributes

ATTRIBUTE NAME (SEE TABLE 19)	JSON MEMBER	USAGE
baseProperty	baseProperty	Value is a JSON string that is a valid URI.
objectType	objectType	Value is a JSON string that is a valid URI.
statistic	statistic	Value is a JSON string that is a valid URI.

**Example – A Property Definition in SensorML format**

```
{  
  "id": "AverageCpuTemp",  
  "label": "Average CPU Temp",  
  "description": "Hourly average of the CPU temperature",  
  "baseProperty": "http://qudt.org/vocab/quantitykind/Temperature",  
  "objectType": "http://dbpedia.org/resource/Central_processing_unit",  
  "statistic": "http://sensorml.com/ont/x-stats/HourlyMean"  
}
```



# ANNEX A (NORMATIVE) CONFORMANCE CLASS ABSTRACT TEST SUITE

---



# ANNEX A (NORMATIVE) CONFORMANCE CLASS ABSTRACT TEST SUITE

## A.1. Supporting Tests

These tests are not associated to any specific requirement but are referenced by other tests.

### ABSTRACT TEST A.1

<b>IDENTIFIER</b>	<code>/conf/api-common/canonical-resources</code>
<b>TEST PURPOSE</b>	Retrieve all items from their canonical endpoint. <i>This is a parameterized test that requires the <code>resource-type</code> parameter</i>
<b>TEST METHOD</b>	Given the <code>resource-type</code> parameter: <ol style="list-style-type: none"><li>1. Issue an HTTP GET request to the URL <code>{resource-type}</code> with the Accepted header set to a media type supported by the server.</li><li>2. Validate that a document was returned with a status code 200.</li><li>3. Iterate through the list of resources in the response, following next links as appropriate.</li></ol>

### ABSTRACT TEST A.2

<b>IDENTIFIER</b>	<code>/conf/api-common/collection-items</code>
<b>TEST PURPOSE</b>	Retrieve items of a collection. <i>This is a parameterized test that requires the <code>collectionId</code> parameter</i>
<b>TEST METHOD</b>	<ol style="list-style-type: none"><li>1. Retrieve the media types supported by the server for the collection items by inspecting the links with relation type <code>items</code>.</li><li>2. If the server does not offer any media type supported by the testing engine, issue a warning and skip the test.</li><li>3. Issue an HTTP GET request to the URL <code>/collections/{collectionId}/items</code> where <code>{collectionId}</code> is the ID of the desired collection.</li></ol>

## ABSTRACT TEST A.2

4. Validate that a document was returned with a status code 200.
5. Iterate through all items in the collection, following next links as appropriate.

## A.2. Conformance Class “Common”

### CONFORMANCE CLASS A.1

IDENTIFIER	/conf/api-common
REQUIREMENTS CLASS	Requirements class 1: /req/api-common
PREREQUISITES	<a href="http://www.opengis.net/spec/ogcapi-1/1.0/conf/core">http://www.opengis.net/spec/ogcapi-1/1.0/conf/core</a> <a href="http://www.opengis.net/spec/ogcapi-common-1/1.0/conf/core">http://www.opengis.net/spec/ogcapi-common-1/1.0/conf/core</a> <a href="http://www.opengis.net/spec/ogcapi-common-1/1.0/conf/landing-page">http://www.opengis.net/spec/ogcapi-common-1/1.0/conf/landing-page</a> <a href="http://www.opengis.net/spec/ogcapi-common-2/1.0/conf/collections">http://www.opengis.net/spec/ogcapi-common-2/1.0/conf/collections</a> <a href="http://www.opengis.net/spec/ogcapi-common-2/1.0/conf/simple-query">http://www.opengis.net/spec/ogcapi-common-2/1.0/conf/simple-query</a>
TARGET TYPE	Web API
CONFORMANCE TESTS	Abstract test A.3: /conf/api-common/resource-ids Abstract test A.4: /conf/api-common/resource-uids Abstract test A.5: /conf/api-common/resource-uids-types Abstract test A.6: /conf/api-common/datetime

### ABSTRACT TEST A.3

IDENTIFIER	/conf/api-common/resource-ids
REQUIREMENT	Requirement 1: /req/api-common/resource-ids
TEST PURPOSE	Validate that resource IDs assigned by the server are unique for a given resource type.
TEST METHOD	For each resource type supported by the server: <ol style="list-style-type: none"><li>1. Retrieve all resources as described in test /conf/api-common/canonical-resources.</li><li>2. For each resource in the response, retrieve its local ID.</li><li>3. Compare the ID with all IDs read previously from resources of the same type and make sure it is unique.</li></ol>

## ABSTRACT TEST A.4

**IDENTIFIER** /conf/api-common/resource-uids

**REQUIREMENT** Requirement 2: /req/api-common/resource-uids

**TEST PURPOSE** Validate that resource UIDs exposed by the server are unique across all collections.

For each resource type supported by the server:

1. Retrieve resources as described in test /conf/api-common/canonical-resources.
2. Retrieve the unique ID of each resource in the response.
3. Compare the UID with all UIDs previously read from any resource and make sure it is unique.

## ABSTRACT TEST A.5

**IDENTIFIER** /conf/api-common/resource-uids-types

**REQUIREMENT** Recommendation 1: /rec/api-common/resource-uids-types

**TEST PURPOSE** Validate that resource UIDs are valid URIs, with a high probability of uniqueness.

For each resource type supported by the server:

1. Retrieve resources as described in test /conf/api-common/canonical-resources.
2. Retrieve the unique ID of each resource in the response.
3. Validate that the unique ID is either a UUID or a URN with a known registered namespace. Issue a warning if not.

## ABSTRACT TEST A.6

**IDENTIFIER** /conf/api-common/datetime

**REQUIREMENT** Requirement 3: /req/api-common/datetime

**TEST PURPOSE** Validate that the server correctly filters features when the datetime query parameter is set.

For each collection advertised by the server:

1. Retrieve the temporal extent of the collection.
2. Execute the Date/Time parameter test of <http://www.opengis.net/spec/ogcapi-features-1/1.0/conf/core/fc-time-response>, using the validTime property of the features in the response as the temporal geometry.

## A.3. Conformance Class “System Features”

### CONFORMANCE CLASS A.2

**IDENTIFIER** /conf/system

**REQUIREMENTS CLASS** Requirements class 2: /req/system

**PREREQUISITE** Conformance class A.1: /conf/api-common

**TARGET TYPE** Web API

**CONFORMANCE TESTS**

- Abstract test A.7: /conf/system/location
- Abstract test A.8: /conf/system/location-time
- Abstract test A.9: /conf/system/canonical-url
- Abstract test A.10: /conf/system/resources-endpoint
- Abstract test A.11: /conf/system/canonical-endpoint
- Abstract test A.12: /conf/system/collections

### ABSTRACT TEST A.7

**IDENTIFIER** /conf/system/location

**REQUIREMENT** Recommendation 2: /rec/system/location

**TEST PURPOSE** Validate that system features include a location.

**TEST METHOD**

1. Retrieve System resources by running test /conf/api-common/canonical-resources with resource-type=systems.
2. For each System resource in the response that does not have assetType set to Simulation or Process, check that the resource representation contains a location. Issue a warning if not.

### ABSTRACT TEST A.8

**IDENTIFIER** /conf/system/location-time

**REQUIREMENT** Requirement 4: /req/system/location-time

**TEST PURPOSE** Validate that the server updates the system location when it changes.

## ABSTRACT TEST A.8

	Given the ID <code>sysId</code> of a mobile system that is known to change location.
TEST METHOD	<ol style="list-style-type: none"><li>1. Issue an HTTP GET request to the URL <code>{api_root}/systems/{sysId}</code>.</li><li>2. Wait until the system has changed location.</li><li>3. Issue an HTTP GET request to the URL <code>{api_root}/systems/{sysId}</code>.</li><li>4. Verify that the locations reported in the two responses are different.</li></ol>

## ABSTRACT TEST A.9

IDENTIFIER `/conf/system/canonical-url`

REQUIREMENT Requirement 5: `/req/system/canonical-url`

TEST PURPOSE Validate that every System resource is accessible via its canonical URL.

	For every collection advertised by the server with the <code>featureType</code> property set to <code>sosa: System</code> :
TEST METHOD	<ol style="list-style-type: none"><li>1. Retrieve the collection items as described in test <code>/conf/api-common/collection-items</code>.</li><li>2. For each item, check that a link with relation type <code>canonical</code> is included.</li><li>3. Dereference this link and validate that a document is returned with a status code 200.</li><li>4. Check that the returned document has the same content as the resource originally included in the collection items (except for the canonical link).</li></ol>

## ABSTRACT TEST A.10

IDENTIFIER `/conf/system/resources-endpoint`

REQUIREMENT Requirement 6: `/req/system/resources-endpoint`

TEST PURPOSE Validate that the server implements a System resources endpoint correctly.  
*This is a parameterized test that requires the endpoint URL as a parameter*

TEST METHOD	<ol style="list-style-type: none"><li>1. Issue an HTTP GET request to the endpoint URL.</li><li>2. Validate that a document was returned with a status code 200.</li><li>3. Validate that the contents of the returned document conform to the media type reported by the response <code>Content-Type</code> header.<ol style="list-style-type: none"><li>a) If the response content type is <code>application/geo+json</code>, execute test <code>/conf/geojson/system-schema</code>.</li><li>b) If the response content type is <code>application/sml+json</code>, execute test <code>/conf/sensorml/system-schema</code>.</li><li>c) For other response content types not supported by the testing engine, issue a warning and skip this test.</li></ol></li></ol>
-------------	---

## ABSTRACT TEST A.11

**IDENTIFIER** /conf/system/canonical-endpoint

**REQUIREMENT** Requirement 7: /req/system/canonical-endpoint

**TEST PURPOSE** Validate that the server exposes the canonical System resources endpoint.

**TEST METHOD** Validate that the server implements a System resources endpoint at path {api\_root}/systems using test /conf/system/resources-endpoint.

## ABSTRACT TEST A.12

**IDENTIFIER** /conf/system/collections

**REQUIREMENT** Requirement 8: /req/system/collections

**TEST PURPOSE** Validate that System collections are tagged with the proper feature type.

For every collection advertised by the server with the featureType property set to sosa: System:

1. Retrieve the collection items as described in test /conf/api-common/collection-items.
2. For each item, retrieve its type.
3. Check that the reported type is one of the URI or CURIE listed in Table 6.

**TEST METHOD**

4. 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/geo+json, execute test /conf/geojson/system-schema.
  - b) If the response content type is application/sml+json, execute test /conf/sensorml/system-schema.
  - c) For other response content types not supported by the testing engine, issue a warning and skip this test.

## A.4. Conformance Class “Subsystems”

---

### CONFORMANCE CLASS A.3

**IDENTIFIER** /conf/subsystem

## CONFORMANCE CLASS A.3

**REQUIREMENTS CLASS** Requirements class 3: /req/subsystem

**PREREQUISITE** Conformance class A.2: /conf/system

**TARGET TYPE** Web API

**CONFORMANCE TESTS** Abstract test A.13: /conf/subsystem/collection  
Abstract test A.14: /conf/subsystem/recursive-param  
Abstract test A.15: /conf/subsystem/recursive-search-systems  
Abstract test A.16: /conf/subsystem/recursive-search-subsystems  
Abstract test A.17: /conf/subsystem/recursive-assoc

## ABSTRACT TEST A.13

**IDENTIFIER** /conf/subsystem/collection

**REQUIREMENT** Requirement 9: /req/subsystem/collection

**TEST PURPOSE** Verify that subsystems are available as a sub-collection of a parent system.

Given the ID sysId of a parent system that has subsystems:

1. Retrieve the parent system resource at {api\_root}/systems/{sysId}.
2. Verify that the response contains a link with relation type subsystems.
3. Verify that the link target is the URL {api\_root}/systems/{id}/subsystems.
4. Dereference this link and validate that a document is returned with a status code 200.

**TEST METHOD**

5. 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/geo+json, execute test /conf/geojson/system-schema.
  - b) If the response content type is application/sml+json, execute test /conf/sensorml/system-schema.
  - c) For other response content types not supported by the testing engine, issue a warning and skip this test.

## ABSTRACT TEST A.14

**IDENTIFIER** /conf/subsystem/recursive-param

**REQUIREMENT** Requirement 10: /req/subsystem/recursive-param

## ABSTRACT TEST A.14

**TEST PURPOSE** Validate that the recursive query parameter is of type boolean

**TEST METHOD**

1. Validate that the request contains a query parameter named `recursive`.
2. Validate that the parameter value is set to `true` or `false`.

## ABSTRACT TEST A.15

**IDENTIFIER** `/conf/subsystem/recursive-search-systems`

**REQUIREMENT** Requirement 11: `/req/subsystem/recursive-search-systems`

**TEST PURPOSE** Verify that subsystems can be queried using the recursive query parameter.

**TEST METHOD**

1. Issue an HTTP GET request to the URL `{api_root}/systems`.
2. Verify that subsystems are not included in the response.
3. Issue an HTTP GET request to the URL `{api_root}/systems?recursive=false`.
4. Verify that subsystems are not included in the response.
5. Issue an HTTP GET request to the URL `{api_root}/systems?recursive=true`.
6. Verify that all subsystems (at all nesting levels) are included in the response.

## ABSTRACT TEST A.16

**IDENTIFIER** `/conf/subsystem/recursive-search-subsystems`

**REQUIREMENT** Requirement 12: `/req/subsystem/recursive-search-subsystems`

**TEST PURPOSE** Verify that nested subsystems can be queried using the recursive query parameter.

**TEST METHOD**

Given the ID `sysId` of a parent system that has subsystems:

1. Issue an HTTP GET request to the URL `{api_root}/systems/{sysId}/subsystems`.
2. Verify that only direct subsystems are included in the response.
3. Issue an HTTP GET request to the URL `{api_root}/systems/{sysId}/subsystems?recursive=false`.
4. Verify that only direct subsystems are included in the response.
5. Issue an HTTP GET request to the URL `{api_root}/systems/{sysId}/subsystems?recursive=true`.
6. Verify that all subsystems (at all nesting levels) are included in the response.

## ABSTRACT TEST A.17

**IDENTIFIER** /conf/subsystem/recursive-assoc

**REQUIREMENT** Requirement 13: /req/subsystem/recursive-assoc

**TEST PURPOSE** Verify that a system's nested resources endpoints include resources from its subsystems.

**TEST METHOD**

For each System resource with local ID sysId that has subsystems:

1. If the server implements Sampling Feature resources, verify that all nested sampling features are returned
  - a) Issue an HTTP GET request to the URL {api\_root}/systems/{sysId}/samplingFeatures.
  - b) Validate that a document was returned with a status code 200.
  - c) Validate that the response includes all Sampling Feature resources associated to the parent system or any of its subsystems (at all nesting levels).
2. If the server implements DataStream resources, verify that all nested datastreams are returned
  - a) Issue an HTTP GET request to the URL {api\_root}/systems/{sysId}/datastreams.
  - b) Validate that a document was returned with a status code 200.
  - c) Validate that the response includes all DataStream resources associated to the parent system or any of its subsystems (at all nesting levels).
3. If the server implements ControlStream resources, verify that all nested controlstreams are returned
  - a) Issue an HTTP GET request to the URL {api\_root}/systems/{sysId}/controlstreams.
  - b) Validate that a document was returned with a status code 200.
  - c) Validate that the response includes all ControlStream resources associated to the parent system or any of its subsystems (at all nesting levels).

## A.5. Conformance Class “Deployment Features”

### CONFORMANCE CLASS A.4

**IDENTIFIER** /conf/deployment

**REQUIREMENTS CLASS** Requirements class 4: /req/deployment

**PREREQUISITE** Conformance class A.1: /conf/api-common

## CONFORMANCE CLASS A.4

**TARGET TYPE** Web API

**CONFORMANCE TESTS** Abstract test A.18: /conf/deployment/canonical-url  
Abstract test A.19: /conf/deployment/resources-endpoint  
Abstract test A.20: /conf/deployment/canonical-endpoint  
Abstract test A.22: /conf/deployment/ref-from-system  
Abstract test A.21: /conf/deployment/collections

## ABSTRACT TEST A.18

**IDENTIFIER** /conf/deployment/canonical-url

**REQUIREMENT** Requirement 14: /req/deployment/canonical-url

**TEST PURPOSE** Validate that every Deployment resource is accessible via its canonical URL.

For every collection advertised by the server with the featureType property set to `sosa:Deployment`:

**TEST METHOD**

1. Retrieve the collection items as described in test /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.19

**IDENTIFIER** /conf/deployment/resources-endpoint

**REQUIREMENT** Requirement 15: /req/deployment/resources-endpoint

**TEST PURPOSE** Validate that the server implements a Deployment 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/geo+json`, execute test /conf/geojson/deployment-schema.
  - b) If the response content type is `application/sml+json`, execute test /conf/sensorml/deployment-schema.

## ABSTRACT TEST A.19

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

## ABSTRACT TEST A.20

**IDENTIFIER** /conf/deployment/canonical-endpoint

**REQUIREMENT** Requirement 16: /req/deployment/canonical-endpoint

**TEST PURPOSE** Validate that the server exposes the canonical Deployment resources endpoint.

**TEST METHOD** Validate that the server implements a Deployment resources endpoint at path {api\_root}/deployments using test /conf/deployment/resources-endpoint.

## ABSTRACT TEST A.21

**IDENTIFIER** /conf/deployment/collections

**REQUIREMENT** Requirement 18: /req/deployment/collections

**TEST PURPOSE** Validate that Deployment collections are tagged with the proper feature type.

For every collection advertised by the server with the featureType property set to sosa : Deployment:

1. Retrieve the collection items as described in test /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.

**TEST METHOD**

- a) If the response content type is application/geo+json, execute test /conf/geojson/deployment-schema.
- b) If the response content type is application/sml+json, execute test /conf/sensorml/deployment-schema.
- c) For other response content types not supported by the testing engine, issue a warning and skip this test.

## ABSTRACT TEST A.22

**IDENTIFIER** /conf/deployment/ref-from-system

**REQUIREMENT** Requirement 17: /req/deployment/ref-from-system

**TEST PURPOSE** Validate that Deployment resources associated to a System are available as sub-resources.

## ABSTRACT TEST A.22

### TEST METHOD

1. Retrieve all System resources by executing test `/conf/api-common/canonical-resources` with parameter `resource-type=systems`.
2. For each System resource in the response:
  - a) Issue an HTTP GET request at path `{api_root}/systems/{sysId}/deployments`, where `sysId` is the local ID of the System 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`, execute test `/conf/geojson/deployment-schema`.
  - e) If the response content type is `{sensorml-mediatype}`, execute test `/conf/sensorml/deployment-schema`.
  - f) Check that the Deployment resource contains a link to the System with ID `sysId`.

## A.6. Conformance Class “Subdeployments”

### CONFORMANCE CLASS A.5

**IDENTIFIER** `/conf/subdeployment`

**REQUIREMENTS CLASS** Requirements class 5: `/req/subdeployment`

**PREREQUISITE** Conformance class A.4: `/conf/deployment`

**TARGET TYPE** Web API

**CONFORMANCE TESTS**

- Abstract test A.23: `/conf/subdeployment/collection`
- Abstract test A.24: `/conf/subdeployment/recursive-param`
- Abstract test A.25: `/conf/subdeployment/recursive-search-deployments`
- Abstract test A.26: `/conf/subdeployment/recursive-search-subdeployments`
- Abstract test A.27: `/conf/subdeployment/recursive-assoc`

### ABSTRACT TEST A.23

**IDENTIFIER** `/conf/subdeployment/collection`

**REQUIREMENT** Requirement 19: `/req/subdeployment/collection`

## ABSTRACT TEST A.23

**TEST PURPOSE** Verify that subdeployments are available as a sub-collection of a parent deployment.

Given the ID `depId` of a parent deployment that has subdeployments:

1. Retrieve the parent deployment resource at `{api_root}/deployments/{depId}`.
2. Verify that the response contains a link with relation type `subdeployments`.
3. Verify that the link target is the URL `{api_root}/deployments/{id}/subdeployments`.
4. Dereference this link and validate that a document is returned with a status code 200.

### TEST METHOD

5. 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/geo+json`, execute test `/conf/geojson/deployment-schema`.
  - b) If the response content type is `application/sml+json`, execute test `/conf/sensorml/deployment-schema`.
  - c) For other response content types not supported by the testing engine, issue a warning and skip this test.

## ABSTRACT TEST A.24

**IDENTIFIER** `/conf/subdeployment/recursive-param`

**REQUIREMENT** Requirement 20: `/req/subdeployment/recursive-param`

**TEST PURPOSE** Validate that the recursive query parameter is of type boolean

- TEST METHOD**
1. Validate that the request contains a query parameter named `recursive`.
  2. Validate that the parameter value is set to `true` or `false`.

## ABSTRACT TEST A.25

**IDENTIFIER** `/conf/subdeployment/recursive-search-deployments`

**REQUIREMENT** Requirement 21: `/req/subdeployment/recursive-search-deployments`

**TEST PURPOSE** Verify that subdeployments can be queried using the recursive query parameter.

- TEST METHOD**
1. Issue an HTTP GET request to the URL `{api_root}/deployments`.
  2. Verify that subdeployments are not included in the response.
  3. Issue an HTTP GET request to the URL `{api_root}/deployments?recursive=false`.

## ABSTRACT TEST A.25

4. Verify that subdeployments are not included in the response.
5. Issue an HTTP GET request to the URL `{api_root}/deployments?recursive=true`.
6. Verify that all subdeployments (at all nesting levels) are included in the response.

## ABSTRACT TEST A.26

**IDENTIFIER** `/conf/subdeployment/recursive-search-subdeployments`

**REQUIREMENT** Requirement 22: `/req/subdeployment/recursive-search-subdeployments`

**TEST PURPOSE** Verify that nested subdeployments can be queried using the `recursive` query parameter.

**TEST METHOD**

Given the ID `depId` of a parent deployment that has subdeployments:

1. Issue an HTTP GET request to the URL `{api_root}/deployments/{depId}/subdeployments`.
2. Verify that only direct subdeployments are included in the response.
3. Issue an HTTP GET request to the URL `{api_root}/deployments/{depId}/subdeployments?recursive=false`.
4. Verify that only direct subdeployments are included in the response.
5. Issue an HTTP GET request to the URL `{api_root}/deployments/{depId}/subdeployments?recursive=true`.
6. Verify that all subdeployments (at all nesting levels) are included in the response.

## ABSTRACT TEST A.27

**IDENTIFIER** `/conf/subdeployment/recursive-assoc`

**REQUIREMENT** Requirement 23: `/req/subdeployment/recursive-assoc`

**TEST PURPOSE** Verify that a deployment's nested resources endpoints include resources from its subdeployments.

**TEST METHOD**

For each `Deployment` resource with local ID `depId` that has subdeployments:

1. If the `Deployment` resource contains a link with relation type `deployedSystems`, verify that all deployed systems are returned
  - a) Issue an HTTP GET request to the link URL.
  - b) Validate that a document was returned with a status code 200.
  - c) Validate that the response includes all `System` resources associated to the parent deployment or any of its subdeployments (at all nesting levels).
2. If the `Deployment` resource contains a link with relation type `featuresOfInterest`, verify that all related features of interest are returned

## ABSTRACT TEST A.27

- a) Issue an HTTP GET request to the link URL.
  - b) Validate that a document was returned with a status code 200.
  - c) Validate that the response includes all Feature of Interest resources associated to the parent deployment or any of its subdeployments (at all nesting levels).
3. If the Deployment resource contains a link with relation type samplingFeatures, verify that all related sampling features are returned
    - a) Issue an HTTP GET request to the link URL.
    - b) Validate that a document was returned with a status code 200.
    - c) Validate that the response includes all Sampling Feature resources associated to the parent deployment or any of its subdeployments (at all nesting levels).
  4. If the Deployment resource contains a link with relation type datastreams, verify that all related datastreams are returned
    - a) Issue an HTTP GET request to the link URL.
    - b) Validate that a document was returned with a status code 200.
    - c) Validate that the response includes all DataStream resources associated to the parent deployment or any of its subdeployments (at all nesting levels).
  5. If the Deployment resource contains a link with relation type controlstreams, verify that all related controlstreams are returned
    - a) Issue an HTTP GET request to the link URL.
    - b) Validate that a document was returned with a status code 200.
    - c) Validate that the response includes all ControlStream resources associated to the parent deployment or any of its subdeployments (at all nesting levels).

## A.7. Conformance Class “Procedure Features”

### CONFORMANCE CLASS A.6

IDENTIFIER	/conf/procedure
REQUIREMENTS CLASS	Requirements class 6: /req/procedure
PREREQUISITE	Conformance class A.1: /conf/api-common
TARGET TYPE	Web API
CONFORMANCE TESTS	Abstract test A.28: /conf/procedure/location Abstract test A.29: /conf/procedure/canonical-url Abstract test A.30: /conf/procedure/resources-endpoint

## CONFORMANCE CLASS A.6

Abstract test A.31: /conf/procedure/canonical-endpoint

Abstract test A.32: /conf/procedure/collections

## ABSTRACT TEST A.28

**IDENTIFIER** /conf/procedure/location

**REQUIREMENT** Requirement 24: /req/procedure/location

**TEST PURPOSE** Validate that Procedure features never include a location.

**TEST METHOD**

1. Issue an HTTP GET request to the URL {api\_root}/procedures.
2. Iterate through the items of the response, following next links as appropriate.
3. For each item, check that no location is not provided.
  - a) If the response content type is application/geo+json, check that the geometry member is set to null.
  - b) If the response content type is application/sml+json, check that the position member is not present.
  - c) For other response content types not supported by the testing engine, issue a warning and skip this test.

## ABSTRACT TEST A.29

**IDENTIFIER** /conf/procedure/canonical-url

**REQUIREMENT** Requirement 25: /req/procedure/canonical-url

**TEST PURPOSE** Validate that every Procedure resource is accessible via its canonical URL.

**TEST METHOD**

For every collection advertised by the server with the featureType property set to sosa : Procedure:

1. Retrieve the collection items as described in test /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.30

**IDENTIFIER** /conf/procedure/resources-endpoint

**REQUIREMENT** Requirement 26: /req/procedure/resources-endpoint

**TEST PURPOSE** Validate that the server implements a Procedure 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/geo+json, execute test /conf/geojson/procedure-schema.
  - b) If the response content type is application/sml+json, execute test /conf/sensorml/procedure-schema.
  - c) For other response content types not supported by the testing engine, issue a warning and skip this test.

## ABSTRACT TEST A.31

**IDENTIFIER** /conf/procedure/canonical-endpoint

**REQUIREMENT** Requirement 27: /req/procedure/canonical-endpoint

**TEST PURPOSE** Validate that the server exposes the canonical Procedure resources endpoint.

**TEST METHOD** Validate that the server implements a Procedure resources endpoint at path {api\_root}/procedures using test /conf/procedure/resources-endpoint.

## ABSTRACT TEST A.32

**IDENTIFIER** /conf/procedure/collections

**REQUIREMENT** Requirement 28: /req/procedure/collections

**TEST PURPOSE** Validate that Procedure collections are tagged with the proper feature type.

**TEST METHOD**

For every collection advertised by the server with the featureType property set to sosa:Procedure:

1. Retrieve the collection items as described in test /conf/api-common/collection-items.
2. For each item, retrieve its type.

## ABSTRACT TEST A.32

3. Check that the reported type is one of the URI or CURIE listed in Table 16.
4. 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/geo+json`, execute test `/conf/geojson/procedure-schema`.
  - b) If the response content type is `application/sml+json`, execute test `/conf/sensorml/procedure-schema`.
  - c) For other response content types not supported by the testing engine, issue a warning and skip this test.

## A.8. Conformance Class “Sampling Features”

### CONFORMANCE CLASS A.7

IDENTIFIER	<code>/conf/sf</code>
REQUIREMENTS CLASS	Requirements class 7: <code>/req/sf</code>
PREREQUISITE	Conformance class A.2: <code>/conf/system</code>
TARGET TYPE	Web API
CONFORMANCE TESTS	Abstract test A.33: <code>/conf/sf/canonical-url</code> Abstract test A.34: <code>/conf/sf/resources-endpoint</code> Abstract test A.35: <code>/conf/sf/canonical-endpoint</code> Abstract test A.37: <code>/conf/sf/ref-from-system</code> Abstract test A.36: <code>/conf/sf/collections</code>

### ABSTRACT TEST A.33

IDENTIFIER	<code>/conf/sf/canonical-url</code>
REQUIREMENT	Requirement 29: <code>/req/sf/canonical-url</code>
TEST PURPOSE	Validate that every Sampling Feature resource is accessible via its canonical URL.
TEST METHOD	For every collection advertised by the server with the <code>featureType</code> property set to <code>sosa:Sample</code> : <ol style="list-style-type: none"><li>1. Retrieve the collection items as described in test <code>/conf/api-common/collection-items</code>.</li></ol>

## ABSTRACT TEST A.33

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.34

**IDENTIFIER** `/conf/sf/resources-endpoint`

**REQUIREMENT** Requirement 30: `/req/sf/resources-endpoint`

**TEST PURPOSE** Validate that the server implements a `Sampling Feature 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/geo+json`, execute test `/conf/geojson/sf-schema`.
    - b) For other response content types not supported by the testing engine, issue a warning and skip this test.

## ABSTRACT TEST A.35

**IDENTIFIER** `/conf/sf/canonical-endpoint`

**REQUIREMENT** Requirement 31: `/req/sf/canonical-endpoint`

**TEST PURPOSE** Validate that the server exposes the canonical `Sampling Feature resources` endpoint.

**TEST METHOD** Validate that the server implements a `Sampling Feature resources` endpoint at path `{api_root}/samplingFeatures` using test `/conf/sf/resources-endpoint`.

## ABSTRACT TEST A.36

**IDENTIFIER** `/conf/sf/collections`

**REQUIREMENT** Requirement 33: `/req/sf/collections`

## ABSTRACT TEST A.36

**TEST PURPOSE** Validate that Sampling Feature collections are tagged with the proper feature type.

For every collection advertised by the server with the featureType property set to sosa :  
Sample:

**TEST METHOD**

1. Retrieve the collection items as described in test /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/geo+json, execute test /conf/geojson/sf-schema.
  - b) For other response content types not supported by the testing engine, issue a warning and skip this test.

## ABSTRACT TEST A.37

**IDENTIFIER** /conf/sf/ref-from-system

**REQUIREMENT** Requirement 32: /req/sf/ref-from-system

**TEST PURPOSE** Validate that Sampling Features attached to a given system are available as a sub-resources.

**TEST METHOD**

1. Retrieve all System resources by executing test /conf/api-common/canonical-resources with parameter resource-type=systems.
2. For each System resource in the response:
  - a) Issue an HTTP GET request at path {api\_root}/systems/{sysId}/samplingFeatures, where sysId is the local ID of the System 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, execute test /conf/geojson/sf-schema.

## A.9. Conformance Class “Property Definitions”

### CONFORMANCE CLASS A.8

**IDENTIFIER** /conf/property

**REQUIREMENTS CLASS** Requirements class 8: /req/property

## CONFORMANCE CLASS A.8

**PREREQUISITE** Conformance class A.1: /conf/api-common

**TARGET TYPE** Web API

**CONFORMANCE TESTS**  
Abstract test A.38: /conf/property/canonical-url  
Abstract test A.39: /conf/property/resources-endpoint  
Abstract test A.40: /conf/property/canonical-endpoint  
Abstract test A.41: /conf/property/collections

## ABSTRACT TEST A.38

**IDENTIFIER** /conf/property/canonical-url

**REQUIREMENT** Requirement 34: /req/property/canonical-url

**TEST PURPOSE** Validate that every Property resource is accessible via its canonical URL.

**TEST METHOD**

For every collection advertised by the server with the itemType property set to `sosa:Property`:

1. Retrieve the collection items as described in test /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.39

**IDENTIFIER** /conf/property/resources-endpoint

**REQUIREMENT** Requirement 35: /req/property/resources-endpoint

**TEST PURPOSE** Validate that the server implements a Property 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 `{sensorml-mediatype}`, execute test /conf/sensorml/property-schema.
  - b) For other response content types not supported by the testing engine, issue a warning and skip this test.

## ABSTRACT TEST A.40

**IDENTIFIER** /conf/property/canonical-endpoint

**REQUIREMENT** Requirement 36: /req/property/canonical-endpoint

**TEST PURPOSE** Validate that the server exposes the canonical Property resources endpoint.

**TEST METHOD** Validate that the server implements a Property resources endpoint at path {api\_root}/properties using test /conf/property/resources-endpoint.

## ABSTRACT TEST A.41

**IDENTIFIER** /conf/property/collections

**REQUIREMENT** Requirement 37: /req/property/collections

**TEST PURPOSE** Validate that Property collections are tagged with the proper item type.

**TEST METHOD**

For every collection advertised by the server with the itemType property set to sosa:Property:

1. Retrieve the collection items as described in test /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 {sensorml-mediatype}, execute test /conf/sensorml/property-schema.
  - b) For other response content types not supported by the testing engine, issue a warning and skip this test.

## A.10. Conformance Class “Advanced Filtering”

---

### CONFORMANCE CLASS A.9

**IDENTIFIER** /conf/advanced-filtering

**REQUIREMENTS CLASS** Requirements class 9: /req/advanced-filtering

**PREREQUISITE** Conformance class A.1: /conf/api-common

**TARGET TYPE** Web API

## CONFORMANCE CLASS A.9

### CONFORMANCE TESTS

Abstract test A.42: /conf/advanced-filtering/id-list-schema  
Abstract test A.43: /conf/advanced-filtering/resource-by-id  
Abstract test A.44: /conf/advanced-filtering/resource-by-keyword  
Abstract test A.45: /conf/advanced-filtering/resource-by-property  
Abstract test A.46: /conf/advanced-filtering/feature-by-geom  
Abstract test A.47: /conf/advanced-filtering/system-by-parent  
Abstract test A.48: /conf/advanced-filtering/system-by-procedure  
Abstract test A.49: /conf/advanced-filtering/system-by-foi  
Abstract test A.50: /conf/advanced-filtering/system-by-obsprop  
Abstract test A.51: /conf/advanced-filtering/system-by-controlprop  
Abstract test A.52: /conf/advanced-filtering/deployment-by-parent  
Abstract test A.53: /conf/advanced-filtering/deployment-by-system  
Abstract test A.54: /conf/advanced-filtering/deployment-by-foi  
Abstract test A.55: /conf/advanced-filtering/deployment-by-obsprop  
Abstract test A.56: /conf/advanced-filtering/deployment-by-controlprop  
Abstract test A.57: /conf/advanced-filtering/procedure-by-obsprop  
Abstract test A.58: /conf/advanced-filtering/procedure-by-controlprop  
Abstract test A.59: /conf/advanced-filtering/sf-by-foi  
Abstract test A.60: /conf/advanced-filtering/sf-by-obsprop  
Abstract test A.61: /conf/advanced-filtering/sf-by-controlprop  
Abstract test A.62: /conf/advanced-filtering/prop-by-baseprop  
Abstract test A.63: /conf/advanced-filtering/prop-by-object  
Abstract test A.64: /conf/advanced-filtering/combined-filters  
Abstract test A.65: /conf/advanced-filtering/indirect-prop  
Abstract test A.66: /conf/advanced-filtering/indirect-foi

## ABSTRACT TEST A.42

IDENTIFIER	/conf/advanced-filtering/id-list-schema
REQUIREMENT	Requirement 38: /req/advanced-filtering/id-list-schema
TEST PURPOSE	Validate that query parameters of type ID List are constructed correctly.
TEST METHOD	Validate that the parameter is a comma separated list of string values.

## ABSTRACT TEST A.43

IDENTIFIER	/conf/advanced-filtering/resource-by-id
REQUIREMENT	Requirement 39: /req/advanced-filtering/resource-by-id

## ABSTRACT TEST A.43

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

**TEST METHOD**

For every canonical resources endpoint:

1. Generate an `id` parameter set to a list of resource IDs (see test `/conf/advanced-filtering/id-list-schema`).
2. Issue an HTTP GET request at the resources endpoint URL with the previously generated `id` parameter in the query string.
3. Validate that a document was returned with a status code 200.
4. Validate that the returned collection only includes the resources with the selected identifiers.
5. Repeat the previous steps with an `id` parameter containing a list of UIDs.

## ABSTRACT TEST A.44

**IDENTIFIER** `/conf/advanced-filtering/resource-by-keyword`

**REQUIREMENT** Requirement 40: `/req/advanced-filtering/resource-by-keyword`

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

**TEST METHOD**

For every canonical resources endpoint:

1. Generate a `q` parameter set to a list of keywords, as specified by the provided OpenAPI 3.0 schema.
2. Issue an HTTP GET request at the resources endpoint URL with the `q` parameter in the query string.
3. Validate that a document was returned with a status code 200.
4. Validate that the returned collection only includes resources with plain text content that includes the keyword.

## ABSTRACT TEST A.45

**IDENTIFIER** `/conf/advanced-filtering/resource-by-property`

**REQUIREMENT** Recommendation 3: `/rec/advanced-filtering/resource-by-property`

**TEST PURPOSE** Validate that custom property query parameters are processed correctly.

**TEST METHOD**

For every canonical resources endpoint:

1. Generate a custom parameter with the same name as a feature property, and set the value to a possible value of the property.

## ABSTRACT TEST A.45

2. Issue an HTTP GET request at the resources endpoint URL with the custom parameter in the query string.
3. Validate that a document was returned with a status code 200.
4. Validate that the returned collection only includes resources with matching property values.

## ABSTRACT TEST A.46

**IDENTIFIER** /conf/advanced-filtering/feature-by-geom

**REQUIREMENT** Requirement 41: /req/advanced-filtering/feature-by-geom

**TEST PURPOSE** Validate that the geom query parameter is processed correctly.

**TEST METHOD** For each of the systems, deployments and samplingFeatures canonical resources endpoints:

1. Generate a geom parameter set to a WKT geometry conforming to the provided OpenAPI 3.0 schema.
2. Issue an HTTP GET request at the resources endpoint URL with the geom parameter in the query string.
3. Validate that a document was returned with a status code 200.
4. Validate that the returned collection only includes resources with a geometry intersecting the provided geometry.

## ABSTRACT TEST A.47

**IDENTIFIER** /conf/advanced-filtering/system-by-parent

**REQUIREMENT** Requirement 42: /req/advanced-filtering/system-by-parent

**TEST PURPOSE** Validate that the parent query parameter is processed correctly.

**TEST METHOD**

1. Issue an HTTP GET request at URL {api\_root}/systems?parent={idList} where {idList} is a list of one or more local IDs of System resources. See test /conf/advanced-filtering/id-list-schema
2. Validate the response using the steps described in test /conf/system/resources-endpoint.
3. For each System resource in the returned collection:
  - a) Follow the parentSystem association to retrieve the parent system description.
  - b) Verify that the system has one of the identifiers included in {idList}.
4. Repeat the previous steps with the parent parameter set to a list of one or more UIDs of System resources..

## ABSTRACT TEST A.48

**IDENTIFIER** /conf/advanced-filtering/system-by-procedure

**REQUIREMENT** Requirement 43: /req/advanced-filtering/system-by-procedure

**TEST PURPOSE** Validate that the procedure query parameter is processed correctly.

**TEST METHOD**

1. Issue an HTTP GET request at URL {api\_root}/systems?procedure={idList} where {idList} is a list of one or more local IDs of Procedure resources.  
See test /conf/advanced-filtering/id-list-schema
2. Validate the response using the steps described in test /conf/system/resources-endpoint.
3. For each System resource in the returned collection:
  - a) Follow the procedure association to retrieve the procedure description.
  - b) Verify that the procedure has one of the identifiers included in {idList}.
4. Repeat the previous steps with the procedure parameter set to a list of one or more UIDs of Procedure resources.

## ABSTRACT TEST A.49

**IDENTIFIER** /conf/advanced-filtering/system-by-foi

**REQUIREMENT** Requirement 44: /req/advanced-filtering/system-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}/systems?foi={idList} where {idList} is a list of one or more local IDs of Feature resources.  
See test /conf/advanced-filtering/id-list-schema
2. Validate the response using the steps described in test /conf/system/resources-endpoint.
3. For each System resource in the returned collection:
  - a) Retrieve the system's sampling features by issuing an HTTP GET request at {systemCanonicalUrl}/samplingFeatures?recursive=true.
  - 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.50

**IDENTIFIER** /conf/advanced-filtering/system-by-obsprop

**REQUIREMENT** Requirement 45: /req/advanced-filtering/system-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}/systems?observedProperty={idList} where {idList} is a list of one or more local IDs of Property resources. See test /conf/advanced-filtering/id-list-schema
  2. Validate the response using the steps described in test /conf/system/resources-endpoint.
  3. For each System resource in the returned collection:
    - a) Retrieve all its nested subsystems by issuing an HTTP GET request at {systemCanonicalUrl}/components?recursive=true.
    - b) Retrieve all observed properties referenced by the main system or one of its subsystems.
    - c) 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 URLs of Property resources.

## ABSTRACT TEST A.51

**IDENTIFIER** /conf/advanced-filtering/system-by-controlprop

**REQUIREMENT** Requirement 46: /req/advanced-filtering/system-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}/systems?controlledProperty={idList} where {idList} is a list of one or more local IDs of Property resources. See test /conf/advanced-filtering/id-list-schema
  2. Validate the response using the steps described in test /conf/system/resources-endpoint.
  3. For each System resource in the returned collection:
    - a) Retrieve all its nested subsystems by issuing an HTTP GET request at {systemCanonicalUrl}/components?recursive=true.
    - b) Retrieve all controlled properties referenced by the main system or one of its subsystems.
    - c) 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 URLs of Property resources.

## ABSTRACT TEST A.52

**IDENTIFIER** /conf/advanced-filtering/deployment-by-parent

**REQUIREMENT** Requirement 47: /req/advanced-filtering/deployment-by-parent

**TEST PURPOSE** Validate that the parent query parameter is processed correctly.

**TEST METHOD**

1. Issue an HTTP GET request at URL {api\_root}/deployments?parent={idList} where {idList} is a list of one or more local IDs of Deployment resources. See test /conf/advanced-filtering/id-list-schema
2. Validate the response using the steps described in test /conf/deployment/resources-endpoint.
3. For each Deployment resource in the returned collection:
  - a) Follow the parentSystem association to retrieve the parent deployment description.
  - b) Verify that the deployment has one of the identifiers included in {idList}.
4. Repeat the previous steps with the parent parameter set to a list of one or more UIDs of Deployment resources..

## ABSTRACT TEST A.53

**IDENTIFIER** /conf/advanced-filtering/deployment-by-system

**REQUIREMENT** Requirement 48: /req/advanced-filtering/deployment-by-system

**TEST PURPOSE** Validate that the system query parameter is processed correctly.

**TEST METHOD**

1. Issue an HTTP GET request at URL {api\_root}/deployments?system={idList} where {idList} is a list of one or more local IDs of System resources. See test /conf/advanced-filtering/id-list-schema
2. Validate the response using the steps described in test /conf/deployment/resources-endpoint.
3. For each Deployment resource in the returned collection:
  - a) Retrieve all deployed systems by issuing an HTTP GET request at {deploymentCanonicalUrl}/deployedSystems?recursive=true.
  - b) Verify that at least one of the systems 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 System resources.

## ABSTRACT TEST A.54

**IDENTIFIER** /conf/advanced-filtering/deployment-by-foi

## ABSTRACT TEST A.54

**REQUIREMENT** Requirement 49: /req/advanced-filtering/deployment-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}/deployments?foi={idList} where {idList} is a list of one or more local IDs of Feature resources.  
See test /conf/advanced-filtering/id-list-schema
  2. Validate the response using the steps described in test /conf/deployment/resources-endpoint.
  3. For each Deployment resource in the returned collection:
    - a) Retrieve the deployment's features of interest by issuing an HTTP GET request at {deploymentCanonicalUrl}/featuresOfInterest
    - b) Verify that at least one of the 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.55

**IDENTIFIER** /conf/advanced-filtering/deployment-by-obsprop

**REQUIREMENT** Requirement 50: /req/advanced-filtering/deployment-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}/deployments?observedProperty={idList} where {idList} is a list of one or more local IDs of Property resources.  
See test /conf/advanced-filtering/id-list-schema
  2. Validate the response using the steps described in test /conf/deployment/resources-endpoint.
  3. For each Deployment resource in the returned collection:
    - a) Retrieve all deployed systems by issuing an HTTP GET request at {deploymentCanonicalUrl}/deployedSystems?recursive=true.
    - b) For each Deployed System resource in the returned collection:
      1. Retrieve the system description by following the system association link.
      2. Collect all observed properties referenced by the system description.
    - c) 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 of Property resources.

## ABSTRACT TEST A.56

**IDENTIFIER** /conf/advanced-filtering/deployment-by-controlprop

**REQUIREMENT** Requirement 51: /req/advanced-filtering/deployment-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}/deployments?controlledProperty={idList} where {idList} is a list of one or more local IDs of Property resources. See test /conf/advanced-filtering/id-list-schema
2. Validate the response using the steps described in test /conf/deployment/resources-endpoint.
3. For each Deployment resource in the returned collection:
  - a) Retrieve all deployed systems by issuing an HTTP GET request at {deploymentCanonicalUrl}/deployedSystems?recursive=true.
  - b) For each Deployed System resource in the returned collection:
    1. Retrieve the system description by following the system association link.
    2. Collect all controlled properties referenced by the system description.
  - c) 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 URLs of Property resources.

## ABSTRACT TEST A.57

**IDENTIFIER** /conf/advanced-filtering/procedure-by-obsprop

**REQUIREMENT** Requirement 52: /req/advanced-filtering/procedure-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}/procedures?observedProperty={idList} where {idList} is a list of one or more local IDs of Property resources. See test /conf/advanced-filtering/id-list-schema
2. Validate the response using the steps described in test /conf/procedure/resources-endpoint.
3. For each Procedure resource in the returned collection:
  - a) Retrieve all observed properties referenced by the procedure description.
  - 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 URLs of Property resources.

## ABSTRACT TEST A.58

**IDENTIFIER** /conf/advanced-filtering/procedure-by-controlprop

**REQUIREMENT** Requirement 53: /req/advanced-filtering/procedure-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}/procedures?controlledProperty={idList} where {idList} is a list of one or more local IDs of Property resources. See test /conf/advanced-filtering/id-list-schema
2. Validate the response using the steps described in test /conf/procedure/resources-endpoint.
3. For each Procedure resource in the returned collection:
  - a) Retrieve all controlled properties referenced by the procedure description.
  - 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 URLs of Property resources.

## ABSTRACT TEST A.59

**IDENTIFIER** /conf/advanced-filtering/sf-by-foi

**REQUIREMENT** Requirement 54: /req/advanced-filtering/sf-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}/samplingFeatures?foi={idList} where {idList} is a list of one or more local IDs of Feature resources. See test /conf/advanced-filtering/id-list-schema
2. Validate the response using the steps described in test /conf/sf/resources-endpoint.
3. For each Sampling Feature resource in the returned collection:
  - a) Follow the sampleOf links to collect 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.
4. Verify that at least one of the collected features has one of the identifiers included in {idList}.
5. Repeat the previous steps with the foi parameter set to a list of one or more UIDs of Feature resources.

## ABSTRACT TEST A.60

**IDENTIFIER** /conf/advanced-filtering/sf-by-obsprop

**REQUIREMENT** Requirement 55: /req/advanced-filtering/sf-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}/samplingFeatures?observedProperty={idList} where {idList} is a list of one or more local IDs of Property resources.  
See test /conf/advanced-filtering/id-list-schema
2. Validate the response using the steps described in test /conf/sf/resources-endpoint.
3. For each Sampling Feature resource in the returned collection:
  - a) Follow the datastreams links to get the datastreams containing observations for this sampling feature.
  - b) Verify that at least one of the datastreams has one or more of the observed properties included in {idList}.
4. Repeat the previous steps with the observedProperty parameter set to a list of one or more URLs of Property resources.

## ABSTRACT TEST A.61

**IDENTIFIER** /conf/advanced-filtering/sf-by-controlprop

**REQUIREMENT** Requirement 56: /req/advanced-filtering/sf-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}/samplingFeatures?controlledProperty={idList} where {idList} is a list of one or more local IDs of Property resources.  
See test /conf/advanced-filtering/id-list-schema
2. Validate the response using the steps described in test /conf/sf/resources-endpoint.
3. For each Sampling Feature resource in the returned collection:
  - a) Follow the controlstreams links to get the control streams with commands targeting this sampling feature.
  - b) Verify that at least one of the control streams has one or more of the controlled properties included in {idList}.
4. Repeat the previous steps with the controlledProperty parameter set to a list of one or more URLs of Property resources.

## ABSTRACT TEST A.62

**IDENTIFIER** /conf/advanced-filtering/prop-by-baseprop

**REQUIREMENT** Requirement 57: /req/advanced-filtering/prop-by-baseprop

**TEST PURPOSE** Validate that the baseProperty query parameter is processed correctly.

**TEST METHOD**

1. Issue an HTTP GET request at URL {api\_root}/properties?baseProperty={idList} where {idList} is a list of one or more local IDs of Property resources. See test /conf/advanced-filtering/id-list-schema
2. Validate the response using the steps described in test /conf/property/resources-endpoint.
3. For each Property resource in the returned collection:
  - a) Follow the baseProperty links to collect the base property, 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 property.
4. Verify that at least one of the collected properties has one of the identifiers included in {idList}.
5. Repeat the previous steps with the baseProperty parameter set to a list of one or more URIs of Property resources.

## ABSTRACT TEST A.63

**IDENTIFIER** /conf/advanced-filtering/prop-by-object

**REQUIREMENT** Requirement 58: /req/advanced-filtering/prop-by-object

**TEST PURPOSE** Validate that the objectType query parameter is processed correctly.

**TEST METHOD**

1. Issue an HTTP GET request at URL {api\_root}/properties?objectType={uriList} where {uriList} is a list of one or more URIs of feature/object types. See test /conf/advanced-filtering/id-list-schema
2. Validate the response using the steps described in test /conf/property/resources-endpoint.
3. Verify that each Property resource in the result set has its objectType property set to one of the URIs included in {uriList}.

## ABSTRACT TEST A.64

**IDENTIFIER** /conf/advanced-filtering/combined-filters

**REQUIREMENT** Requirement 59: /req/advanced-filtering/combined-filters

**TEST PURPOSE** Validate that the server correctly implements a logical AND between query filters.

## ABSTRACT TEST A.64

<b>TEST METHOD</b>	For each canonical resources endpoint:
	<ol style="list-style-type: none"><li>1. Issue HTTP GET requests at the resources endpoint URL with different combinations of query parameters that are available for this resource type.</li><li>2. Verify that each resource in the result set passes the checks described in the test corresponding to each filter.</li></ol>

## ABSTRACT TEST A.65

**IDENTIFIER** /conf/advanced-filtering/indirect-prop

**REQUIREMENT** Recommendation 4: /rec/advanced-filtering/indirect-prop

**TEST PURPOSE** Check if the server can follow baseProperty associations transitively.

<b>TEST METHOD</b>	For each Property resource available at the canonical resources endpoint:
	<ol style="list-style-type: none"><li>1. Retrieve the value of the id attribute. Store it as propId.</li><li>2. Retrieve the value of the baseProperty attribute. Store it as basePropId.</li><li>3. Verify that the server supports querying systems using transitive base properties:<ol style="list-style-type: none"><li>a) Issue an HTTP GET request at URL {api_root}/systems?observedProperty={propId}. Store the response as set 1.</li><li>b) Issue an HTTP GET request at URL {api_root}/systems?observedProperty={basePropId}. Store the response as set 2.</li><li>c) Check that set 2 contains all resources from set 1. Issue a warning if not.</li></ol></li><li>4. Verify that the server supports querying deployments using transitive base properties:<ol style="list-style-type: none"><li>a) Issue an HTTP GET request at URL {api_root}/deployments?observedProperty={propId}. Store the response as set 1.</li><li>b) Issue an HTTP GET request at URL {api_root}/deployments?observedProperty={basePropId}. Store the response as set 2.</li><li>c) Check that set 2 contains all resources from set 1. Issue a warning if not.</li></ol></li><li>5. Verify that the server supports querying procedures using transitive base properties:<ol style="list-style-type: none"><li>a) Issue an HTTP GET request at URL {api_root}/procedures?observedProperty={propId}. Store the response as set 1.</li><li>b) Issue an HTTP GET request at URL {api_root}/procedures?observedProperty={basePropId}. Store the response as set 2.</li><li>c) Check that set 2 contains all resources from set 1. Issue a warning if not.</li></ol></li><li>6. Verify that the server supports querying sampling features using transitive base properties:<ol style="list-style-type: none"><li>a) Issue an HTTP GET request at URL {api_root}/systems?observedProperty={propId}. Store the response as set 1.</li><li>b) Issue an HTTP GET request at URL {api_root}/systems?observedProperty={basePropId}. Store the response as set 2.</li></ol></li></ol>

## ABSTRACT TEST A.65

- c) Check that set 2 contains all resources from set 1. Issue a warning if not.
7. Verify that the server supports querying properties using transitive base properties:
  - a) Issue an HTTP GET request at URL {api\_root}/properties?baseProperty={propId}. Store the response as set 1.
  - b) Issue an HTTP GET request at URL {api\_root}/properties?baseProperty={basePropId}. Store the response as set 2.
  - c) Check that set 2 contains all resources from set 1. Issue a warning if not.

## ABSTRACT TEST A.66

**IDENTIFIER** /conf/advanced-filtering/indirect-foi

**REQUIREMENT** Recommendation 5: /rec/advanced-filtering/indirect-foi

**TEST PURPOSE** Check if the server can follow sampledFeature and sampleOf associations transitively.

For each SamplingFeature resource available at the canonical resources endpoint:

1. Retrieve the value of the id attribute. Store it as sfId.
2. Retrieve the feature referenced by the sampledFeature property. Retrieve the value of the feature id attribute and store it as ultimateFoiId.
3. Retrieve one of the features referenced by the sampleOf association. Retrieve the value of the feature id attribute and store it as parentSfId.
4. Verify that the server supports querying sampling features using transitive sampleOf associations:
  - a) Issue an HTTP GET request at URL {api\_root}/samplingFeatures?foi={parentSfId}. Store the response as set 1.
  - b) Issue an HTTP GET request at URL {api\_root}/samplingFeatures?foi={ultimateFoiId}. Store the response as set 2.

**TEST METHOD**

- c) Check that both resource sets contain the Sampling Feature with local ID sfId.
5. Verify that the server supports querying systems using transitive sampled features:
  - a) Issue an HTTP GET request at URL {api\_root}/systems?foi={sfId}. Store the response as set 1.
  - b) Issue an HTTP GET request at URL {api\_root}/systems?foi={parentSfId}. Store the response as set 2.
  - c) Issue an HTTP GET request at URL {api\_root}/systems?foi={ultimateFoiId}. Store the response as set 3.
  - d) Check that set 3 contains all resources from set 1. Issue a warning if not.
  - e) Check that set 3 contains all resources from set 2. Issue a warning if not.
  - f) Check that set 2 contains all resources from set 1. Issue a warning if not.
6. Verify that the server supports querying deployments using transitive base properties:

## ABSTRACT TEST A.66

- a) Issue an HTTP GET request at URL {api\_root}/deployments?foi={sfld}. Store the response as set 1.
- b) Issue an HTTP GET request at URL {api\_root}/deployments?foi={parentSfld}. Store the response as set 2.
- c) Issue an HTTP GET request at URL {api\_root}/deployments?foi={ultimateFoild}. Store the response as set 3.
- d) Check that set 3 contains all resources from set 1. Issue a warning if not.
- e) Check that set 3 contains all resources from set 2. Issue a warning if not.
- f) Check that set 2 contains all resources from set 1. Issue a warning if not.

## A.11. Conformance Class “Create/Replace/Delete”

### CONFORMANCE CLASS A.10

**IDENTIFIER** /conf/create-replace-delete

**REQUIREMENTS CLASS** Requirements class 10: /req/create-replace-delete

**PREREQUISITES** Conformance class A.1: /conf/api-common  
<http://www.opengis.net/spec/ogcapi-4/1.0/conf/create-replace-delete>

**TARGET TYPE** Web API

**CONFORMANCE TESTS** Abstract test A.67: /conf/create-replace-delete/system  
Abstract test A.68: /conf/create-replace-delete/system-delete-cascade  
Abstract test A.69: /conf/create-replace-delete/subsystem  
Abstract test A.70: /conf/create-replace-delete/deployment  
Abstract test A.71: /conf/create-replace-delete/subdeployment  
Abstract test A.72: /conf/create-replace-delete/procedure  
Abstract test A.73: /conf/create-replace-delete/sampling-feature  
Abstract test A.74: /conf/create-replace-delete/property  
Abstract test A.75: /conf/create-replace-delete/create-in-collection  
Abstract test A.76: /conf/create-replace-delete/replace-in-collection  
Abstract test A.77: /conf/create-replace-delete/delete-in-collection  
Abstract test A.78: /conf/create-replace-delete/add-to-collection

## ABSTRACT TEST A.67

**IDENTIFIER** /conf/create-replace-delete/system

**REQUIREMENT** Requirement 60: /req/create-replace-delete/system

**TEST PURPOSE** Validate that the server implements CREATE/REPLACE/DELETE operations correctly on System collections.

**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 (for CREATE)
  - b) At resource endpoint {api\_root}/systems/{id} (for REPLACE and DELETE)

## ABSTRACT TEST A.68

**IDENTIFIER** /conf/create-replace-delete/system-delete-cascade

**REQUIREMENT** Requirement 61: /req/create-replace-delete/system-delete-cascade

**TEST PURPOSE** Validate that the server implements the cascade query parameter correctly.

**TEST METHOD**

1. Given a System resource with ID sysId that has sub-resources:
  - a) Issue an HTTP DELETE request at URL {api\_root}/systems/{sysId}?cascade=false.
  - b) Verify that the server responds with an error code 409.
  - c) Issue an HTTP DELETE request at URL {api\_root}/systems/{sysId}?cascade=true.
  - d) Verify that the system and all its sub-resources have been deleted.
2. Given a System resource with ID sysId that is referenced by a Deployed System resource:
  - a) Issue an HTTP DELETE request at URL {api\_root}/systems/{sysId}?cascade=false.
  - b) Verify that the server responds with an error code 409.
  - c) Issue an HTTP DELETE request at URL {api\_root}/systems/{sysId}?cascade=true.
  - d) Verify that the server responds with an error code 409.

## ABSTRACT TEST A.69

**IDENTIFIER** /conf/create-replace-delete/subsystem

**REQUIREMENT** Requirement 62: /req/create-replace-delete/subsystem

## ABSTRACT TEST A.69

**TEST PURPOSE** Validate that the server implements the CREATE operation correctly on subsystem collections.

- 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) For each subsystem collection nested in a parent system:
      - Resources endpoint `{api_root}/systems/{sysId}/subsystems` (for CREATE)
  2. Verify that the subsystem is also available at its canonical URL:
    - a) Issue an HTTP GET request at the system canonical URL.
    - b) Validate that a document was returned with a status code 200.
    - c) Validate that the received document has the same content as the one provided for the CREATE operation.

## ABSTRACT TEST A.70

**IDENTIFIER** `/conf/create-replace-delete/deployment`

**REQUIREMENT** Requirement 63: `/req/create-replace-delete/deployment`

**TEST PURPOSE** Validate that the server implements CREATE/REPLACE/DELETE operations correctly on Deployment collections.

- 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}/deployments` (for CREATE)
    - b) At resource endpoint `{api_root}/deployments/{id}` (for REPLACE and DELETE)

## ABSTRACT TEST A.71

**IDENTIFIER** `/conf/create-replace-delete/subdeployment`

**REQUIREMENT** Requirement 64: `/req/create-replace-delete/subdeployment`

**TEST PURPOSE** Validate that the server implements the CREATE operation correctly on subdeployment collections.

- 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) For each subdeployment collection nested in a parent deployment:
      - Resources endpoint `{api_root}/deployments/{depId}/subdeployments` (for CREATE)
  2. Verify that the subdeployment is also available at its canonical URL:
    - a) Issue an HTTP GET request at the system canonical URL.
    - b) Validate that a document was returned with a status code 200.

## ABSTRACT TEST A.71

- c) Validate that the received document has the same content as the one provided for the CREATE operation.

## ABSTRACT TEST A.72

**IDENTIFIER** /conf/create-replace-delete/procedure

**REQUIREMENT** Requirement 65: /req/create-replace-delete/procedure

**TEST PURPOSE** Validate that the server implements CREATE/REPLACE/DELETE operations correctly on Procedure collections.

- 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}/procedures (for CREATE)
    - b) At resource endpoint {api\_root}/procedures/{id} (for REPLACE and DELETE)

## ABSTRACT TEST A.73

**IDENTIFIER** /conf/create-replace-delete/sampling-feature

**REQUIREMENT** Requirement 66: /req/create-replace-delete/sampling-feature

**TEST PURPOSE** Validate that the server implements CREATE/REPLACE/DELETE operations correctly on Sampling Feature collections.

- 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}/samplingFeatures (for CREATE)
    - b) At resource endpoint {api\_root}/systems/{sysId}/samplingFeatures/{id} (for REPLACE and DELETE)
    - c) At resource endpoint {api\_root}/samplingFeatures/{id} (for REPLACE and DELETE)

## ABSTRACT TEST A.74

**IDENTIFIER** /conf/create-replace-delete/property

**REQUIREMENT** Requirement 67: /req/create-replace-delete/property

**TEST PURPOSE** Validate that the server implements CREATE/REPLACE/DELETE operations correctly on Property collections.

## ABSTRACT TEST A.74

<b>TEST METHOD</b>	<ol style="list-style-type: none"><li>1. Execute all tests from conformance class <a href="http://www.opengis.net/spec/ogcapi-features-4/1.0/conf/create-replace-delete">http://www.opengis.net/spec/ogcapi-features-4/1.0/conf/create-replace-delete</a> at the following endpoints:<ol style="list-style-type: none"><li>a) At resources endpoint {api_root}/properties (for CREATE)</li><li>b) At resource endpoint {api_root}/properties/{id} (for REPLACE and DELETE)</li></ol></li></ol>
--------------------	--

## ABSTRACT TEST A.75

**IDENTIFIER** /conf/create-replace-delete/create-in-collection

**REQUIREMENT** Requirement 68: /req/create-replace-delete/create-in-collection

**TEST PURPOSE** Validate that the server implements the correct behavior when creating new resources in custom collections.

<b>TEST METHOD</b>	<p>For each resource type among System, Procedure, Deployment, Sampling Feature, Property:</p> <ol style="list-style-type: none"><li>1. Find a resource collection for a resource of that type. Assume its ID is colId.</li><li>2. Add a new resource at the resources endpoint {api_root}/collections/{colId}/items by following requirements for the CREATE operation (see tests <a href="http://www.opengis.net/spec/ogcapi-features-4/1.0/create-replace-delete/conf/post*">http://www.opengis.net/spec/ogcapi-features-4/1.0/create-replace-delete/conf/post*</a>).</li><li>3. Retrieve the canonical URL of the resource that must be included in the response.</li><li>4. Verify that the new resource exists at the canonical resources endpoint:<ol style="list-style-type: none"><li>a) Issue an HTTP GET request at the resource's canonical URL.</li><li>b) Validate that a document was returned with a status code 200.</li><li>c) Validate that the received document has the same content as the one provided in the POST request.</li></ol></li></ol>
--------------------	--

## ABSTRACT TEST A.76

**IDENTIFIER** /conf/create-replace-delete/replace-in-collection

**REQUIREMENT** Requirement 69: /req/create-replace-delete/replace-in-collection

**TEST PURPOSE** Validate that the server implements the correct behavior when replacing resources in custom collections.

<b>TEST METHOD</b>	<p>For each resource type among System, Procedure, Deployment, Sampling Feature, Property:</p> <ol style="list-style-type: none"><li>1. Find a resource collection for a resource of that type. Assume its ID is colId.</li><li>2. Replace the resource at the resource endpoint {api_root}/collections/{colId}/items/{id} by following requirements for the REPLACE operation (see tests <a href="http://www.opengis.net/spec/ogcapi-features-4/1.0/create-replace-delete/conf/put*">http://www.opengis.net/spec/ogcapi-features-4/1.0/create-replace-delete/conf/put*</a>).</li><li>3. Verify that the resource has been updated at its canonical location:</li></ol>
--------------------	---

## ABSTRACT TEST A.76

- a) Issue an HTTP GET request at the resource's canonical URL.
- b) Validate that a document was returned with a status code 200.
- c) Validate that the received document has the same content as the one provided in the PUT request.

## ABSTRACT TEST A.77

**IDENTIFIER** /conf/create-replace-delete/delete-in-collection

**REQUIREMENT** Requirement 70: /req/create-replace-delete/delete-in-collection

**TEST PURPOSE** Validate that the server implements the correct behavior when deleting resources in custom collections.

**TEST METHOD** For each resource type among System, Procedure, Deployment, Sampling Feature, Property:

1. Find a resource collection for a resource of that type. Assume its ID is colId.
2. Delete the resource at the resource endpoint {api\_root}/collections/{colId}/items/{id} by following requirements for the DELETE operation (see tests [http://www.opengis.net/spec/ogcapi-features-4/1.0/create-replace-delete/conf/delete\\*](http://www.opengis.net/spec/ogcapi-features-4/1.0/create-replace-delete/conf/delete*)).
3. Verify that the resource is still available at its canonical location:
  - a) Issue an HTTP GET request at the resource's canonical URL.
  - b) Validate that a document was returned with a status code 200.

## ABSTRACT TEST A.78

**IDENTIFIER** /conf/create-replace-delete/add-to-collection

**REQUIREMENT** Requirement 71: /req/create-replace-delete/add-to-collection

**TEST PURPOSE** Validate that the server implements the correct behavior when adding existing resources to custom collections.

**TEST METHOD** For each resource type among System, Procedure, Deployment, Sampling Feature, Property:

1. Find a resource collection for a resource of that type. Assume its ID is colId.
2. Add links to existing resources by issuing a POST request at the resources endpoint {api\_root}/collections/{colId}/items.
  - a) Follow requirements for the CREATE operation (see tests [http://www.opengis.net/spec/ogcapi-features-4/1.0/create-replace-delete/conf/post\\*](http://www.opengis.net/spec/ogcapi-features-4/1.0/create-replace-delete/conf/post*)).
  - b) Set the POST request Content-Type header to text/uri-list.

## ABSTRACT TEST A.78

- c) Set the POST request body to a list of canonical URLs of existing resources on the same server, and of a type compatible with the selected resource collection.
3. Verify that the resources have been added to the custom collection:
  - a) For each added resource, extract its `id` from the canonical URL.
    1. Issue an HTTP GET request at the resource endpoint `{api_root}/collections/{colId}/items/{id}`.
    2. Validate that a document was returned with a status code 200.
    3. Validate that the received document has the same content as the one received when connecting at the canonical URL.

## A.12. Conformance Class “Update”

### CONFORMANCE CLASS A.11

IDENTIFIER	/conf/update
REQUIREMENTS CLASS	Requirements class 11: /req/update
PREREQUISITES	Conformance class A.1: /conf/api-common <a href="http://www.opengis.net/spec/ogcapi-4/1.0/conf/update">http://www.opengis.net/spec/ogcapi-4/1.0/conf/update</a>
TARGET TYPE	Web API
CONFORMANCE TESTS	Abstract test A.79: /conf/update/system Abstract test A.80: /conf/update/deployment Abstract test A.81: /conf/update/procedure Abstract test A.82: /conf/update/sampling-feature Abstract test A.83: /conf/update/property

### ABSTRACT TEST A.79

IDENTIFIER	/conf/update/system
REQUIREMENT	Requirement 72: /req/update/system
TEST PURPOSE	Validate that the server implements the UPDATE operation correctly on System collections.
TEST METHOD	<ol style="list-style-type: none"><li>1. Execute all tests from conformance class <a href="http://www.opengis.net/spec/ogcapi-features-4/1.0/conf/update">http://www.opengis.net/spec/ogcapi-features-4/1.0/conf/update</a> at the following endpoints:<ol style="list-style-type: none"><li>a) For the System canonical resources endpoint:</li></ol></li></ol>

## ABSTRACT TEST A.79

- Resource endpoint {api\_root}/systems/{id}
- b) For each System Feature Collection advertised by the server:
  - Resource endpoint {api\_root}/collections/{systemCollectionId}/items/{id}

## ABSTRACT TEST A.80

**IDENTIFIER** /conf/update/deployment

**REQUIREMENT** Requirement 73: /req/update/deployment

**TEST PURPOSE** Validate that the server implements the UPDATE operation correctly on Deployment collections.

- 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) For the Deployment canonical resources endpoint:
      - Resource endpoint {api\_root}/deployments/{id}
    - b) For each Deployment Feature Collection advertised by the server:
      - Resource endpoint {api\_root}/collections/{deploymentCollectionId}/items/{id}

## ABSTRACT TEST A.81

**IDENTIFIER** /conf/update/procedure

**REQUIREMENT** Requirement 74: /req/update/procedure

**TEST PURPOSE** Validate that the server implements the UPDATE operation correctly on Procedure collections.

- 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) For the Procedure canonical resources endpoint:
      - Resource endpoint {api\_root}/procedures/{id}
    - b) For each Procedure Feature Collection advertised by the server:
      - Resource endpoint {api\_root}/collections/{procedureCollectionId}/items/{id}

## ABSTRACT TEST A.82

**IDENTIFIER** /conf/update/sampling-feature

## ABSTRACT TEST A.82

**REQUIREMENT** Requirement 75: /req/update/sampling-feature

**TEST PURPOSE** Validate that the server implements the UPDATE operation correctly on Sampling Feature collections.

**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) For the Sampling Feature canonical resources endpoint:
    - Resource endpoint {api\_root}/samplingFeatures/{id}
  - b) For each Sampling Feature Collection advertised by the server:
    - Resource endpoint {api\_root}/collections/{sfCollectionId}/items/{id}

## ABSTRACT TEST A.83

**IDENTIFIER** /conf/update/property

**REQUIREMENT** Requirement 76: /req/update/property

**TEST PURPOSE** Validate that the server implements the UPDATE operation correctly on Property collections.

**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) For the Property canonical resources endpoint:
    - Resource endpoint {api\_root}/properties/{id}
  - b) For each Property Resource Collection advertised by the server:
    - Resource endpoint {api\_root}/collections/{sfCollectionId}/items/{id}

## A.13. Conformance Class “GeoJSON”

---

### CONFORMANCE CLASS A.12

**IDENTIFIER** /conf/geojson

**REQUIREMENTS CLASS** Requirements class 12: /req/geojson

**PREREQUISITES** Conformance class A.1: /conf/api-common  
<http://www.opengis.net/spec/ogcapi-1/1.0/conf/geojson>

**TARGET TYPE** Web API

## CONFORMANCE CLASS A.12

### CONFORMANCE TESTS

Abstract test A.84: /conf/geojson/mediatype-read  
Abstract test A.85: /conf/geojson/mediatype-write  
Abstract test A.86: /conf/geojson/relation-types  
Abstract test A.87: /conf/geojson/feature-attribute-mapping  
Abstract test A.88: /conf/geojson/system-schema  
Abstract test A.89: /conf/geojson/system-mappings  
Abstract test A.90: /conf/geojson/deployment-schema  
Abstract test A.91: /conf/geojson/deployment-mappings  
Abstract test A.92: /conf/geojson/procedure-schema  
Abstract test A.93: /conf/geojson/procedure-mappings  
Abstract test A.94: /conf/geojson/sf-schema  
Abstract test A.95: /conf/geojson/sf-mappings

## ABSTRACT TEST A.84

**IDENTIFIER** /conf/geojson/mediatype-read

**REQUIREMENT** Requirement 77: /req/geojson/mediatype-read

**TEST PURPOSE** Verify that the server advertises support for the GeoJSON format on retrieval operations.

### TEST METHOD

1. Execute test <http://www.opengis.net/spec/ogcapi-features-1/1.0/conf/geojson/definition>.
2. Verify that server advertises support for media type application/geo+json in the API definition for GET operations:
  - a) On the canonical resources endpoints of resource types supported by the server.
  - b) On the custom collection endpoints advertised by the server.

## ABSTRACT TEST A.85

**IDENTIFIER** /conf/geojson/mediatype-write

**REQUIREMENT** Requirement 78: /req/geojson/mediatype-write

**TEST PURPOSE** Verify that the server advertises support for the GeoJSON format on transactional operations.

### TEST METHOD

1. Verify that server advertises support for media type application/geo+json in the API definition for CREATE or REPLACE operations, for at least one canonical resources endpoint.

## ABSTRACT TEST A.86

IDENTIFIER	/conf/geojson/relation-types
REQUIREMENT	Requirement 79: /req/geojson/relation-types
TEST PURPOSE	Verify that correct link relation types are used.
TEST-METHOD-TYPE	Manual Inspection
TEST METHOD	Given the GeoJSON representation of a resource returned by the server: <ol style="list-style-type: none"><li>1. Inspect the links in the <code>links</code> property of the response document.</li><li>2. Check that the relation types are used as described in the associations mapping table of Clause 19.1 corresponding to the resource type.</li></ol>

## ABSTRACT TEST A.87

IDENTIFIER	/conf/geojson/feature-attribute-mapping
REQUIREMENT	Requirement 80: /req/geojson/feature-attribute-mapping
TEST PURPOSE	Verify that common feature properties are used correctly.
TEST-METHOD-TYPE	Manual Inspection
TEST METHOD	Given the GeoJSON representation of a feature resource returned by the server: <ol style="list-style-type: none"><li>1. Inspect the contents of the GeoJSON Feature object.</li><li>2. Check that the properties are used as described in the mapping table Table 20.</li></ol>

## ABSTRACT TEST A.88

IDENTIFIER	/conf/geojson/system-schema
REQUIREMENT	Requirement 81: /req/geojson/system-schema
TEST PURPOSE	Validate that the GeoJSON representation of System resources is valid.
TEST METHOD	<ol style="list-style-type: none"><li>1. Request a single System resource.<ol style="list-style-type: none"><li>a) Issue an HTTP GET request at <code>{api_root}/systems/{id}</code> with the Accept header set to <code>application/geo+json</code>.</li><li>b) Validate that a document was returned with a status code 200.</li><li>c) Validate the document against the schema <a href="#">system.json</a> using a JSON Schema validator.</li></ol></li><li>2. Request multiple System resources.</li></ol>

## ABSTRACT TEST A.88

- a) Issue an HTTP GET request at `{api_root}/systems` with the Accept header set to `application/geo+json`.
- b) Validate that a document was returned with a status code 200.
- c) Validate the document against the schema `systemCollection.json` using a JSON Schema validator.

## ABSTRACT TEST A.89

**IDENTIFIER** `/conf/geojson/system-mappings`

**REQUIREMENT** Requirement 82: `/req/geojson/system-mappings`

**TEST PURPOSE** Verify that System properties are used correctly.

**TEST-METHOD-TYPE** Manual Inspection

**TEST METHOD** Given the GeoJSON representation of a System resource returned by the server:

1. Inspect the contents of the GeoJSON Feature object.
2. Check that the properties are used as described in Table 21 and Table 22.

## ABSTRACT TEST A.90

**IDENTIFIER** `/conf/geojson/deployment-schema`

**REQUIREMENT** Requirement 83: `/req/geojson/deployment-schema`

**TEST PURPOSE** Validate that the GeoJSON representation of Deployment resources is valid.

**TEST METHOD**

1. Request a single Deployment resource.
  - a) Issue an HTTP GET request at `{api_root}/deployments/{id}` with the Accept header set to `application/geo+json`.
  - b) Validate that a document was returned with a status code 200.
  - c) Validate the document against the schema `deployment.json` using a JSON Schema validator.
2. Request multiple Deployment resources.
  - a) Issue an HTTP GET request at `{api_root}/deployments` with the Accept header set to `application/geo+json`.
  - b) Validate that a document was returned with a status code 200.
  - c) Validate the document against the schema `deploymentCollection.json` using a JSON Schema validator.

## ABSTRACT TEST A.91

**IDENTIFIER** /conf/geojson/deployment-mappings

**REQUIREMENT** Requirement 84: /req/geojson/deployment-mappings

**TEST PURPOSE** Verify that Deployment properties are used correctly.

**TEST-METHOD-TYPE** Manual Inspection

**TEST METHOD** Given the GeoJSON representation of a Deployment resource returned by the server:

1. Inspect the contents of the GeoJSON Feature object.
2. Check that the properties are used as described in Table 23 and Table 24.

## ABSTRACT TEST A.92

**IDENTIFIER** /conf/geojson/procedure-schema

**REQUIREMENT** Requirement 85: /req/geojson/procedure-schema

**TEST PURPOSE** Validate that the GeoJSON representation of Procedure resources is valid.

**TEST METHOD**

1. Request a single Procedure resource.
  - a) Issue an HTTP GET request at {api\_root}/procedures/{id} with the Accept header set to application/geo+json.
  - b) Validate that a document was returned with a status code 200.
  - c) Validate the document against the schema [procedure.json](#) using a JSON Schema validator.
2. Request multiple Procedure resources.
  - a) Issue an HTTP GET request at {api\_root}/procedures with the Accept header set to application/geo+json.
  - b) Validate that a document was returned with a status code 200.
  - c) Validate the document against the schema [procedureCollection.json](#) using a JSON Schema validator.

## ABSTRACT TEST A.93

**IDENTIFIER** /conf/geojson/procedure-mappings

**REQUIREMENT** Requirement 86: /req/geojson/procedure-mappings

## ABSTRACT TEST A.93

**TEST PURPOSE** Verify that Procedure properties are used correctly.

**TEST-METHOD-TYPE** Manual Inspection

**TEST METHOD** Given the GeoJSON representation of a Procedure resource returned by the server:

1. Inspect the contents of the GeoJSON Feature object.
2. Check that the properties are used as described in Table 25 and Table 26.

## ABSTRACT TEST A.94

**IDENTIFIER** /conf/geojson/sf-schema

**REQUIREMENT** Requirement 87: /req/geojson/sf-schema

**TEST PURPOSE** Validate that the GeoJSON representation of Sampling Feature resources is valid.

**TEST METHOD**

1. Request a single Sampling Feature resource.
  - a) Issue an HTTP GET request at {api\_root}/samplingFeatures/{id} with the Accept header set to application/geo+json.
  - b) Validate that a document was returned with a status code 200.
  - c) Validate the document against the schema [anySamplingFeature.json](#) using a JSON Schema validator.
2. Request multiple Sampling Feature resources.
  - a) Issue an HTTP GET request at {api\_root}/samplingFeatures with the Accept header set to application/geo+json.
  - b) Validate that a document was returned with a status code 200.
  - c) Validate the document against the schema [samplingFeatureCollection.json](#) using a JSON Schema validator.

## ABSTRACT TEST A.95

**IDENTIFIER** /conf/geojson/sf-mappings

**REQUIREMENT** Requirement 88: /req/geojson/sf-mappings

**TEST PURPOSE** Verify that Sampling Feature properties are used correctly.

**TEST-METHOD-TYPE** Manual Inspection

**TEST METHOD** Given the GeoJSON representation of a Sampling Feature resource returned by the server:

## ABSTRACT TEST A.95

1. Inspect the contents of the GeoJSON Feature object.
2. Check that the properties are used as described in Table 27 and Table 28.

## A.14. Conformance Class “SensorML”

### CONFORMANCE CLASS A.13

**IDENTIFIER** /conf/sensorml

**REQUIREMENTS CLASS** Requirements class 13: /req/sensorml

**PREREQUISITE** Conformance class A.1: /conf/api-common

**INDIRECT PREREQUISITE** <http://www.opengis.net/spec/sensorml/2.1/conf/json>

**TARGET TYPE** Web API

**CONFORMANCE TESTS**

Abstract test A.96: /conf/sensorml/mediatype-read  
Abstract test A.97: /conf/sensorml/mediatype-write  
Abstract test A.98: /conf/sensorml/relation-types  
Abstract test A.99: /conf/sensorml/resource-id  
Abstract test A.100: /conf/sensorml/feature-attribute-mapping  
Abstract test A.101: /conf/sensorml/system-schema  
Abstract test A.102: /conf/sensorml/system-sml-class  
Abstract test A.103: /conf/sensorml/system-mappings  
Abstract test A.104: /conf/sensorml/deployment-schema  
Abstract test A.105: /conf/sensorml/deployment-mappings  
Abstract test A.106: /conf/sensorml/procedure-schema  
Abstract test A.107: /conf/sensorml/procedure-sml-class  
Abstract test A.108: /conf/sensorml/procedure-mappings  
Abstract test A.109: /conf/sensorml/property-schema  
Abstract test A.110: /conf/sensorml/property-mappings

### ABSTRACT TEST A.96

**IDENTIFIER** /conf/sensorml/mediatype-read

**REQUIREMENT** Requirement 89: /req/sensorml/mediatype-read

## ABSTRACT TEST A.96

**TEST PURPOSE** Verify that the server advertises support for the SensorML format on retrieval operations.

**TEST METHOD**

1. Verify that server advertises support for media type `application/sml+json` in the API definition for GET operations:
  - a) On the canonical resources endpoints of resource types supported by the server.
  - b) On the custom collection endpoints advertised by the server.

## ABSTRACT TEST A.97

**IDENTIFIER** `/conf/sensorml/mediatype-write`

**REQUIREMENT** Requirement 90: `/req/sensorml/mediatype-write`

**TEST PURPOSE**

Verify that the server advertises support for the SensorML format on transactional operations.

1. Verify that server advertises support for media type `application/sml+json` in the API definition for CREATE or REPLACE operations, for at least one canonical resources endpoint.

## ABSTRACT TEST A.98

**IDENTIFIER** `/conf/sensorml/relation-types`

**REQUIREMENT** Requirement 91: `/req/sensorml/relation-types`

**TEST PURPOSE** Verify that correct link relation types are used.

**TEST-METHOD-TYPE** Manual Inspection

**TEST METHOD**

Given the SensorML representation of a resource returned by the server:

1. Inspect the links in the `links` property of the response document.
2. Check that the relation types are used as described in the associations mapping table of Clause 19.2 corresponding to the resource type.

## ABSTRACT TEST A.99

**IDENTIFIER** `/conf/sensorml/resource-id`

**REQUIREMENT** Requirement 92: `/req/sensorml/resource-id`

**TEST PURPOSE** Verify that the resource ID is set properly in the response.

## ABSTRACT TEST A.99

<b>TEST METHOD</b>	Given the SensorML representation of a resource obtained from its canonical URL: <ol style="list-style-type: none"><li>1. Inspect the contents of the SensorML object.</li><li>2. Verify that the <code>id</code> property is set to the same value as the <code>{id}</code> portion of the canonical resource URL.</li></ol>
--------------------	---

## ABSTRACT TEST A.100

<b>IDENTIFIER</b>	<code>/conf/sensorml/feature-attribute-mapping</code>
<b>REQUIREMENT</b>	Requirement 93: <code>/req/sensorml/feature-attribute-mapping</code>
<b>TEST PURPOSE</b>	Verify that common feature properties are used correctly.
<b>TEST-METHOD-TYPE</b>	Manual Inspection
<b>TEST METHOD</b>	Given the SensorML representation of a resource returned by the server: <ol style="list-style-type: none"><li>1. Inspect the contents of the SensorML object.</li><li>2. Check that the properties are used as described in the mapping table Table 29.</li></ol>

## ABSTRACT TEST A.101

<b>IDENTIFIER</b>	<code>/conf/sensorml/system-schema</code>
<b>REQUIREMENT</b>	Requirement 94: <code>/req/sensorml/system-schema</code>
<b>TEST PURPOSE</b>	Validate that the SensorML representation of System resources is valid.
<b>TEST METHOD</b>	<ol style="list-style-type: none"><li>1. Request a single System resource.<ol style="list-style-type: none"><li>a) Issue an HTTP GET request at <code>{api_root}/systems/{id}</code> with the Accept header set to <code>application/sml+json</code>.</li><li>b) Validate that a document was returned with a status code 200.</li><li>c) Validate the document against the schema <code>system.json</code> using a JSON Schema validator.</li></ol></li><li>2. Request multiple System resources.<ol style="list-style-type: none"><li>a) Issue an HTTP GET request at <code>{api_root}/systems</code> with the Accept header set to <code>application/sml+json</code>.</li><li>b) Validate that a document was returned with a status code 200.</li><li>c) Validate the document against the schema <code>systemCollection.json</code> using a JSON Schema validator.</li></ol></li></ol>

## ABSTRACT TEST A.102

**IDENTIFIER** /conf/sensorml/system-sml-class

**REQUIREMENT** Requirement 95: /req/sensorml/system-sml-class

**TEST PURPOSE** Verify that System SensorML types are used correctly.

**TEST-METHOD-TYPE** Manual Inspection

**TEST METHOD** Given the SensorML representation of a System resource returned by the server:

1. Inspect the contents of the SensorML object.
2. Check that the value of the type property is compatible with the system being described (i.e. process/simulation vs. physical thing).

## ABSTRACT TEST A.103

**IDENTIFIER** /conf/sensorml/system-mappings

**REQUIREMENT** Requirement 96: /req/sensorml/system-mappings

**TEST PURPOSE** Verify that System properties are used correctly.

**TEST-METHOD-TYPE** Manual Inspection

**TEST METHOD** Given the SensorML representation of a System resource returned by the server:

1. Inspect the contents of the SensorML object.
2. Check that the properties are used as described in Table 30 and Table 31.

## ABSTRACT TEST A.104

**IDENTIFIER** /conf/sensorml/deployment-schema

**REQUIREMENT** Requirement 97: /req/sensorml/deployment-schema

**TEST PURPOSE** Validate that the SensorML representation of Deployment resources is valid.

**TEST METHOD**

1. Request a single Deployment resource.
  - a) Issue an HTTP GET request at {api\_root}/deployments/{id} with the Accept header set to application/sml+json.
  - b) Validate that a document was returned with a status code 200.
  - c) Validate the document against the schema [deployment.json](#) using a JSON Schema validator.

## ABSTRACT TEST A.104

2. Request multiple Deployment resources.
  - a) Issue an HTTP GET request at `{api_root}/deployments` with the Accept header set to `application/sml+json`.
  - b) Validate that a document was returned with a status code 200.
  - c) Validate the document against the schema [deploymentCollection.json](#) using a JSON Schema validator.

## ABSTRACT TEST A.105

**IDENTIFIER** /conf/sensorml/deployment-mappings

**REQUIREMENT** Requirement 98: /req/sensorml/deployment-mappings

**TEST PURPOSE** Verify that Deployment properties are used correctly.

**TEST-METHOD-TYPE** Manual Inspection

**TEST METHOD** Given the SensorML representation of a Deployment resource returned by the server:

1. Inspect the contents of the SensorML object.
2. Check that the properties are used as described in Table 32 and Table 33.

## ABSTRACT TEST A.106

**IDENTIFIER** /conf/sensorml/procedure-schema

**REQUIREMENT** Requirement 99: /req/sensorml/procedure-schema

**TEST PURPOSE** Validate that the SensorML representation of Procedure resources is valid.

**TEST METHOD**

1. Request a single Procedure resource.
  - a) Issue an HTTP GET request at `{api_root}/procedures/{id}` with the Accept header set to `application/sml+json`.
  - b) Validate that a document was returned with a status code 200.
  - c) Validate the document against the schema [procedure.json](#) using a JSON Schema validator.
2. Request multiple Procedure resources.
  - a) Issue an HTTP GET request at `{api_root}/procedures` with the Accept header set to `application/sml+json`.
  - b) Validate that a document was returned with a status code 200.
  - c) Validate the document against the schema [procedureCollection.json](#) using a JSON Schema validator.

## ABSTRACT TEST A.107

IDENTIFIER	/conf/sensorml/procedure-sml-class
REQUIREMENT	Requirement 100: /req/sensorml/procedure-sml-class
TEST PURPOSE	Verify that Procedure SensorML types are used correctly.
TEST-METHOD-TYPE	Manual Inspection
TEST METHOD	Given the SensorML representation of a Procedure resource returned by the server: <ol style="list-style-type: none"><li>1. Inspect the contents of the SensorML object.</li><li>2. Check that the value of the type property is compatible with the procedure being described (i.e. process/simulation/methodology vs. datasheet of hardware equipment).</li></ol>

## ABSTRACT TEST A.108

IDENTIFIER	/conf/sensorml/procedure-mappings
REQUIREMENT	Requirement 101: /req/sensorml/procedure-mappings
TEST PURPOSE	Verify that Procedure properties are used correctly.
TEST-METHOD-TYPE	Manual Inspection
TEST METHOD	Given the SensorML representation of a Procedure resource returned by the server: <ol style="list-style-type: none"><li>1. Inspect the contents of the SensorML object.</li><li>2. Check that the properties are used as described in Table 34 and Table 35.</li></ol>

## ABSTRACT TEST A.109

IDENTIFIER	/conf/sensorml/property-schema
REQUIREMENT	Requirement 102: /req/sensorml/property-schema
TEST PURPOSE	Validate that the SensorML representation of Property resources is valid.
TEST METHOD	<ol style="list-style-type: none"><li>1. Request a single Property resource.<ol style="list-style-type: none"><li>a) Issue an HTTP GET request at {api_root}/properties/{id} with the Accept header set to application/sml+json.</li><li>b) Validate that a document was returned with a status code 200.</li></ol></li></ol>

## ABSTRACT TEST A.109

- c) Validate the document against the schema [property.json](#) using a JSON Schema validator.
2. Request multiple Property resources.
    - a) Issue an HTTP GET request at `{api_root}/properties` with the Accept header set to `application/sml+json`.
    - b) Validate that a document was returned with a status code 200.
    - c) Validate the document against the schema [propertyCollection.json](#) using a JSON Schema validator.

## ABSTRACT TEST A.110

IDENTIFIER	/conf/sensorml/property-mappings
REQUIREMENT	Requirement 103: /req/sensorml/property-mappings
TEST PURPOSE	Verify that Property properties are used correctly.
TEST-METHOD-TYPE	Manual Inspection
TEST METHOD	Given the SensorML representation of a Property resource returned by the server: <ol style="list-style-type: none"><li>1. Inspect the contents of the SensorML object.</li><li>2. Check that the properties are used as described in Table 36.</li></ol>



# ANNEX B (INFORMATIVE) EXAMPLES

---



## ANNEX B (INFORMATIVE) EXAMPLES

---

More JSON examples are available in the project's GitHub repository at:

<https://github.com/opengeospatial/ogcapi-connected-systems/tree/master/api/part1/openapi/examples>



# ANNEX C (INFORMATIVE) RELATIONSHIP WITH OTHER OGC/ISO STANDARDS (INFORMATIVE)

---



# ANNEX C

## (INFORMATIVE)

### RELATIONSHIP WITH OTHER OGC/ISO STANDARDS (INFORMATIVE)

---

#### C.1. W3C Semantic Sensor Network Ontology

---

The W3C Semantic Sensor Network Ontology (SOSA/SSN) Recommendation provides the semantic model from which the resource model of the OGC API – Connected Systems Standard (CS API) is derived.

The CS API does not use any SOSA/SSN serialization (e.g. RDF, Turtle, etc.) directly as a resource representation format. Instead, GeoJSON, SensorML-JSON or in some cases plain JSON and even binary representations are used, with references to SOSA/SSN concepts (by URI or CURIE) whenever appropriate (e.g. System Types, Procedure Types, etc.).

##### **Divergence from the SOSA/SSN Patterns**

The CS API resource model diverges from currently used SOSA/SSN patterns in the following ways:

1. In the CS API, all systems can have both `datastreams` and `controlstreams`, even though they are tagged with only one type (i.e. `sosa:Sensor`, `sosa:Actuator`, `sosa:Sampler` or `sosa:Platform`). Semantic tagging is only used to provide the main role of the system or subsystem without limiting other API functionality. This design simplifies the description of sensors that can also accept commands (e.g. even simple sensors often have commands to change the sampling rate or the sensitivity), or actuators that output data (e.g. most actuators provide some kind of state information). The same effect can be achieved when using the SOSA/SSN ontology by creating RDF resources that have several parent classes such as:  

```
<Motor1> a sosa:System, sosa:Sensor, sosa:Actuator ; ...  
<UAV1> a sosa:System, sosa:Platform ; ...
```
2. In the CS API, an observation can be associated to more than one observed property, and a command to more than one controlled property, while SOSA/SSN has a cardinality of one on these associations. This also means that observations are not limited to scalar results, and commands are not limited to scalar parameters. Discussions in the SOSA/SSN working group have shown that

such vectorization of observed properties and corresponding observation results is an accepted practice and will be clearly documented in the next revision of the SOSA/SSN standard.

3. We use the concepts of `ObservationCollection` and `ActuationCollection` (in the form of `DataStream` and `ControlStream` respectively) to factor out many properties that are common to observations/actuators associated to the same sensor/actuator. Note that such collections are not part of the current version of SOSA/SSN at the time of writing but will be incorporated in the next revision.

These patterns are used in the CS API to reduce the number (and sometimes the size) of resources that one needs to create and maintain in order to fully describe complex systems and their data feeds. Where a few resources are necessary using the CS API, the SOSA/SSN model would require many more small resources to achieve the same functionality, which leads to complexity for API consumers.

Rather than imposing a very verbose schema, these changes give more flexibility for choosing the appropriate level of granularity used when describing hierarchical systems, while still allowing the API implementation to exchange all required dynamic data feeds. Granularity can range from creating a different sensor or actuator for every single scalar property observed or acted on, all the way to a single resource that is used to represent an entire complex system as a “black box”.

## C.2. OGC Sensor Modeling Language (SensorML) Standard

---

OGC Sensor Model Language (SensorML) is one of the foundational building blocks of the OGC API – Connected Systems Standard (CS API).

Many of the foundational ideas for the CS API Standard are inherited from SensorML, although original models have been converted to a resource oriented approach and aligned with SOSA/SSN in the process of creating the API.

The CS API Standard specifies how to use SensorML JSON encodings as a resource encoding format, but the XML version of SensorML can also be used as a response format by the API server.

Currently, SensorML is the only encoding format supporting the provision of detailed specsheets for systems, procedures, and deployments.

## C.3. OGC/ISO Observations, Measurements and Samples (OMS) Standard

The Observations, Measurements and Samples (OMS) Standard is one of the foundational building blocks of the OGC API – Connected Systems Standard, through the use of the Semantic Sensor Network Ontology (SOSA/SSN). Indeed, SOSA/SSN was originally based on a previous version of OMS, Observations and Measurements (O&M), so many concepts from OMS are naturally carried over to the CS API Standard.

The following table indicates which classes of OMS Conceptual Observation Schema and Conceptual Sample Schema are implemented by OGC API – Connected Systems (CS API):

OMS CLASS	O&M CLASS	CS API RESOURCE	COMMENTS
Observation	OM_ Observation	Observation	
ObservableProperty	GFI_ PropertyType	Property	
Procedure	OM_Process	Procedure (type=sosa: Procedure)	
ObservingProcedure	OM_Process	Procedure (type=sosa-oms: ObservingProcedure)	
Observer	OM_Process	System (type=sosa:Sensor)	
Host	-	System (type=sosa:Platform)	
Deployment	-	Deployment	
Sample	SF_ SamplingFeature	SamplingFeature	
SamplingProcedure	-	Procedure (type=sosa-oms: SamplingProcedure)	
Sampler	-	System (type=sosa:Sampler)	

## C.4. IETF GeoJSON

---

GeoJSON is one of the main encoding formats specified in the OGC API – Connected Systems Standard. A GeoJSON profile for each feature type is provided in Clause 19.1, with complete JSON schemas and mappings to the abstract resource model.

Since SensorML is available for cases where detailed information about systems, procedures and deployments is needed (e.g. contact information, detailed system datasheets, etc.), the GeoJSON profile is intentionally kept simple and is intended to be used as a summary representation.

Although not mandated by the Standard, a typical workflow would be to retrieve collections of feature descriptions (e.g. as the result of search) by requesting the GeoJSON representation first, since the GeoJSON encoding leads to much smaller documents than the SensorML-JSON encoding. Then, after reviewing the summary descriptions provided as GeoJSON, the full SensorML description could be fetched (typically only for some of the features of the collection).

## C.5. OGC Features and Geometries JSON (JSON-FG)

---

Similarly to GeoJSON, a profile of JSON-FG could be defined to provide an alternative representation of feature types defined in the CS API Standard. The JSON-FG requirements class has been left out of Part 1 but will be considered at a later stage once more implementation experience has been collected.

## C.6. OGC API – Features Standard

---

The OGC API – Connected Systems Standard is an extension of the OGC API – Features Standard, and thus inherits all requirements from OGC API – Features – Part 1: Core. In addition, requirements from OGC API – Features – Part 4: Create, Replace, Update and Delete are inherited by Requirements Class “Create/Replace/Delete” and Requirements Class “Update”. Requirements from OGC API – Features – Part 2: Coordinate Reference Systems by Reference also apply if coordinate reference systems other than CRS:84 or CRS:84h are to be used.

Additionally, some resources defined by the CS API Standard are designed to link to feature resources hosted by other servers (e.g. domain features) that act as features of interest of observations (e.g. rivers, roads, buildings, etc.). Such domain features can typically be hosted by separate implementation instances of OGC API – Features. This linking capability allows OGC API – Connected Systems implementation instances to refer to existing feature repositories such as land registers, building databases, transportation networks, hydrographic networks, oceanic features, etc, without duplicating the data.

## C.7. OGC API – Moving Features Standard

---

The OGC API – Moving Features Standard defines an alternative way to provide dynamic feature property data, but is more limited in scope. The key difference between OGC API – Moving Features and OGC API – Connected Systems is that the former does not implement the O&M or SSN models, and thus does not support the provision of metadata about the observation process, nor does it support tasking capabilities.

If needed, both APIs can be implemented at the same endpoint (since both extend OGC API – Features) to provide two complementary viewpoints of the same underlying dynamic data.

## C.8. OGC API – Environmental Data Retrieval (EDR) Standard

---

The OGC API – Environmental Data Retrieval (EDR) Standard can also be used to retrieve observation data. EDR is especially suited for extracting data from large multi-dimensional coverages and can be used jointly with the OGC API – Connected Systems Standard.

Weblinks can be used to associate resources exposed by OGC API – EDR and OGC API – Connected Systems (CS API). Such links can be used to implement the following client functionality:

- An EDR API client can retrieve more information about the observing system that produced the data (i.e. the data in an EDR collection or instance) from the CS API.
- Conversely, a Connected Systems API client can be redirected to an EDR accessible collection or instance in order to benefit from the advanced query operators defined in the EDR Standard (e.g. radius, cube, trajectory, corridor, etc.), and thus extract data from large coverage results more efficiently.

To this effect, the following weblinks can be added to OGC API – EDR resources to refer to OGC API – Connected Systems (CS API) resources:

EDR RESOURCE	TARGET CS API RESOURCES	COMMENTS
Collection Metadata	System Deployment DataStream	
Instance Metadata	System Deployment DataStream	

And the following weblinks can be added to OGC API – Connected Systems resources to refer to OGC API – EDR resources:

CS API RESOURCE	TARGET EDR RESOURCES	COMMENTS
System	Collection Instance	
DataStream	Collection Instance	

## C.9. OGC SensorThings API Standard

The SensorThings API (STA) is another OGC Standard designed to provide access to sensor observations and tasking through a REST API.

Although the two API Standards are in some ways similar, the SensorThings API was designed to solve IoT use cases and does not address the need of all sensor systems. OGC API – Connected Systems takes a more generic approach to the problem by extending OGC API – Features and using SOSA/SSN and SensorML as the main conceptual and implementation models behind the API.

The following table compares the design choices made in OGC API – Connected Systems and SensorThings API:

DESIGN CHOICE	CONNECTED SYSTEMS	SENSORTHINGS
API Platform	Extension of OGC API Common and OGC – API Features.	OData Version 4.0
Query Language	Query string arguments, decoupled from resource encoding.	Generic query language inherited from OData.
Resource Model	Based on SOSA/SSN/OMS and SensorML.	Simplified and adapted from O&M.
Supported Observation Types	Scalar, vector, N-D coverage, video.	Scalar and simple records only.
Multiple Format Support	Yes, including non-JSON such as <a href="#">Protocol Buffers</a> or other binary formats.	OData compatible JSON only.

The next table shows a comparison of SensorThings and OGC API – Connected Systems (CS API) resources:

STA RESOURCE	CS API RESOURCE	COMMENTS
Thing	System	type = sosa:Platform
Location	Observation	Location is implemented as a specific kind of observation whose result is a location vector.
HistoricalLocation	DataStream	Historical locations are implemented as a DataStream containing location observations (see above).
Datastream	DataStream	
Sensor	System	type = sosa:Sensor
ObservedProperty	Property	
Observation	Observation	
FeatureOfInterest	SamplingFeature	The sampling feature is a proxy to any other feature resource.
Actuator	System	type = sosa:Actuator
TaskingCapability	CommandStream	
Task	Command	
-	Procedure	
-	Deployment	

If needed, the following weblinks can be added to OGC API – Connected Systems resources to refer to SensorThings API resources:

CS API RESOURCE	TARGET STA RESOURCES	COMMENTS
System	Thing Sensor Actuator	
DataStream	Datastream	
ControlStream	TaskingCapability	

## C.10. Coverages

---

Observation results are sometimes coverages (e.g. satellite imagery, weather forecast, etc.). In the case of large coverages, providing access to the observation result is better handled by APIs that allow subsetting the coverage along its various dimensions.

Instead of duplicating existing functionality, OGC API – Connected Systems supports linking to coverage datasets hosted by other API implementations or web services when appropriate, instead of including the coverage result data inline in the observation.

In particular, links to implementation instances of the following OGC services and APIs are possible:

- OGC API – Coverages
- OGC API – Maps
- OGC API – EDR
- OGC Web Coverage Service
- OGC Web Map Service

The exact mechanism for linking Connected Systems resources and Coverage datasets is implemented will be specified in a future OGC Best Practice document.

## C.11. 3D Features

---

The following OGC Standards can be used to represent and/or transfer complex 3D content and/or scenes:

- OGC CityGML Standard
- OGC CityJSON Community Standard
- OGC 3D Tiles Community Standard
- OGC Indexed 3d Scene Layer (I3S) Community Standard
- OGC API – 3D GeoVolumes (Draft)

Such 3D scenes contain feature objects (i.e. features of interest) that can be the target of observations or commands (e.g. a building feature in the 3D model of a city, a mechanical part in the 3D model of an engine, etc.).

These features of interest can be referenced by OGC API – Connected Systems resources, enabling clients to associate the observations to the exact object in the 3D scene (e.g. the user could click an object in the scene and be presented with a chart or a list of dynamic data stream about this object). The reverse link going from the 3D model to the Connected Systems datastream is also desirable.

The exact mechanism for linking Connected Systems resources and 3D objects is implemented will be specified in a future OGC Best Practice document.

## C.12. OGC Sensor Observation Service (SOS) Standard

The functionality provided by a conformant implementation of the OGC SOS Standard (web service) is fully supported by Parts 1 and 2 of the OGC API – Connected Systems Standard. The following table lists the mappings between SOS service operations and corresponding OGC API – Connected Systems (CS API) resources:

SOS OPERATION	CS API RESOURCE	API VERB	COMMENTS
GetCapabilities	Landing Page	GET	
DescribeSensor	System	GET	GET on collection using the UID filter.
InsertSensor	System	POST	
DeleteSensor	System	DELETE	
GetObservation	Observation	GET	GET on collection.
GetObservationById	Observation	GET	GET on resource ID.
InsertObservation	Observation	POST	+ POST on SamplingFeature to add embedded features of interest.
GetResult	Observation	GET	must use SWE Common format.
InsertResult	Observation	POST	must use SWE Common format.
GetResultTemplate	DataStreamSchema	GET	Retrieve the DataStream schema.
InsertResultTemplate	DataStream	POST	Create a DataStream with its schema.
GetFeatureOfInterest	SamplingFeature	GET	

## C.13. OGC Sensor Planning Service (SPS) Standard

The functionality provided by a conformant implementation of the SPS Standard (web service) is fully supported by Parts 1 and 2 of the OGC API – Connected Systems Standard. The following table lists the mappings between SPS service operations and corresponding OGC API – Connected Systems (CS API) resources:

SPS OPERATION	CS API RESOURCE	API VERB	COMMENTS
GetCapabilities	Landing Page	GET	
DescribeSensor	System	GET	GET on collection using the UID filter.
DescribeTasking	ControlStream	GET	Retrieve the ControlStream schema.
Submit	Command	POST	
Update	Command	PUT or PATCH	
Cancel	Command	DELETE	
GetStatus	CommandStatus	GET	
GetTask	Command	GET	
DescribeResultAccess	CommandResult	GET	
GetFeasibility	Command	POST	Feasibility workflow implemented as a linked CommandStream. Feasibility result provided as CommandResult.
Reserve	Command	POST	Reservation/confirmation workflow implemented as a linked CommandStream.
Confirm	Command	POST	Reservation/confirmation workflow implemented as a linked CommandStream.



# ANNEX D (INFORMATIVE) REVISION HISTORY

---



# ANNEX D (INFORMATIVE) REVISION HISTORY

---

DATE	RELEASE	EDITOR	PRIMARY CLAUSES MODIFIED	DESCRIPTION
2022-12-16	1.0 draft	Alex Robin	All	Initial draft version
2023-04-21	1.0 draft	Alex Robin	All	Migration to Metanorma
2023-12-07	1.0 draft	Alex Robin	All	Incorporated feedback from various SWG
2024-05-21	1.0 draft	Alex Robin	All	Incorporated OAB feedback (“resources endpoint” terminology)
2024-09-05	1.0 draft	Alex Robin	All	Updated ATS



# 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] 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>.
- [5] 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>.
- [6] 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>.
- [7] Ingo Simonis, Johannes Echterhoff: OGC 09-000, *OGC® Sensor Planning Service Implementation Standard*. Open Geospatial Consortium (2011).
- [8] OGC API – Moving Features – Part 1: Core, Version 1.0.draft. [https://opengeospatial.github.io/ogcapi-movingfeatures/standard/standard\\_document.html](https://opengeospatial.github.io/ogcapi-movingfeatures/standard/standard_document.html)
- [9] QUDT Quantity Kinds, Version 2.1. [https://www.qudt.org/doc/DOC\\_VOCAB-QUANTITY-KINDS.html](https://www.qudt.org/doc/DOC_VOCAB-QUANTITY-KINDS.html)

