OGC API - Coverages - Part 1

Core

Open Geospatial Consortium

Submission Date: <yyyy-mm-dd>

Approval Date: <yyyy-mm-dd>

Publication Date: 2021-04-16

External identifier of this OGC® document: http://www.opengis.net/doc/is/ogcapi-coverages-1/1.0

Internal reference number of this OGC® document: 19-087

Version: 0.0.6

Category: OGC® Implementation Specification

Editor: Charles Heazel

OGC API - Coverages - Part 1: Core

Document type: OGC® Implementation Specification

Document stage: Draft

Document language: English

License Agreement

Permission is hereby granted by the Open Geospatial Consortium, ("Licensor"), free of charge and subject to the terms set forth below, to any person obtaining a copy of this Intellectual Property and any associated documentation, to deal in the Intellectual Property without restriction (except as set forth below), including without limitation the rights to implement, use, copy, modify, merge, publish, distribute, and/or sublicense copies of the Intellectual Property, and to permit persons to whom the Intellectual Property is furnished to do so, provided that all copyright notices on the intellectual property are retained intact and that each person to whom the Intellectual Property is furnished agrees to the terms of this Agreement.

If you modify the Intellectual Property, all copies of the modified Intellectual Property must include, in addition to the above copyright notice, a notice that the Intellectual Property includes modifications that have not been approved or adopted by LICENSOR.

THIS LICENSE IS A COPYRIGHT LICENSE ONLY, AND DOES NOT CONVEY ANY RIGHTS UNDER ANY PATENTS THAT MAY BE IN FORCE ANYWHERE IN THE WORLD.

THE INTELLECTUAL PROPERTY IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NONINFRINGEMENT OF THIRD PARTY RIGHTS. THE COPYRIGHT HOLDER OR HOLDERS INCLUDED IN THIS NOTICE DO NOT WARRANT THAT THE FUNCTIONS CONTAINED IN THE INTELLECTUAL PROPERTY WILL MEET YOUR REQUIREMENTS OR THAT THE OPERATION OF THE INTELLECTUAL PROPERTY WILL BE UNINTERRUPTED OR ERROR FREE. ANY USE OF THE INTELLECTUAL PROPERTY SHALL BE MADE ENTIRELY AT THE USER'S OWN RISK. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR ANY CONTRIBUTOR OF INTELLECTUAL PROPERTY RIGHTS TO THE INTELLECTUAL PROPERTY BE LIABLE FOR ANY CLAIM, OR ANY DIRECT, SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES, OR ANY DAMAGES WHATSOEVER RESULTING FROM ANY ALLEGED INFRINGEMENT OR ANY LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR UNDER ANY OTHER LEGAL THEORY, ARISING OUT OF OR IN CONNECTION WITH THE IMPLEMENTATION, USE, COMMERCIALIZATION OR PERFORMANCE OF THIS INTELLECTUAL PROPERTY.

This license is effective until terminated. You may terminate it at any time by destroying the Intellectual Property together with all copies in any form. The license will also terminate if you fail to comply with any term or condition of this Agreement. Except as provided in the following sentence, no such termination of this license shall require the termination of any third party end-user sublicense to the Intellectual Property which is in force as of the date of notice of such termination. In addition, should the Intellectual Property, or the operation of the Intellectual Property, infringe, or in LICENSOR's sole opinion be likely to infringe, any patent, copyright, trademark or other right of a third party, you agree that LICENSOR, in its sole discretion, may terminate this license without any compensation or liability to you, your licensees or any other party. You agree upon termination of any kind to destroy or cause to be destroyed the Intellectual Property together with all copies in any form, whether held by you or by any third party.

Except as contained in this notice, the name of LICENSOR or of any other holder of a copyright in all or part of the Intellectual Property shall not be used in advertising or otherwise to promote the sale, use or other dealings in this Intellectual Property without prior written authorization of LICENSOR or such copyright holder. LICENSOR is and shall at all times be the sole entity that may authorize you or any third party to use certification marks, trademarks or other special designations to indicate compliance with any LICENSOR standards or specifications. This Agreement is governed by the laws of the Commonwealth of Massachusetts. The application to this Agreement of the United Nations Convention on Contracts for the International Sale of Goods is hereby expressly excluded. In the event any provision of this Agreement shall be deemed unenforceable, void or invalid, such provision shall be modified so as to make it valid and enforceable, and as so modified the entire Agreement shall remain in full force and effect. No decision, action or inaction by LICENSOR shall be construed to be a waiver of any rights or remedies available to it.

Table of Contents

i. Abstract

<Insert Abstract Text here>

ii. Keywords

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

ogcdoc, OGC document, <tags separated by commas>

iii. Preface

NOTE

Insert Preface Text here. Give OGC specific commentary: describe the technical content, reason for document, history of the document and precursors, and plans for future work. > Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. The Open Geospatial Consortium shall not be held responsible for identifying any or all such patent rights.

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

iv. Submitting organizations

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

Organization name(s)

v. Submitters

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

Name	Affiliation
Charles Heazel	HeazelTech
Stephan Meissl <stephan.meissl@eox.at>	EOX IT Services GmbH
Tom Kralidis <tom.kralidis@canada.ca>	Meteorological Service of Canada
Jerome St-Louis <jerome@ecere.com>	Ecere Corporation

Chapter 1. Scope

This OGC API Coverages (https://github.com/opengeospatial/ogcapi-coverages) specification establishes how to access coverages as defined by the Coverage Implementation Schema (CIS) 1.1 (http://docs.opengeospatial.org/is/09-146r6/09-146r6.html) through a Web API such as those described by the OpenAPI specification (https://www.openapis.org/).

1.1. Service Scope

The functionality provided by API-Coverages resembles that of the OGC Web Coverage Service (WCS) 2.1 Interface Standard. It is expected that Coverage APIs and WCS services will be able to interoperate, allowing developers to pick the solution best suited for their requirements.

The OGC is using an incremental approach to their API development. The initial goal is to develop a relatively simple API standard which will meet the needs of a large percentage of implementors. Additional capabilities will be added based on community demand.

As a result, this API-Coverages Part 1 - Core standard does not provide a full duplication of the WCS capabilities. The functionality covered in Part 1 is:

- Coverage extraction functionality
- Subsetting by domain set
- Subsetting of domain set
- Subsetting by range type (bands)
- Scaling
- Content type negotiation.

1.2. Content Scope

The API-Coverages standard provides access to content which complies with the Coverage Implementation Schema (CIS) 1.1 (http://docs.opengeospatial.org/is/09-146r6/09-146r6.html).

The functionality covered in Part 1 is:

- Gridded or multi-point coverages
- GeneralGridCoverage

Chapter 2. Conformance

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

The one Standardization Target for this standard is Web APIs.

OGC API-Common provides a common foundation for OGC API standards. Therefore, this standard should be viewed as an extension to API-Common. Conformance to this standard requires demonstrated conformance to the applicable Conformance Classes of API-Common.

This standard identifies five Conformance Classes. The Conformance Classes implemented by an API are advertised through the `/conformance` path on the landing page. Each Conformance Class has an associated Requirements Class. The Requirements Classes define the functional requirements which will be tested through the associated Conformance Class.

The Requirements Classes for OGC API-Coverages are:

- **Core**
- **Subset**
- **Scaling**
- **RangeSubset**
- **Tiles**
- **DomainSetSubset**
- **HTML**
- **JSON**
- **OpenAPI 3.0**

The *Core* Requirements Class is the minimal useful service interface for an OGC Coverages API. The requirements specified in this Requirements Class are mandatory for all implementations of API-Coverages.

The *Subset* Requirements Class provides capabilities to select a sub-set of a `Coverage` using a multi-dimensional "bounding box" which is suitable for any coordinate reference system and any dimension.

The *HTML* and *JSON* Requirements Classes address support for encodings commonly used with APIs.

The *OpenAPI 3.0* Requirements Class addresses the use of the OpenAPI 3.0 standard to document and communicate the API Definition.

Chapter 3. References

The following normative documents contain provisions that, through reference in this text, constitute provisions of this document. For dated references, subsequent amendments to, or revisions of, any of these publications do not apply. For undated references, the latest edition of the normative document referred to applies.

- Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T.: IETF RFC 2616, **HTTP/1.1**, RFC 2616
- Rescorla, E.: IETF RFC 2818, **HTTP Over TLS**, RFC 2818
- Klyne, G., Newman, C.: IETF RFC 3339, **Date and Time on the Internet: Timestamps**, RFC 3339
- Berners-Lee, T., Fielding, R., Masinter, L.: IETF RFC 3986, **Uniform Resource Identifier (URI): Generic Syntax**, RFC 3986
- Duerst, M., Suignard, M.: IETF RFC 3987, **Internationalized Resource Identifiers (IRIs)**, RFC 3987
- Gregorio, J., Fielding, R., Hadley, M., Nottingham, M., Orchard, D.: IETF RFC 6570, **URI Template**, RFC 6570
- Nottingham, M.: IETF RFC 8288, **Web Linking**, RFC 8288
- OGC 19-072: **OGC API - Common - Part 1: Core**, (Draft) https://github.com/opengeospatial/oapi_common/blob/Master/19-072.pdf
- OGC 20-024: **OGC API - Common - Part 2: Geospatial Data**, (Draft) https://github.com/opengeospatial/oapi_common/blob/Master/20-024.pdf
- OGC 09-146: **OGC Coverage Implementation Schema (CIS)**, version 1.1, CIS
- OGC 19-008: **OGC GeoTIFF Standard**, Version 1.1, http://docs.opengeospatial.org/is/19-008r4/19-008r4.html
- OGC Schema: **OGC JSON Schema for Coverage Implementation Schema**, version 1.1, 2017, CIS Schema
- OGC 10-090: **OGC Network Common Data Form (NetCDF) Core Encoding Standard**, Version 1.0, http://portal.opengeospatial.org/files/?artifact_id=43732
- OGC 17-089: **OGC Web Coverage Service (WCS) Interface Standard - Core**, Version 2.1, (WCS 2.1)
- Open API Initiative: **OpenAPI Specification 3.0.2**, OpenAPI
- **Schema.org**: Schema.org
- W3C: **HTML5**, W3C Recommendation, HTML5
- OGC: OGC 07-011, **Abstract Specification Topic 6: The Coverage Type and its Subtypes**, version 7.0 (identical to ISO 19123:2005), 2007
- OGC: OGC 08-094, **OGC® SWE Common Data Model Encoding Standard**, version 2, 2011
- OGC: OGC 12-000, **OGC® SensorML: Model and XML Encoding Standard**, version 2, 2014
- OGC: OGC 09-146r2, **GML 3.2.1 Application Schema – Coverages**, version 1.0.1, 2012

- OGC: OGC 16-083, **Coverage Implementation Schema – ReferenceableGridCoverage Extension**, version 1, 2017
- OGC: OGC 09-110r4, **Web Coverage Service (WCS) Core Interface Standard**, version 2, 2012
- OGC: OGC 13-102r2, **Name type specification – Time and index coordinate reference system definitions** (OGC Policy Document), version 1, 2014
- IETF: RFC 7159, **The JavaScript Object Notation (JSON) Data Interchange Format**, https://www.ietf.org/rfc/rfc7159.txt, 2014
- W3C: W3C JSON-LD 1.0, **A JSON-based Serialization for Linked Data**. http://www.w3.org/TR/json-ld/, 2014
- W3C: W3C JSON-LD 1.0 **Processing Algorithms and API**. http://www.w3.org/TR/json-ld-api, 2014
- W3C: W3C RDF 1.1 **Concepts and Abstract Syntax.** https://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/, 2014

Chapter 4. Terms and Definitions

This document uses the terms defined in Sub-clause 5 of OGC API - Common - Part 1: Core (OGC 19-072), which is based on the ISO/IEC Directives, Part 2, Rules for the structure and drafting of International Standards. In particular, the word “shall” (not “must”) is the verb form used to indicate a requirement to be strictly followed to conform to this standard.

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

4.1. Coverage

feature that acts as a function to return values from its range for any direct position within its spatiotemporal domain, as defined in OGC Abstract Topic 6

4.2. Regular grid

grid whose grid lines have a constant distance along each grid axis

4.3. Irregular grid

Grid whose grid lines have individual distances along each grid axis

4.4. Displaced grid

grid whose direct positions are topologically aligned to a grid, but whose geometric positions can vary arbitrarily

4.5. Mesh

coverage consisting of a collection of curves, surfaces, or solids, respectively

4.6. Partition [of a coverage]

separately stored coverage acting, by being referenced in the coverage on hand, as one of its components

4.7. Sensor model

mathematical model for estimating geolocations from recorded sensor data such as digital imagery

4.8. Transformation grid

grid whose direct positions are given by some transformation algorithm not further specified in this standard

Chapter 5. Conventions

The following conventions will be used in this document. Examples of conventions are symbols, abbreviations, use of XML schema, or special notes regarding how to read the document.

5.1. Identifiers

The normative provisions in this standard are denoted by the URI

http://www.opengis.net/spec/ogcapi-coverages-1/1.0

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

5.2. Examples

Most of the examples provided in this standard are encoded in JSON. JSON was chosen because it is widely understood by implementers and easy to include in a text document. This convention should NOT be interpreted as a requirement that JSON must be used. Implementors are free to use any format they desire as long as there is a Conformance Class for that format and the API advertises its support for that Conformance Class.

5.3. Schema

JSON Schema is used throughout this standard to define the structure of resources. These schema are typically represented using YAML encoding. This convention is for the ease of the user. It does not prohibit the use of another schema language or encoding. Nor does it indicate that JSON schema is required. Implementations should use a schema language and encoding appropriate for the format of the resource.

5.4. UML Notation

Diagrams using the Unified Modeling Language (UML) adhere to the following conventions:

- UML elements having a package name of “GML” are those defined in the UML model of GML 3.2.1
- UML elements having a package name of “SWE Common” are those defined in the UML model of SWE Common 2.0
- UML elements not qualified with a package name, or with “CIS”, are those defined in this standard.

Further, in any class where an attribute name or association role name is identical to a name in some superclass the local definition overrides the superclass definition.

5.5. Namespace Prefix Conventions

UML diagrams and XML code fragments adhere to the namespace conventions shown in Table 1. The namespace prefixes used in this document are not normative and are merely chosen for convenience. The namespaces to which the prefixes correspond are normative, however.

Whenever a data item from a CIS-external namespace is referenced this constitutes a normative dependency on the data structure imported together with all requirements defined in the namespace referenced.

Table 1. Namespace mapping conventions

UML prefix	GML prefix	Namespace URL	Description
CIS	cis	http://www.opengis.net/cis/1.1	Coverage Implementation Schema 1.1
CIS10	cis10	http://www.opengis.net/gmlcov/1.0	Coverage Implementation Schema 1.0
GML	gml	http://www.opengis.net/gml/3.2	GML 3.2.1
GML33	gml33	http://www.opengis.net/gml/3.3	GML 3.3
SWE Common	swe	http://www.opengis.net/swe/2.0	SWE Common 2.0
SML	sml	http://www.opengis.net/sensorml/2.0	SensorML 2.0

Chapter 6. Overview

6.1. General

The OGC API family of standards enable access to resources using the HTTP protocol and its' associated operations (GET, PUT, POST, etc.). OGC API-Common defines a set of features which are applicable to all OGC APIs. Other OGC standards extend API-Common with features specific to a resource type. This OGC API-Coverages standard defines an API with two goals:

1. Provide access to ***Coverages*** conformant to the OGC CIS standard.
2. Provide functionality comparable to that of the OGC WCS standard.

Resources exposed through an OGC API may be accessed through a Universal Resource Identifier (URI). URIs are composed of three sections:

- Dataset distribution API: The endpoint corresponding to a dataset distribution, where the landing page resource as defined in OGC API - Common - Part 1: Core is available (subsequently referred to as Base URI or `{datasetAPI}`)
- Access Paths: Unique paths to Resources
- Query: Parameters to adjust the representation of a Resource or Resources like encoding format or subsetting

Access Paths are used to build resource identifiers. It is recommended, but not required, that these paths

Most resources are also accessible through links on previously accessed resources. Unique relation types are used for each resource.

Table 2 summarizes the access paths and relation types defined in this standard.

Table 2. Coverage API Resources

Resource URI	Relation Type	Description
{datasetAPI}/	`ogc:common:dataset`	Landing page for this dataset distribution
{datasetAPI}/api	`service-desc`	API description (e.g. OpenAPI)
{datasetAPI}/api	`service-doc`	API documentation (optional, e.g. HTML)
{datasetAPI}/conformance	`conformance`	Conformance Classes
{datasetAPI}/collections	`data`	The list off all collections available, some or all of which may be accessible using this Coverage API. Each of these collection objects take the same form as that of the collection resource object described immediately below.

Resource URI	Relation Type	Description
{datasetAPI}/collections/{collectionId}	ogc:common:collection	resource corresponding to the collection with the unique identifier {collectionId}, which may be accessible as a coverage. The resource will describe key elements such as an id, title, description, available crs and extent (the coverage envelope) as well as links to resources pertaining to this collection. For coverages, it will either embed or link to a CIS JSON encoding of both the range type and the domain set. It is comparable to a WCS ***DescribeCoverage*** response, with the exception that the range type and domain set may have to be retrieved separately by following a link to accommodate the case where they may be considerably large, and the domain set may support query parameters to subset it.
Coverages		
{datasetAPI}/collections/{collectionId}/coverage	http://www.opengis.net/def/rel/ogc/1.0/coverage	returns the coverage including all of its components (domain set, range type, range set and metadata), to the extent that it is supported by the selected representation (see format encoding for ways to retrieve in specific formats). It is comparable to a WCS ***GetCoverage*** response.
{datasetAPI}/collections/{collectionId}/coverage/rangeset	http://www.opengis.net/def/rel/ogc/1.0/coverage-rangeset	if supported by the service and by the selected representation, returns only the coverage's range set, i.e., the actual values in the selected representation without any accompanying description or extra information.
{datasetAPI}/collections/{collectionId}/coverage/rangetype	http://www.opengis.net/def/rel/ogc/1.0/coverage-rangetype	if available separately from the collection resource, returns the coverage's range type information, i.e., a description of the data semantics (their components and data type).
{datasetAPI}/collections/{collectionId}/coverage/domainset	http://www.opengis.net/def/rel/ogc/1.0/coverage-domainset	if available separately from the collection resource, returns the coverage's domain set definition (the detailed n-dimensional space covered by the data).
{datasetAPI}/collections/{collectionId}/coverage/metadata	http://www.opengis.net/def/rel/ogc/1.0/coverage-metadata	if available, returns the associated coverage metadata as defined by the CIS model, which may be e.g. domain specific metadata.

Where:

- {datasetAPI} = URI of the landing page for the API distributing the dataset
- {collectionId} = an identifier for a specific coverage (collection)

6.2. Coverage Implementation Schema

OGC API-Coverages specifies the fundamental API building blocks for interacting with coverages. The spatial data community uses the term 'coverage' for homogeneous collections of values located in space/time such as; spatio-temporal sensor, image, simulation, and statistical data.

This OGC API - Coverages standard establishes how to access coverages as defined by the Coverage Implementation Schema (CIS) 1.1 through Web APIs. A high-level view of the CIS data model is provided in Figure 1.

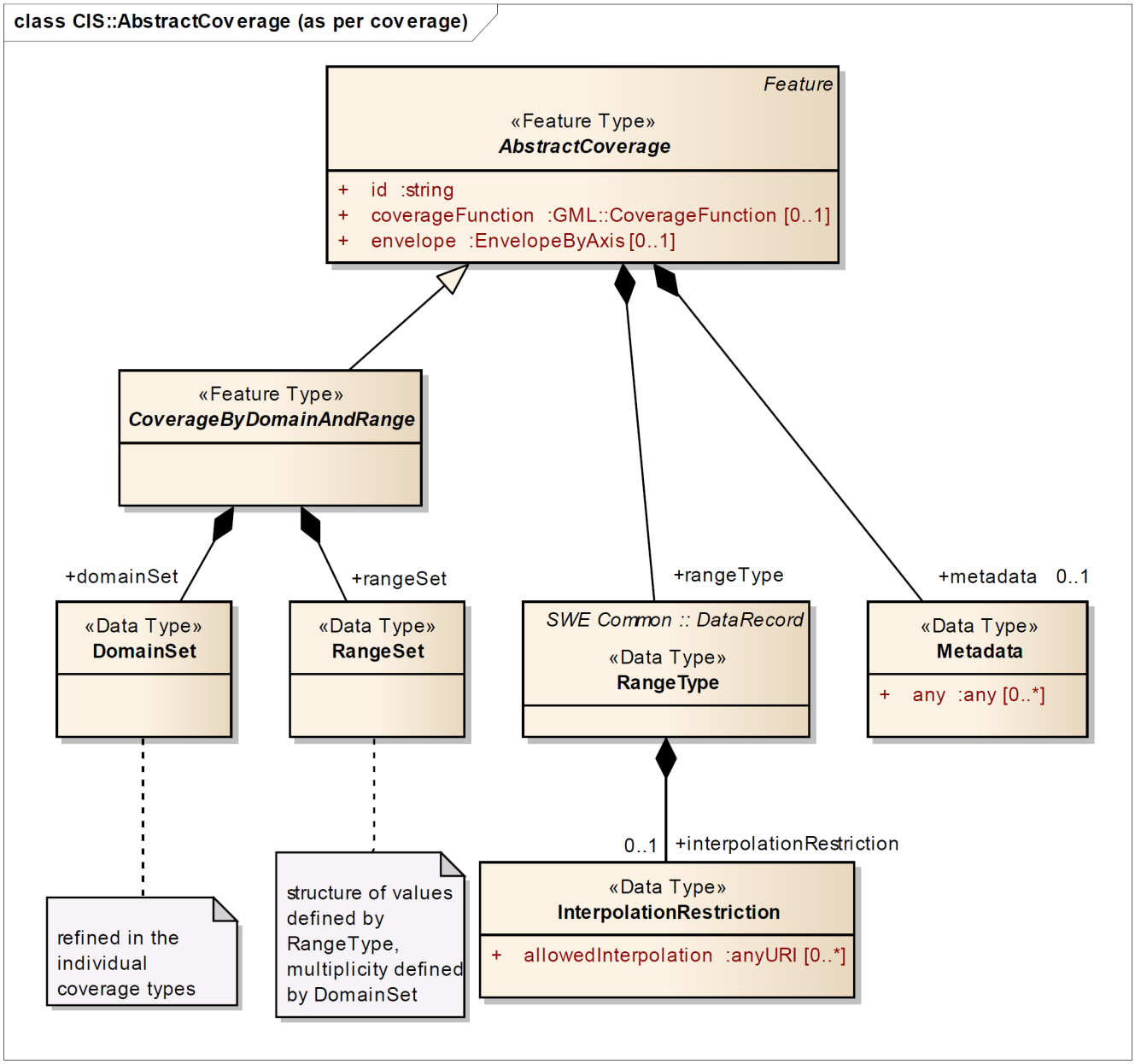


Figure 1. Abstract Coverage

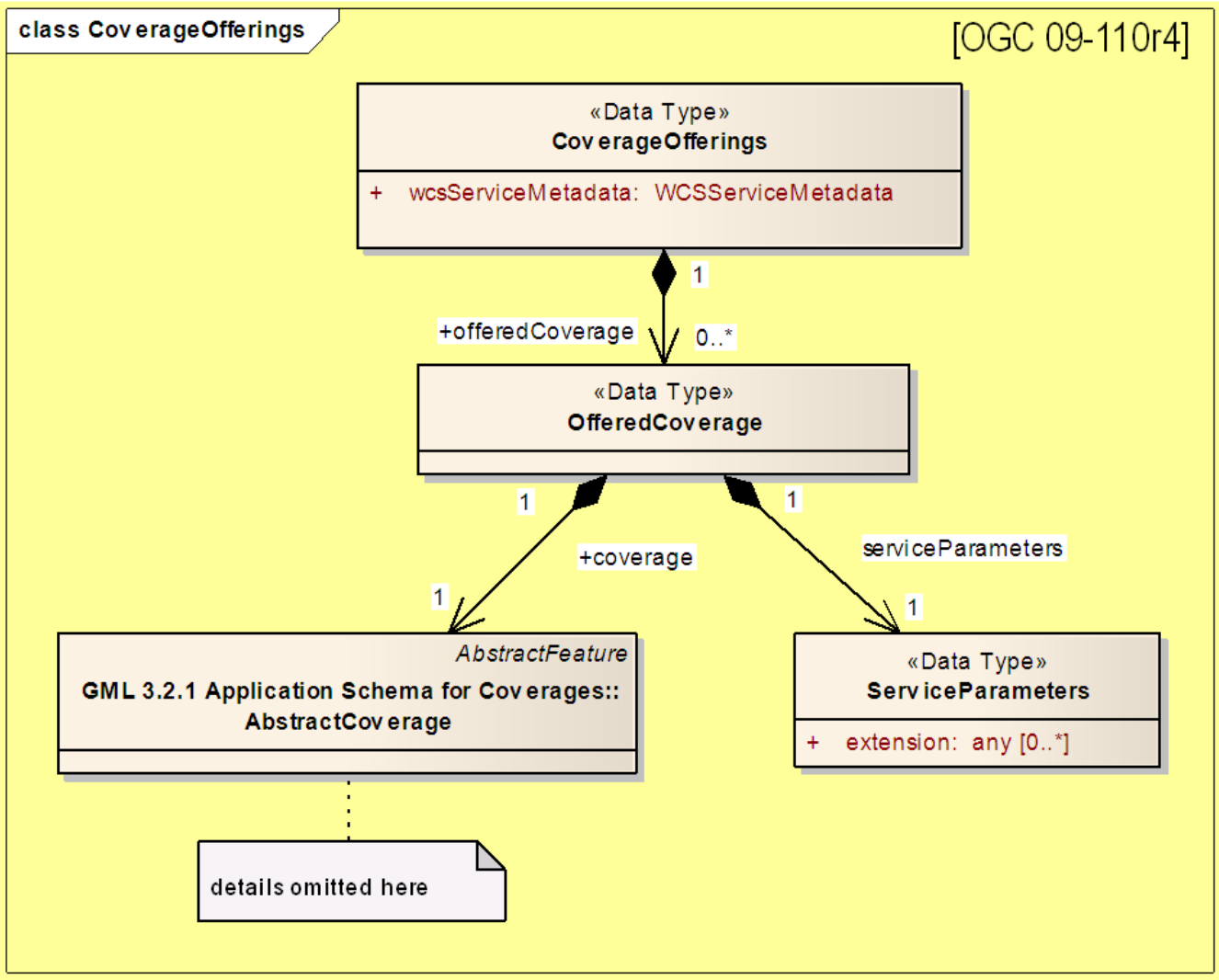
If you are unfamiliar with the term 'coverage', the explanations on the Coverages DWG Wiki

provide more detail and links to educational material. Additionally, Coverages: describing properties that vary with location (and time) in the W3C/OGC Spatial Data on the Web Best Practice document may be considered.

6.3. API Behavior Model

The Coverages API is designed to be compatible but not conformant with the OGC Web Coverage Service. This allows API-Coverage and WCS implementations to co-exist in a single processing environment.

OGC Web Coverage Service standard version 2 has an internal model of its storage organization based on which the classic operations GetCapabilities, DescribeCoverage, and GetCoverage can be explained naturally. This model consists of a single CoverageOffering resembling the complete WCS data store. It holds some service metadata describing service qualities (such as WCS extensions, encodings, CRSs, and interpolations supported, etc.). At its heart, this offering holds any number of OfferedCoverages. These contain the coverage payload to be served, but in addition can hold coverage-specific service-related metadata (such as the coverage's Native CRS).



Discussion has shown that the API model also assumes underlying service and object descriptions, so a convergence seems possible. In any case, it will be advantageous to have a similar "mental model" of the server store organization on hand to explain the various functionalities introduced below.

6.4. Dependencies

The OGC API-Coverages standard is an extension of the OGC API-Common standard. Therefore, an implementation of API-Coverages must first satisfy the appropriate Requirements Classes from API-Common. Table 3 Identifies the API-Common Requirements Classes which are applicable to each section of this Standard. Instructions on when and how to apply these Requirements Classes are provided in each section.

Table 3. Mapping API-Coverages Sections to API-Common Requirements Classes

API-Coverage Section	API-Common Requirements Class
API Landing Page	http://www.opengis.net/spec/ogcapi_common-1/1.0/req/core
API Definition	http://www.opengis.net/spec/ogcapi_common-1/1.0/req/core
Declaration of Conformance Classes	http://www.opengis.net/spec/ogcapi_common-1/1.0/req/core
Collection Access	http://www.opengis.net/spec/ogcapi_common-2/1.0/req/collections
OpenAPI 3.0	http://www.opengis.net/spec/ogcapi_common-1/1.0/req/oas30
JSON	http://www.opengis.net/spec/ogcapi_common-1/1.0/req/json
HTML	http://www.opengis.net/spec/ogcapi_common-1/1.0/req/html

Chapter 7. General Requirements

The following requirements and recommendations are applicable to all OGC Web APIs.

7.1. HTTP 1.1

The standards used for Web APIs are built on the HTTP protocol. Therefore, conformance with HTTP or a closely related protocol is required.

Requirement 1	**/req/general/http**
A	OGC Web APIs SHALL conform to HTTP 1.1.
B	If the API supports HTTPS, then the API SHALL also conform to HTTP over TLS.

7.2. HTTP Status Codes

Table 4 lists the main HTTP status codes that clients should be prepared to receive. This includes support for specific security schemes or URI redirection. In addition, other error situations may occur in the transport layer outside of the server.

Table 4. Typical HTTP status codes

Status code	Description
`200`	A successful request.
`302`	The target resource was found but resides temporarily under a different URI. A 302 response is not evidence that the operation has been successfully completed.
`303`	The server is redirecting the user agent to a different resource. A 303 response is not evidence that the operation has been successfully completed.
`304`	An entity tag was provided in the request and the resource has not changed since the previous request.
`307`	The target resource resides temporarily under a different URI and the user agent MUST NOT change the request method if it performs an automatic redirection to that URI.
`308`	Indicates that the target resource has been assigned a new permanent URI and any future references to this resource ought to use one of the enclosed URIs.
`400`	The server cannot or will not process the request due to an apparent client error. For example, a query parameter had an incorrect value.
`401`	The request requires user authentication. The response includes a `WWW-Authenticate` header field containing a challenge applicable to the requested resource.

Status code	Description
403	The server understood the request, but is refusing to fulfill it. While status code 401 indicates missing or bad authentication, status code 403 indicates that authentication is not the issue, but the client is not authorized to perform the requested operation on the resource.
404	The requested resource does not exist on the server. For example, a path parameter had an incorrect value.
405	The request method is not supported. For example, a POST request was submitted, but the resource only supports GET requests.
406	Content negotiation failed. For example, the Accept header submitted in the request did not support any of the media types supported by the server for the requested resource.
500	An internal error occurred in the server.

The return status codes described in Table 4 do not cover all possible conditions.

Permission 1	/per/general/additional-status-codes
A	Servers MAY implement additional capabilities provided by the HTTP protocol. Therefore, they MAY return status codes in addition to those listed in Table 4.

When a server encounters an error in the processing of a request, it may wish to include information in addition to the status code in the response. Since Web API interactions are often machine-to-machine, a machine-readable report would be prefered. IETF RFC 7807 addresses this need by providing "Problem Details" response schemas for both JSON and XML.

Recommendation 1	/rec/general/problem-details
An OGC Web API should include a "Problem Details" report in any error response in accordance with IETF RFC 7807.	

7.3. Query parameters

Requirement 2	/req/general/query-param-unknown
A	IF a request URI includes a query parameter that is not specified in the API definition, THEN The server SHALL return a response with the status code 400.

Requirement 3	/req/general/query-param-invalid

A	IF a request URI includes a query parameter that has an invalid value, THEN The server SHALL return a response with the status code `400`.

The criteria for a parameter to be "specified" in the API definition depends on the API definition language used, the complexity of the resources exposed, and the abilty of the API server to tolerate errors.

Implementors of a service should endeavour to provide as much detail in the server's API definition as the API definition language allows. However, there is no requirement to list every endpoint for which there is a non-404 behaviour, for it to list every possible query parameter that might affect the behaviour of an endpoint, or for it to list every possible value that each query parameter might accept.

Permission 2	**/per/general/query-param-specified**
A	The specification of a query parameter in the API definition MAY encompass a range of parameter names. Any query parameter which falls within the specified range can be considered "specified" in the API definition. Examples of a parameter range include: • A regular expression which defines the valid parameter names, • A URL Template segment which defines the valid parameter names, • An indication that all parameter names are accepted (no parameter validation).
B	The API definition language chosen may not be capable of expressing the desired range of values. In that case the server SHOULD provide: • A definition of the parameter range which best expresses the intended use of that parameter, • Additional human readable text documenting the actual range of validity.

Permission 3	**/per/general/query-param-tolerance**

A	Servers MAY display tolerance for requests with incorrect query parameters. These acts of tolerance include: • accept alternate capitalizations, spellings, and/or aliases of parameters, • ignore unknown/unrecognized parameters, • return a response with a status code of 30x redirecting the client to a more correct version of the request.
B	Servers should not be excessively tolerant. The response a client receives from the server should be a reasonable response for the request submitted.

7.4. Web Caching

Entity tags are a mechanism for web cache validation and for supporting conditional requests to reduce network traffic. Entity tags are specified by HTTP/1.1 (RFC 7232).

Recommendation 2	**/rec/general/etag**
A	The service SHOULD support entity tags and the associated headers as specified by HTTP/1.1.

7.5. Support for Cross-Origin Requests

If the data is located on another host than the webpage ("same-origin policy"), access to data from a HTML page is by default prohibited for security reasons. A typical example is a web-application accessing feature data from multiple distributed datasets.

Recommendation 3	**/rec/general/cross-origin**
A	If the server is intended to be accessed from a browser, cross-origin requests SHOULD be supported. Note that support can also be added in a proxy layer on top of the server.

Two common mechanisms to support cross-origin requests are:

- Cross-origin resource sharing (CORS)
- JSONP (JSON with padding)

7.6. String Internationalization

If the server supports representing resources in multiple languages, the usual HTTP content

negotiation mechanisms apply. The client states its language preferences in the `Accept-Language` header of a request and the server responds with responses that have linguistic text in the language that best matches the requested languages and the capabilities of the server.

Recommendation 4	**/rec/general/string-i18n**
A	For encodings that support string internationalization, the server SHOULD include information about the language for each string value that includes linguistic text.

For example, if JSON-LD is used as an encoding, the built-in capabilities to annotate a string with its language should be used.

The link object based on RFC 8288 (Web Linking) includes a `hreflang` attribute that can be used to state the language of the referenced resource. This can be used to include links to the same data in, for example, English or French. Just like with multiple encodings, a server that wants to use language-specific links will have to support a mechanism to mint language-specific URIs for resources in order to express links to, for example, the same resource in another language. Again, this document does not mandate any particular approach how such a capability is supported by the server.

7.7. Resource Encodings

A Web API provides access to resources through representations of those resources. One property of a representation is the format used to encode it for transfer. Components negotiate which encoding format to use through the content negotiation process defined in IETF RFC 7231.

Additional content negotiation techniques are allowed, but support is not required of implementations conformant to this Standard.

While this standard does not specify any mandatory encoding, the following encodings are recommended:

HTML encoding recommendation:

Recommendation 5	**/rec/general/html**
A	To support browsing an API with a web browser and to enable search engines to crawl and index the dataset, implementations SHOULD consider supporting an HTML encoding.

JSON encoding recommndation:

Recommendation 6	**/rec/general/json**

A	To support processing of an API with a web applet, implementations SHOULD consider supporting a JSON encoding.

Requirement `/req/general/http` implies that the encoding of a server response is determined using content negotiation as specified by the HTTP RFC.

The section Media Types includes guidance on media types for encodings that are specified in this document.

Note that any server that supports multiple encodings will have to support a mechanism to mint encoding-specific URIs for resources in order to express links, for example, to alternate representations of the same resource. This document does not mandate any particular approach how this is supported by the server.

As clients simply need to dereference the URI of the link, the implementation details and the mechanism how the encoding is included in the URI of the link are not important. Developers interested in the approach of a particular implementation, for example, to manipulate ("hack") URIs in the browser address bar, can study the API definition.

Two common approaches are to use:

- an additional path for each encoding of each resource (this can be expressed, for example, using format specific suffixes like ".html");
- an additional query parameter (for example, "accept" or "f") that overrides the Accept header of the HTTP request.

7.8. Parameter Encoding

The following sections provide the requirements and guidelines for encoding parameters for use in an OGC Web API request.

OGC Web API requests are issued using a Uniform Resource Identifier (URI). The URI syntax is defined in IETF RFC 3986. Rules for building URI Templates can be found in IETF RFC 6570.

The Backus-Naur Form (BNF) definition of a URI is provided in Annex F.

7.8.1. Capitalization

IETF RFC 3986 sections 6.2.2.1 and 2.1 provide the requirements for capitalization in URIs.

Requirement 4	**/req/general/query-param-capitalization**
A	Parameter names and values SHALL be case sensitive.

B	IF a parameter name or value includes a percent encoded (escaped) character, THEN The upper case hexadecimal digits ("A" through "F") of that percent encoded character SHALL be equivalent to the lower case digits "a" through "f" respectively.

In order to minimize capitalization issues for implementors of OGC Web API standards:

Recommendation 7	**/rec/general/query-param-capitalization**
A	Query parameter names SHOULD be in kebab case. (lower case with dash "-" delimiters)
B	Query parameter values are usually reflective of the internal structure of the target resource. Unless otherwise specified, these values SHOULD be in Kebab case.

A Web API may allow filtering on properties of the target resource. In that case, the parameter name would be the name of the resource property. These names are defined by the standards and specifications defining the resource and cannot be constrained by this standard.

7.8.2. Parameter Value Lists

Parameters may pass more than one value. These lists of parameter values may be passed in two ways.

1. Repeated name:value pairs where the parameter name is repeated for each value in the list
2. A parameter name followed by a delimited list of values.

The folowing requirements define how to encode a delimited list (case 2) of parameter values. They do not apply if replication (case 1) is uses.

Requirement 5	**/req/general/query-param-list-delimiter**
A	Parameters values containing lists SHOULD specify the delimiter to be used in the API definition.
B	The default list item delimiter SHALL be the comma (",").

Requirement 6	**/req/general/query-param-list-escape**

A	Any list item values which include a space or comma SHALL escape the space or comma character using the URL encoding rules from IETF RFC 3986

Requirement 7	**/req/general/query-param-list-empty**
A	All empty entries SHALL be represented by the empty string ("").

Thus, two successive commas indicates an empty item, as does a leading comma or a trailing comma. An empty list ("") can either be interpreted as a list containing no items or as a list containing a single empty item, depending on the context.

7.8.3. Numeric and Boolean Values

Geospatial is a mathematical discipline. The clear and accurate exchange of mathematical values is essential. The encoding rules in this section standarize the encoding of numeric and boolean primitives when included in a URL. These rules are based on the computer science basic data types identified by Kernighan and Ritchie.

Boolean values conform to the following requirement.

Requirement 8	**/req/general/query-param-value-boolean**
A	Boolean values shall be represented by the lowercase strings "true" and "false", representing Boolean true and false respectively.

Integer values conform to the following requirement.

Requirement 9	**/req/general/query-param-value-integer**
A	Integer values SHALL be represented by a finite-length sequence of decimal digits with an optional leading negative "-" sign. Positive values are assumed is the leading sign is omitted.

Real numbers can be represented using either the decimal or double (exponential) format. The decimal format is typically used except for very large or small values.

Decimal values conform to the following requirement.

Requirement 10	**/req/general/query-param-value-decimal**

A	Decimal values SHALL be represented by a finite-length sequence of decimal digits separated by a period as a decimal indicator. • An optional leading negative sign ("-") is allowed. • If the sign is omitted, positive ("+") is assumed. • Leading and trailing zeroes are optional. • If the fractional part is zero, the period and following zero(es) can be omitted.

Double values conform to the following requirement.

Requirement 11	**/req/general/query-param-value-double**
A	Double values SHALL be represented by a mantissa followed, optionally, by the character "e", followed by an exponent.
B	The exponent SHALL be an integer.
C	The mantissa SHALL be a decimal number.
D	The representations for exponent and mantissa SHALL follow the lexical rules for integer and decimal.
E	If the "e" and the following exponent are omitted, an exponent value of 0 SHALL be assumed.

Special values conform to the following requirement.

Requirement 12	**/req/general/query-param-value-special**
A	The special values positive and negative infinity and not-a-number SHALL be represented using the strings `inf`, `-inf` and `nan`, respectively.

Chapter 8. Requirements Class "Platform"

The Platform Requirements Class defines a set of common capabilities which are applicable to any OGC Web API. Those capabilities provide the platform upon which resource-specific APIs can be built. This section describes those capabilities and any modifications needed to better support Coverage resources.

8.1. Resource Requirements

The `platform` resources are introduced in Table 5. The requirements and recommendations applicable to these resources are provided in this sections below.

Table 5. Platform Resources

URI Path	Description
"/"	the landing page
"/api"	the API Definition document for this API
"/conformance"	the conformance information for this API

8.1.1. API landing page

A Web API has a single landing page on the `{root}` node.

The purpose of the landing page is to provide clients with a starting point for using the API. Any resource exposed through an API can be accessed by following paths or links starting from the landing page.

The landing page includes three metadata elements; title, description, and attribution. These three elements describe the API as a whole. Clients can expect to encounter metadata which is more resource-specific as they follow links and paths from the landing page.

While the three metadata elements are defined as text strings, the attribution element is special. Specifically, it can contain markup text. Markup allows a text string to import images and format text. The capabilities are only limited by the markup language used. See the example landing page for an example of the use of markup in the attribution element.

8.1.1.1. Operation

Requirement 13	**/req/platform/root-op**
A	The server SHALL support the HTTP GET operation on the URI `{root}/`.
B	The response to the HTTP GET request issued in `A` SHALL satisfy requirement /req/platform/root-success.

8.1.1.2. Response

Requirement 14	/req/platform/root-success
A	A successful execution of the operation SHALL be reported as a response with an HTTP status code `200`.
B	The content of that response SHALL be based upon the schema landingPage.json and include links to the following resources: • `/api` (relation type 'service-desc' or 'service-doc') • `/conformance` (relation type 'http://www.opengis.net/def/rel/ogc/1.0/conformance')

The landing page returned by this operation is based on the following JSON schema.

landingPage.json

```
{
"$schema": "http://json-schema.org/draft-07/schema#",
"title": "Landing Page Schema",
"description": "JSON schema for the OGC API - Common landing page",
"type": "object",
"required": [
    "links"
    ],
"properties": {
    "title": {
        "title": "The title of the API.",
        "description": "While a title is not required, implementors are strongly
advised to include one.",
        "type": "string"
        },
    "description": {
        "description": "A textual description of the API",
        "type": "string"
        },
    "attribution" : {
        "type" : "string",
        "title" : "attribution for the API",
        "description" : "The `attribution` should be short and intended for
presentation to a user, for example, in a corner of a map. Parts of the text can be
links to other resources if additional information is needed. The string can include
HTML markup."
        },
    "links": {
        "description": "Links to the resources exposed through this API.",
        "type": "array",
        "items": {"$href": "link.json"}
        }
    },
    "additionalProperties": true
}
```

It is recommended that OGC Web APIs provide a set of Service Metadata which identifies the service and provides information about the service provider.

Recommendation 8	**/rec/platform/service-metadata**
A Web API SHOULD provide service metadata.	
A	A Web API service SHOULD provide one or more service metadata resources accessible by an HTTP GET operation.

B	The landing page for a Web API service SHOULD provide links to the service metadata resources using the relation type `service-meta`.
C	A successful execution of the operation SHOULD be reported as a response with an HTTP status code `200`.

Additional information about Service Metadata can be found in the OAPI-Common Users Guide.

Examples of OGC landing pages are provided in Example Landing Pages.

In addition to the required resources, links to additional resources may be included in the Landing Page.

8.1.1.3. Error Situations

See HTTP Status Codes for general guidance.

8.1.2. API Definition

Every API should provide an API Definition resource which describes capabilities provided by that API. This resource can be used by developers to understand the API, by software clients to connect to the server, and by development tools to support the implementation of servers and clients.

8.1.2.1. Operation

Requirement 15	**/req/platform/api-definition-op**
A	The server SHALL support the HTTP GET operation on all links from the landing page which have the relation type `service-desc`.
B	The server SHALL support the HTTP GET operation on all links from the landing page which have the relation type `service-doc`.
C	The responses to all HTTP GET requests issued in `A` and `B` SHALL satisfy requirement /req/platform/api-definition-success.

Recommendation 9	**/rec/platform/api-definition-op**
A	The server SHOULD support the HTTP GET operation on the URI `{root}/api`.
B	The response to the HTTP GET request issued in `A` SHOULD satisfy requirement /req/platform/api-definition-success.

8.1.2.2. Response

Requirement 16	/req/platform/api-definition-success
A	A successful execution of the operation SHALL be reported as a response with a HTTP status code `200`.
B	The content of that response SHALL be an API Definition document.
C	The API Definition document SHALL shall be consistent with the media type identified through HTTP content negotiation.
NOTE:	The `-f` parameter MAY be used to satisfy this requirement.

Recommendation 10	/rec/platform/api-definition-oas
A	If the API definition document uses the OpenAPI Specification 3.0, THEN The document SHOULD conform to the OpenAPI Specification 3.0 requirements class.

8.1.2.3. Error Situations

See HTTP Status Codes for general guidance.

8.1.3. Declaration of Conformance Classes

The OGC Web API Standards define a collection of modules which can be assembled into a Web API. The first question a client will ask when accessing one of these APIs is "what are you?" In other words, what modules were used to create you? Since implementors have a choice on which modules to use, there is no simple answer. The best that can be done is to provide a list of the modules implemented, a declaration of the Conformance Classes.

The list of Conformance Classes is key to understanding and using an OGC Web API. So it is important that they are easy to access. A simple GET using an easily constructed URL is all that should be required. Therefore, the path to the Conformance Declaration is fixed.

Ease of access is also supported by the structure of the Conformance Declaration resource. It is a simple list of URIs. This is a structure that requires almost no parsing and little interpretation. Designed to be accessible to even the simplest client.

8.1.3.1. Operation

Requirement 17	/req/platform/conformance-op

A	The server SHALL support the HTTP GET operation on the URI `{root}/conformance`.
B	The server SHALL support the HTTP GET operation on all links from the landing page which have the relation type `http://www.opengis.net/def/rel/ogc/1.0/conformance`.
C	The responses to all HTTP GET requests issued in `A` and `B` SHALL satisfy requirement /req/platform/conformance-success.

8.1.3.2. Response

Requirement 18	**/req/platform/conformance-success**
A	A successful execution of the operation SHALL be reported as a response with a HTTP status code `200`.
B	The content of that response SHALL be based upon the schema confClasses.json and list all OGC API conformance classes that the API conforms to.

The Conformance Declaration resource returned by this operation is based on the following Conformance Declaration Schema.

Examples of OGC Conformance Declarations are provided in Conformance Examples.

Conformance Declaration Schema

```
{
"$schema": "http://json-schema.org/draft-07/schema#",
"title": "Conformance Declaration Schema",
"description": "This schema defines the resource returned from the /Conformance path",
"type": "object",
"required": [
    "conformsTo"
    ],
"properties": {
    "conformsTo": {
        "type": "array",
        "description": "ConformsTo is an array of URLs.  Each URL should correspond to
a defined OGC Conformance class.  Unrecognized URLs should be ignored",
        "items": {
            "type": "string",
            "example": "http://www.opengis.net/spec/ogcapi-common-1/1.0/conf/core"
            }
        }
    }
}
```

8.1.3.3. Error situations

See HTTP Status Codes for general guidance.

Chapter 9. Requirements Class "GeoData"

OGC Web API Standards start with the assumption that geospatial resources are organized into collections. An API will expose one or more collections. The GeoData Conformance Class defines how to organize and provide access to a collection of collections.

This Requirements Class describes the resources and operations used to discover and query resource collections exposed through an OGC Web API. It does not include any requirements about how resources are aggregated into collections nor about the aggregated resources themselves. That detail is reserved for resource-specific OGC Web API standards.

The three resources defined in this Requirements Class are summarized in Table 6. Detailed requirements for each of these resources are provided in the Resource Requirements section.

Table 6. Collection Resources

Resource	URI	HTTP Method	Description
Collections Metadata	/collections	GET	Information which describes the set of supported Collections
Collection Description	/collections/{collectionId}	GET	Information about a specific collection of geospatial data with links to distribution
Collection Resource	/collections/{collectionId}/items	GET	A specific distribution of geospatial data

This Requirements Class also describes the operations which can be performed on these resources. Requirements for these operations are included with their associated resource descriptions in the Resource Requirements section.

Three parameters are defined for use in these operations. These parameters are documented in the Parameter Requirements section. A summary of the parameters is provided in Table 7.

Table 7. Parameter Summary

Parameter Name	Target	Description
Bounding Box	Extent	Selects resources which have an Extent element that intersects the bounding box
Date-Time	Extent	Selects resources which have an Extent element that intersects the specified time period
Limit	The result set	Limits the number of resources which can be returned in one response

Finally, requirements which have general applicablity are provided in the General Requirements section.

9.1. Resource Requirements

This section expresses the requirements for resources and operations used to discover and query resource collections.

9.1.1. Collections Metadata

OGC APIs typically organize their Spatial Resources into collections. Information about those collections is accessed through the `/collections` path.

9.1.1.1. Operation

Requirement 19	**/req/geodata/rc-md-op**
A	The API SHALL support the HTTP GET operation at the path `/collections`.

9.1.1.2. Parameters

The following query parameters can be used with this operation:

- Bounding Box: Limits the contents of the `collections` element to those collections whose spatial extent intersects the bounding box specified by the Bounding Box parameter.
- Date Time Limits the contents of the `collections` element to those collections whose temporal extent intersects the instance or period specified by the Date Time parameter.
- Limit: Limits the contents of the `collections` element to the number of items specified by the Limit parameter.

9.1.1.3. Response

Requirement 20	**/req/geodata/rc-md-success**
A	A successful execution of the operation SHALL be reported as a response with a HTTP status code `200`.
B	The content of that response SHALL be based upon the JSON schema collections.json.

The collections metadata returned by this operation is based on the collections.json JSON schema. Examples of collections metadata are provided in Collections Metadata Example.

collections.json

```
{
    "$schema": "http://json-schema.org/draft-07/schema#",
    "title": "Collections Schema",
    "description": "This schema defines the metadata resource returned from
/collections.",
    "type": "object",
    "required": [
        "links",
        "collections"
        ],
    "properties": {
        "links": {
            "type": "array",
            "items": {"$href": "link.json"}
            },
        "timeStamp": {
            "type": "string",
            "format": "date-time"
            },
        "numberMatched": {
            "type": "integer",
            "min": "0"
            },
        "numberReturned": {
            "type": "integer",
            "min": "0"
            },
        "collections": {
            "type": "array",
            "items": {"$href": "collectionDesc.json"}
            }
        }
    }
```

This schema is further constrained by the following requirements and recommendations.

To support hypermedia navigation, the `links` property must be populated with sufficient hyperlinks to navigate through the whole dataset.

Requirement 21	**/req/geodata/rc-md-links**

A	A `200`-response SHALL include the following links in the `links` property of the response: • A link to this response document (relation: `self`), • A link to the response document in every other media type supported by the API (relation: `alternate`).
B	All links SHALL include the `rel` and `type` link parameters.

Additional information may be available to assist in understanding and using this dataset. Links to those resources should be provided as well.

Recommendation 11	**/rec/geodata/rc-md-descriptions**
A	If external schemas or descriptions exist that provide additional information about the structure or semantics for the resource, a `200`-response SHOULD include links to each of those resources in the `links` property of the response (relation: `describedBy`).
B	The `type` link parameter SHOULD be provided for each link. This applies to resources that describe the whole dataset.

The `timeStamp` property of the Collections Metadata indicates when the response was generated.

Requirement 22	**/req/geodata/rc-timeStamp**
A	If a property `timeStamp` is included in the response, the value SHALL be set to the time stamp when the response was generated.

The `collections` property of the Collections Metadata provides a description of each individual collection hosted by the API. These descriptions are based on the Collection Description Schema. This schema is described in detail in the Collection Description section of this Standard.

Requirement 23	**/req/geodata/rc-md-items**
A	For each spatial resource collection accessible through this API, metadata describing that collection SHALL be provided in the `collections` property of the Collections Metadata.
B	This metadata shall be based on the same schema as the Collection Description resource.

A client may select a subset of the hosted collections using the `bbox` and the `datetime` parameter. These parameters are evaluated against the `extent` element of the Collection Description for each collection.

The requirements governing the processing of these parameters are:

Requirement 24	**/req/geodata/rc-bbox-collections-unsupported**
A	If the `bbox` parameter is provided by the client but it is not supported by the server, then the server SHALL process the request as if the parameter had not been provided.

Requirement 25	**/req/geodata/rc-bbox-collections-response**
A	If the `bbox` parameter is provided by the client and supported by the server, then only extents that have a spatial geometry that intersect the geometry specified by the `bbox` parameter SHALL be part of the result set.
B	If an extent has multiple spatial geometries, it is the decision of the server whether only a single spatial geometry is used to determine the extent or all relevant geometries.
C	The `bbox` parameter SHALL also match all collections that do not have an extent element with a spatial geometry.
D	The bounding box parameter is provided with four or six numbers, depending on whether the coordinate reference system includes a vertical axis (height or depth): • Lower left corner, coordinate axis 1 • Lower left corner, coordinate axis 2 • Minimum value, coordinate axis 3 (optional) • Upper right corner, coordinate axis 1 • Upper right corner, coordinate axis 2 • Maximum value, coordinate axis 3 (optional)
E	The values for the CRS axis 1 and 2 SHALL be interpreted as WGS84 longitude/latitude (http://www.opengis.net/def/crs/OGC/1.3/CRS84) unless a different coordinate reference system is specified in a parameter `bbox-crs`.

F	The coordinate values SHALL be within the extent specified for the coordinate reference system.

Requirement 26	**/req/geodata/rc-time-collections-unsupported**
A	If the `datetime` parameter is provided by the client but it is not supported by the server, then the server SHALL process the request as if the parameter had not been provided.

Requirement 27	**/req/geodata/rc-time-collections-response**
A	If the `datetime` parameter is provided by the client and supported by the server, then only extents that have a temporal geometry that intersects the interval specified by the `datetime` parameter SHALL be part of the result set.
B	If an extent has multiple temporal geometries, the API implementor decides whether only a single temporal geometry is used to determine the extent or all relevant temporal geometry.
C	The `datetime` parameter SHALL match all collections which do not have an extent element with a temporal geometry.

D	Temporal geometries are either a date-time value or a time interval. The parameter value conforms to the following syntax (using ABNF):
	`interval-closed = date-time "/" date-time`
	`interval-open-start = "../" date-time`
	`interval-open-end = date-time "/.."`
	`interval = interval-closed / interval-open-start / interval-open-end`
	`datetime = date-time / interval`

E	The syntax of `date-time` is specified by RFC 3339, 5.6.
F	Open ranges in time intervals at the start or end are supported using a double-dot (`..`).

The number of collections returned in a response may be further limited by the client using the `limit` parameter. When applied against the `/collections` resource, the `limit` parameter indicates the maximum number of collections which should be included in a single response.

Requirement 28	/req/geodata/rc-limit-collections-unsupported
A	If the `limit` parameter is provided by the client but it is not supported by the server, then the server SHALL process the request as if the parameter had not been provided.

NOTE This requirement does not preclude server imposed limits on the size of a response.

Requirement 29	/req/geodata/rc-limit-collections-response
A	If the `limit` parameter is provided by the client and supported by the server, then the response SHALL not contain more items in the `collections` element than specified by the `limit` parameter.
B	If the API definition specifies a maximum value for the `limit` parameter, the response SHALL not contain more items in the `collections` element than this maximum value.

The `numberMatched` property of the Collections Metadata indicates the number of collection descriptions included in the Collections Metadata. This may be a subset of the total set of collections hosted by the API. Selection of which collections to include in a subset is controled through the `bbox`, `datetime` and other selection parameters provided by the client.

Requirement 30	/req/geodata/rc-numberMatched
A	If a property `numberMatched` is included in the response, the value SHALL be identical to the number of hosted collections that meet the selection parameters provided by the client. Selection parameters include `bbox`, `datetime` or additional filter parameters.
B	A server MAY omit this information in a response, if the information about the number of matching resources is not known or difficult to compute.

The Collections Metadata response should describe all of the collections that meet the client's selection criteria. However, in some cases that is impractical. An API implementation has the option of limiting the size of the Collections Metadata response.

Permission 4	/per/geodata/rc-md-items

A	To support servers with many collections, servers MAY limit the number of items included in the `collections` property.

The number of collection descriptions included in a Collections Metadata response may be a subset of the number matched. In that case, the `numberReturned` property of the Collections Metadata indicates the number of collection descriptions returned in this "page" of the Collections Metadata.

Requirement 31	**/req/geodata/rc-numberReturned**
A	If a property `numberReturned` is included in the response, the value SHALL be identical to the number of items in the `collections` array in the Collections Metadata document.
B	A server MAY omit this information in a response, if the information about the number of resources in the response is not known or difficult to compute.

If the Collections Metadata contains a subset of the selected collection descriptions (numberReturned is less than numberMatched) then the Collections Metadata should contain links for navigating to the rest of the collection descriptions as described in the limit parameter section.

9.1.1.4. Error situations

See HTTP Status Codes for general guidance.

9.1.2. Collection Description

Each resource collection is described by a set of metadata. That metadata is accessed directly using the `/collections/{collectionId}` path or as an entry in the `collections` property of the Collections Metadata resource.

9.1.2.1. Operation

Requirement 32	**/req/geodata/src-md-op**
A	The API SHALL support the HTTP GET operation at the path `/collections/{collectionId}`.
B	The parameter `collectionId` is each `id` property in the resource collections response (JSONPath: `$.collections[*].id`).

9.1.2.2. Parameters

No parameters have been standardized for this operation.

9.1.2.3. Response

Requirement 33	/req/geodata/src-md-success
A	A successful execution of the operation SHALL be reported as a response with a HTTP status code `200`.
B	The content of that response SHALL be based upon the JSON schema collectionInfo.json.
C	The content of that response SHALL be consistent with the content for this resource collection in the `/collections` response. That is, the values for `id`, `title`, `description` and `extent` SHALL be identical.

Collection Descriptions are based on the Collection Description Schema. Examples of Collection Descriptions are provided in Collection Description Examples.

Collection Description Schema

```
{
    "$schema": "http://json-schema.org/draft-07/schema#",
    "title": "Collection Description Schema",
    "description": "This schema defines metadata resource returned from
/collections/{collectionId}.",
    "type": "object",
    "required": [
        "id",
        "links"
        ],
    "properties": {
        "id": {
            "description": "identifier of the collection used, for example, in URIs",
            "type": "string"
            },
        "title": {
            "description": "human readable title of the collection",
            "type": "string"
            },
        "description": {
            "description": "a description of the members of the collection",
            "type": "string"
            },
        "attribution" : {
            "type" : "string",
            "title" : "attribution for the collection",
            "description" : "The `attribution` should be short and intended for
presentation to a user, for example, in a corner of a map. Parts of the text can be
links to other resources if additional information is needed. The string can include
```

```
HTML markup."
            },
        "links": {
            "type": "array",
            "items": {"$href": "link.json"}
            },
        "extent": {"$href": "extent.json"},
        "itemType": {
            "description": "An indicator about the type of the items in the
collection.",
            "type": "string"
            },
        "crs": {
            "description": "the list of coordinate reference systems supported by the
API; the first item is the default coordinate reference system",
            "type": "array",
            "items": {
                "type": "string"
                },
            "default": [
                "http://www.opengis.net/def/crs/OGC/1.3/CRS84"
                ],
            "example": [
                "http://www.opengis.net/def/crs/OGC/1.3/CRS84",
                "http://www.opengis.net/def/crs/EPSG/0/4326"
                ]
            }
        }
    }
```

The purpose of the Collection Description is to provide clients with a basic understanding of a single collection. It also serves as a starting point for further navigation through the collection. Many of the properties of the Collection Description are self-explanitory. However, a few call for additional explanation.

Attribution

The attribution element is a special type of string property. Specifically, it can contain markup text. Markup allows a text string to import images and format text. The capabilities are only limited by the markup language used. See the example collection description for an example of the use of markup in the attribution element.

Item Type

A Collection Description describes a collection of items. The itemType property identifies the type of item which has been collected.

Recommendation 12	**/rec/geodata/rc-md-item-type**

A	If a resource at the path `/collection/{collectionId}/items` is supported by the API and is accessible, then the `itemType` key SHOULD be included in the collection object to indicate the type of the items (e.g. `feature` or `record`).

Links

To support hypermedia navigation, the `links` property must be populated with sufficient hyperlinks to navigate through the whole dataset.

Requirement 34	**/req/geodata/rc-md-items-links**
A	`200`-response SHALL include the following links in the `links` property of the response: • A link to this response document (relation: `self`), • A link to the response document in every other media type supported by the API (relation: `alternate`).
B	The `links` property of the response SHALL include an item for each supported encoding of that collection with a link to the collection resource (relation: `items`).
B	All links SHALL include the `rel` and `type` properties.

Additional information may be available to assist in understanding and using this dataset. Links to those resources should be provided as well.

Recommendation 13	**/rec/geodata/rc-md-items-descriptions**
A	If external schemas or descriptions exist that provide additional information about the structure or semantics of the collection, a `200`-response SHOULD include links to each of those resources in the `links` property of the response (relation: `describedBy`).
B	The `type` link parameter SHOULD be provided for each link.

Extent

The extent property defines a spatial-temporal surface which encompasses all of the items in the collection. Since not all collections are nicely clustered around a single place in space and time, the extent property provides flexibility in how that surface can be defined.

- Bbox provides a set of rectangular bounding boxes which use geographic coordinates to envelope portions of the collection. Typically only the first element would be populated.

Additional boxes may be useful, for example, when the collection is clustered in multiple, widely-separated locations.

- Envelope addresses the case where the explicit identification of axis is desired. Indoor navigation, for example, may use this option.
- Interval provides a set of temporal periods. Typically only the first temporal period would be populated. However, like bbox, additional periods can be added if the collection does not form a single temporal cluster.

Requirement 35	**/req/geodata/rc-md-extent**
A	For each spatial resource collection, the `extent` property, if provided, SHALL define boundaries that encompass the spatial and temporal properties of all of the resources in this collection. The temporal extent may use `null` values to indicate an open time interval.
B	If a spatial resource has multiple properties with spatial or temporal information, it is the decision of the API implementation whether only a single spatial or temporal geometry property is used to determine the extent or all relevant geometries.

Recommendation 14	**/rec/geodata/rc-md-extent**
A	If an extent contains multiple spatial boundaries (multiple bbox, a bbox and envelope, etc.), then the extent SHOULD include in the first bbox a boundary which represents the union of all of the other boundaries.
B	If an extent contains multiple temporal intervals, then the extent SHOULD include as the first interval an interval which represents the union of all of the other intervals.

Recommendation 15	**/rec/geodata/rc-md-extent-single**
A	While the spatial and temporal extents support multiple bounding boxes (`bbox` array) and time intervals (`interval` array) for advanced use cases, implementations SHOULD provide only a single bounding box or time interval unless the use of multiple values is important for the use of the dataset and agents using the API are known to be support multiple bounding boxes or time intervals.

Permission 5	/per/geodata/rc-md-extent-extensions
A	API-Common only specifies requirements for spatial and temporal extents. However, the `extent` object MAY be extended with additional members to represent other extents, such as thermal or pressure ranges.
B	API-Common only supports • Spatial extents in CRS84 or CRS84h and • Temporal extents in the Gregorian calendar These are the only *enum* values in extent.yaml).
C	Extensions MAY add additional reference systems to the `extent` object.

9.1.2.4. Error situations

See HTTP Status Codes for general guidance.

If the parameter `collectionId` does not exist on the server, the status code of the response will be `404` (see Table 4).

9.1.3. Collection Resource

A collection resource is the content of the collection as opposed to metadata about that collection. This standard defines the general behavior of this operation, but detailed requirements are the purview of the API standard for that resource type.

9.1.3.1. Operation

Requirement 36	/req/geodata/rc-op
A	For every resource collection identified in the resource collections response (path `/collections`), the API SHALL support the HTTP GET operation at the path `/collections/{collectionId}/items`. • The parameter `collectionId` is each `id` property in the resource collections response (JSONPath: `$.collections[*].id`).

9.1.3.2. Parameters

The following parameters may be used with this operation:

- Bounding Box: Selects items from the collection whose spatial extent intersects the bounding

box specified by the Bounding Box parameter.

- Date Time Selects items from the collection whose temporal extent intersects the instance or period specified by the Date Time parameter.
- Limit: Limits the size of the response to the number of items specified by the Limit parameter.

NOTE	Since the type of resource which makes up the collection is not defined, the behavior of any parameters must be tailored to the structure and information content of specific resource types. That tailoring will take place in resource-specific OGC API standards.

9.1.3.3. Response

Requirement 37	**/req/geodata/rc-bbox-items-unsupported**
A	If the `bbox` parameter is provided by the client but it is not supported by the server, then the server SHALL process the request as if the parameter had not been provided.

Requirement 38	**/req/geodata/rc-time-items-unsupported**
A	If the `datetime` parameter is provided by the client but it is not supported by the server, then the server SHALL process the request as if the parameter had not been provided.

Requirement 39	**/req/geodata/rc-limit-items-unsupported**
A	If the `limit` parameter is provided by the client but it is not supported by the server, then the server SHALL process the request as if the parameter had not been provided.

NOTE	This requirement does not preclude server imposed limits on the size of a response.

Requirement 40	**/req/geodata/rc-response**
A	A successful execution of the operation SHALL be reported as a response with a HTTP status code `200`.
B	The response SHALL only include resources selected by the request.

9.1.3.4. Error situations

See HTTP Status Codes for general guidance.

9.2. Parameter Requirements

Query parameters are used in URLs to limit the resources which are returned on a GET request. The GeoData Requirements Class defines three query parameters for use in OGC API standards:

- bbox: Bounding Box
- datetime: Date and Time
- limit: Response resource count limit

The behavior generated by these parameters is specific to the operation and resource upon which they are applied. Those behaviors are described for each resource type and operation in the Resource Requirements section.

Use of these query parameters with any specific operation is optional. Developers of API-GeoData servers should document their supported parameters in the API definition as describe in the Platform Requirements Class.

9.2.1. Parameter bbox

The `bbox` parameter is defined as follows:

Requirement 41	**/req/geodata/rc-bbox-definition**

A	The `bbox` parameter SHALL possess the following characteristics (using an OpenAPI Specification 3.0 fragment):
	
	`name: bbox`
	`in: query`
	`required: false`
	`schema:`
	`  type: array`
	`  minItems: 4`
	`  maxItems: 6`
	`  items:`
	`  type: number`
	`style: form`
	`explode: false`

While the processing of the `bbox` parameter is specific to the resource and operation for which it is applied, there is a general set of requirements which all implementations must address.

Requirement 42	**/req/geodata/rc-bbox-response**

A	If the `bbox` parameter is provided by the client and supported by the server, then only resources that have a spatial geometry that intersects the bounding box SHALL be part of the result set.
B	If a resource 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.
C	The `bbox` parameter SHALL also match all resources in the collection that are not associated with a spatial geometry.
D	The bounding box is provided as four or six numbers, depending on whether the coordinate reference system includes a vertical axis (height or depth): • Lower left corner, coordinate axis 1 • Lower left corner, coordinate axis 2 • Minimum value, coordinate axis 3 (optional) • Upper right corner, coordinate axis 1 • Upper right corner, coordinate axis 2 • Maximum value, coordinate axis 3 (optional)
E	The values for the CRS axis 1 and 2 SHALL be interpreted as WGS84 longitude/latitude (http://www.opengis.net/def/crs/OGC/1.3/CRS84) unless a different coordinate reference system is specified in a parameter `bbox-crs`.
F	The coordinate values SHALL be within the extent specified for the coordinate reference system.

"Intersects" means that a coordinate that is part of the spatial geometry of the resource falls within the area specified in the parameter `bbox`. This includes the boundaries of the geometries. For curves the boundary includes the start and end position. For surfaces the boundary includes the outer and inner rings.

In case of a degenerate bounding box, the resulting geometry is used. For example, if the lower left corner is the same as the upper right corner, all resources match where the geometry intersects with this point.

This standard does not specify requirements for the parameter `bbox-crs`. Those requirements will be specified in a later version of this standard.

For WGS 84 longitude/latitude the bounding box is in most cases the sequence of minimum longitude, minimum latitude, maximum longitude and maximum latitude. However, in cases

where the box spans the anti-meridian (180th meridian) the first value (west-most box edge) is larger than the third value (east-most box edge).

Example 1. The bounding box of the New Zealand Exclusive Economic Zone

The bounding box of the New Zealand Exclusive Economic Zone in WGS84 (from 160.6°E to 170°W and from 55.95°S to 25.89°S) would be represented in JSON as `[ 160.6, -55.95, -170, -25.89 ]` and in a query as `bbox=160.6,-55.95,-170,-25.89`.

Note that the server will return an error if a latitude value of `160.0` is used.

If the vertical axis is included, the third and the sixth number are the bottom and the top of the 3-dimensional bounding box.

A template for the definition of the parameter in YAML according to OpenAPI 3.0 is available at bbox.yaml.

9.2.2. Parameter datetime

The `datetime` parameter is defined as follows:

Requirement 43	**/req/geodata/rc-time-definition**

A	The `datetime` parameter SHALL have the following characteristics (using an OpenAPI Specification 3.0 fragment):
	`name: datetime`
	`in: query`
	`required: false`
	`schema:`
	`  type: string`
	`style: form`
	`explode: false`

While the processing of the `datetime` parameter is specific to the resource and operation for which it is applied, there is a general set of requirements which all implementations must address.

Requirement 44	**/req/geodata/rc-time-response**
A	If the `datetime` parameter is provided by the client and supported by the server, then only resources that have a temporal geometry that intersects the temporal information in the `datetime` parameter SHALL be part of the result set.

B	If a resource has multiple temporal properties, the API implementor decides whether only a single temporal property is used to determine the extent or all relevant temporal properties.
C	The `datetime` parameter SHALL match all resources in the collection that are not associated with a temporal geometry.

D	Temporal geometries are either a date-time value or a time interval. The parameter value SHALL conform to the following syntax (using ABNF):
	`interval-closed = date-time "/" date-time`
	`interval-open-start = "../" date-time`
	`interval-open-end = date-time "/.."`
	`interval = interval-closed / interval-open-start / interval-open-end`
	`datetime = date-time / interval`

E	The syntax of `date-time` is specified by RFC 3339, 5.6.
F	Open ranges in time intervals at the start or end SHALL be supported using a double-dot (`..`).

"Intersects" means that the time (instant or period) specified in the parameter `datetime` includes a timestamp that is part of the temporal geometry of the resource (again, a time instant or period). For time periods this includes the start and end time.

Note	ISO 8601-2 distinguishes open start/end timestamps (double-dot) and unknown start/end timestamps (empty string). For queries, an unknown start/end has the same effect as an open start/end.

Example 2. A date-time

February 12, 2018, 23:20:52 GMT:

`time=2018-02-12T23%3A20%3A52Z`

For resources with a temporal property that is a timestamp (like `lastUpdate`), a date-time value would match all resources where the temporal property is identical.

For resources with a temporal property that is a date or a time interval, a date-time value would match all resources where the timestamp is on that day or within the time interval.

Example 3. Intervals

February 12, 2018, 00:00:00 GMT to March 18, 2018, 12:31:12 GMT:

`datetime=2018-02-12T00%3A00%3A00Z%2F2018-03-18T12%3A31%3A12Z`

February 12, 2018, 00:00:00 UTC or later:

`datetime=2018-02-12T00%3A00%3A00Z%2F..`

March 18, 2018, 12:31:12 UTC or earlier:

`datetime=..%2F2018-03-18T12%3A31%3A12Z`

A template for the definition of the parameter in YAML according to OpenAPI 3.0 is available at datetime.yaml.

9.2.3. Parameter limit

The `limit` parameter limits the number of resources that can be returned in a single response.

Requirement 45	**/req/geodata/rc-limit-definition**

A	The operation SHALL support a parameter `limit` with the following characteristics (using an OpenAPI Specification 3.0 fragment):
	`name: limit`
	`in: query`
	`required: false`
	`schema:`
	`  type: integer`
	`  minimum: 1`
	`  maximum: 10000`
	`  default: 10`
	`style: form`
	`explode: false`

Note:	The values for `minimum`, `maximum` and `default` are only examples and MAY be changed.

While the processing of the `limit` parameter is specific to the resource and operation for which it is applied, there is a general set of requirements which all implementations must address.

Requirement 46	**/req/geodata/rc-limit-response**

A	If the `limit` parameter is provided by the client and supported by the server, then the response SHALL not contain more resources than specified by the `limit` parameter.
B	If the API definition specifies a maximum value for the `limit` parameter, the response SHALL not contain more resources than this maximum value.
C	Only items are counted that are on the first level of the collection. Any nested objects contained within the explicitly requested items SHALL not be counted.

The number of resources returned depends on the server and the value of the `limit` parameter.

- The client can request a limit to the number of resources returned.
- The server may have a default value for the limit, and a maximum limit.
- If the server has any more results available than it returns (the number it returns is less than or equal to the requested/default/maximum limit) then the server will include a link to the next set of results.

Permission 6	**/per/geodata/rc-server-limit**
A	If a server is configured with a maximum response size, then the server MAY page responses which exceed that threshold.

Recommendation 16	**/rec/geodata/rc-server-limit**
A	Clients SHOULD be prepared to handle a paged response even if they have not specified a `limit` parameter in their query.

The effect of the limit parameter is to divide the response into a number of pages. Each page (except for the last) contains the specified number of entities. The response contains the first `page`. Additional pages can be accessed through hyperlink navigation.

Recommendation 17	**/rec/geodata/rc-next-1**
A	A `200`-response SHOULD include a link to the next "page" (relation: `next`), if more resources have been selected than returned in the response.

Recommendation 18	**/rec/geodata/rc-next-2**

A	Dereferencing a `next` link SHOULD return additional resources from the set of selected resources that have not yet been returned.

Recommendation 19	**/rec/geodata/rc-next-3**
A	The number of resources in a response to a `next` link SHOULD follow the same rules as for the response to the original query and again include a `next` link, if there are more resources in the selection that have not yet been returned.

Providing `prev` links supports navigating back and forth between pages, but depending on the implementation approach it may be too complex to implement.

Permission 7	**/per/geodata/rc-prev**
A	A response to a `next` link MAY include a `prev` link to the resource that included the `next` link.

Chapter 10. Requirements Class "GeoData-Coverage"

Requirements Class	
http://www.opengis.net/spec/ogcapi-coverages-1/1.0/req/geodata-coverage	
Target type	Web API
Dependency	http://www.opengis.net/spec/ogcapi_common-1/1.0/req/core
Dependency	http://www.opengis.net/spec/ogcapi_common-2/1.0/req/collections

The `GeoData-Coverage` Requirements Class defines the requirements for locating, understanding, and accessing a geospatial data resource as a coverage. The `GeoData-Coverage` Requirements Class is presented in five sections:

1. API Platform: a set of common capabilities
2. Collection Access: operations for accessing collections of `Coverages`
3. Coverage Access: operations for accessing `Coverage` resources
4. Parameters: parameters for use in the API-Coverage operations.
5. General: general principles for use with this standard.

10.1. Resource Requirements

In this clause, API-Common is extended to support `Coverage` resources.

A `Coverage` is a collection of measured values. The structure of that collection is defined by the CIS standard. CIS contains four principle components:

- A `DomainSet` component describing the coverage's domain (the set of "direct positions", i.e., the locations for which values are stored in the coverage)
- A `RangeType` component which describes the coverage's `RangeSet` data structure (in the case of images usually called the "pixel data type").
- A `RangeSet` component containing the stored values (often referred to as "pixels", "voxels") of the coverage.
- A `Metadata` component which represents an extensible slot for metadata. The intended use is to hold any kind of application-specific metadata structures.

A coverage containing the `DomainSet`, `RangeType`, `RangeSet`, and `Metadata` components is accessible at

- The relative path `/coverage`
- The link relation `ogc:coverage:coverage`

The `DomainSet` and `RangeType` are accessible from resources linked by links within the `/collections/{coverageid}` resource, using the relation types `http://www.opengis.net/def/rel/ogc/1.0/coverage-domainset` and `http://www.opengis.net/def/rel/ogc/1.0/coverage-rangetype`,

respectively. Those links may either point to separate resource, in which case the recommended paths are `/collections/{coverageid}/coverage/domainset` and `/collections/{coverageid}/coverage/rangetype`, or point to the coverage description resource itself the domain set and/or range type are embedded. In the latter case those links will point specifically to properties using `#`, e.g., `#domainset` and `#rangetype`.

The `RangeSet` may be available as its own resource as well, if supported by the selected representation (format).

If `Metadata` is available, it will also be be available as its own resource.

The paths discussed in this section are all branches off of the `/collections/{coverageid}` root.

10.1.1. Coverage

The `Coverage` operation returns all the components of the coverage (rangeset, domain set, range type, metadata). It is comparable to a WCS GetCoverage operation.

10.1.1.1. Operation

The `Coverage` operation is defined by the following requirement.

Requirement 47	**/req/geodata-coverage/coverage-op**
A	The API SHALL support the HTTP GET operation at the path `/collections/{coverageid}/coverage`. • `coverageid` is the local identifier for a `Coverage`. It serves the same role and is subject to the same requirements as the `collectionid` parameter defined in API-Common.

10.1.1.2. Response

A successful response to the `Coverage` operation shall meet the following requirement.

Requirement 48	**/req/geodata-coverage/coverage-success**
A	A successful execution of the operation SHALL be reported as a response with a HTTP status code `200`.
B	The content of that response SHALL include the `rangeset` of the coverage equivalent to that defined in the JSON schema coverage.json.
C	The content of that response SHALL also include the `domainset` and `rangetype` as well, to the extent that the negotiated format can describe it.

D	The content of that response MAY also include the `metadata` for that coverage. The server SHOULD return that metadata if it is available and the negotiated format can describe it.
E	The response SHALL be encoded using the format(s) negotiated through the HTTP protocol.
F	If no format is negotiated, then the response SHALL be encoded using the format associated with the media type described in the link object which links to this resource, contained within the coverage (collection) resource.

Coverage Response Schema

```
$schema: http://json-schema.org/draft-07/schema#
title: Coverage object
description: 'Component of OGC Coverage Implementation Schema 1.1. Last updated: 2016-
may-18.
  Copyright (c) 2016 Open Geospatial Consortium, Inc. All Rights Reserved. To obtain
  additional rights of use, visit http://www.opengeospatial.org/legal/.'
type: object
oneOf:
- required:
  - type
  - domainSet
  - rangeSet
  - rangeType
  properties:
    id:
      type: string
    type:
      enum:
      - CoverageByDomainAndRangeType
    envelope:
      "$ref": coverage_envelope.json#/envelope
    domainSet:
      "$ref": coverage_domainset.json#/domainSet
    rangeSet:
      "$ref": coverage_rangeset.json#/rangeSet
    rangeType:
      "$ref": coverage_rangetype.json#/rangeType
    metadata:
      "$ref": coverage_metadata.json#/metadata
- required:
  - type
  - partitionSet
  - rangeType
  properties:
    id:
      type: string
    type:
      enum:
      - CoverageByPartitioningType
    envelope:
      "$ref": coverage_envelope.json#/envelope
    partitionSet:
      "$ref": coverage_partitionset.json#/partitionSet
    rangeType:
      "$ref": coverage_rangetype.json#/rangeType
    metadata:
      "$ref": coverage_metadata.json#/metadata
```

The following JSON fragment is an example of a response to a Coverage request.

Coverage Example

```
{
  "TBD": [
    "filler1",
    "filler2"
  ]
}
```

10.1.1.3. Error situations

The requirements for handling unsuccessful requests are provided in [http-response]. General guidance on HTTP status codes and how they should be handled is provided in HTTP Status Codes.

10.2. Parameter Requirements

The API-Coverages standard inherits basic query and subsetting parameters from API-Common. This section provides a short description of each parameter and identifies the relevant requirements.

All of the permissions and recommendations in API-Common regarding the these parameters also apply to API-Coverages implementations.

10.2.1. Coverage Domain Set

The `Coverage Domain Set` operation returns the coverage's domain set definition

10.2.1.1. Operation

The `Coverage Domain Set` operation is defined by the following requirement.

Requirement 49	**/req/geodata-coverage/domainset-op**
A	The API SHALL include a link with the relation type `http://www.opengis.net/def/rel/ogc/1.0/coverage-domainset` within the links of the coverage (collection) resource, to a resource supporting the HTTP GET operation and returning a CIS JSON representation including a `DomainSet` property. This link can either be a link to a separate resource (e.g., recommended path `/collections/{coverageid}/coverage/domainset`) or a link to the coverage resource itself pointing to the property using RFC 6901 (JSON Pointer) e.g., `#/DomainSet`. • `coverageid` is the local identifier for a `Coverage`. It serves the same role and is subject to the same requirements as the `collectionid` parameter defined in API-Common.

10.2.1.2. Response

A successful response to the `Coverage Domain Set` operation shall meet the following requirement.

Requirement 50	**/req/geodata-coverage/domainset-success**
A	A successful execution of the operation SHALL be reported as a response with a HTTP status code `200`.
B	The content of that response SHALL provide the `Domain Set` definition of the coverage equivalent to that defined in the JSON schema coverage_domainset.json.
C	The response SHALL be encoded using the format(s) negotiated through the HTTP protocol.
D	If no format is negotiated, then the response SHALL be encoded using CIS JSON, which SHALL also be the format associated with the media type described in the link object which links to this resource, contained within the coverage (collection) resource.

Coverage Domain Set Response Schema

```
$schema: http://json-schema.org/draft-07/schema#
domainSet:
  title: domainSet
  description: The domainSet describes the *direct positions* of the coverage, i.e.,
    the locations for which values are available.
  type: object
  oneOf:
  - required:
    - type
    - generalGrid
    properties:
      type:
        enum:
        - DomainSetType
      generalGrid:
        title: General Grid
        description: A general n-D grid is defined through a sequence of axes, each
          of which can be of a particular axis type.
        type: object
        required:
        - type
        additionalProperties: false
        properties:
          type:
            enum:
            - GeneralGridCoverageType
```

```
          id:
            type: string
          srsName:
            type: string
            format: uri
          axisLabels:
            type: array
            items:
              type: string
          axis:
            type: array
            items:
              type: object
              oneOf:
              - title: Index Axis
                description: An Index Axis is an axis with only integer positions
                  allowed.
                required:
                - type
                - axisLabel
                - lowerBound
                - upperBound
                additionalProperties: false
                properties:
                  type:
                    enum:
                    - IndexAxisType
                  id:
                    type: string
                  axisLabel:
                    type: string
                  lowerBound:
                    type: number
                  upperBound:
                    type: number
              - title: Regular Axis
                description: A Regular Axis is an axis where all direct coordinates
                  are at a common distance from its immediate neighbors.
                required:
                - type
                - axisLabel
                - lowerBound
                - upperBound
                - resolution
                - uomLabel
                additionalProperties: false
                properties:
                  type:
                    enum:
                    - RegularAxisType
                  id:
```

```
                  type: string
                axisLabel:
                  type: string
                lowerBound:
                  type:
                  - number
                  - string
                  - 'null'
                  - boolean
                upperBound:
                  type:
                  - number
                  - string
                  - 'null'
                  - boolean
                uomLabel:
                  type: string
                resolution:
                  type: number
            - title: Irregular Axis
              description: An irregular axis enumerates all possible direct position
                coordinates.
              required:
              - type
              - axisLabel
              - uomLabel
              - coordinate
              additionalProperties: false
              properties:
                type:
                  enum:
                  - IrregularAxisType
                id:
                  type: string
                axisLabel:
                  type: string
                uomLabel:
                  type: string
                coordinate:
                  type: array
                  items:
                    type:
                    - number
                    - string
                    - boolean
        displacement:
          title: Displacement
          description: A Displacement is a warped axis nest where points on the
            grid all have their individual direct position coordinates. The
sequenceRule
            element describes linearization order.
```

```
            type: object
            oneOf:
            - required:
              - type
              - axisLabels
              - uomLabels
              - coordinates
              properties:
                type:
                  enum:
                  - DisplacementAxisNestType
                id:
                  type: string
                axisLabel:
                  type: string
                srsName:
                  type: string
                  format: uri
                axisLabels:
                  type: array
                  items:
                    type: string
                uomLabels:
                  type: array
                  items:
                    type: string
                coordinates:
                  type: array
                  items:
                    type: array
                    items:
                      type:
                      - number
                      - string
                      - boolean
            - required:
              - type
              - axisLabels
              - uomLabels
              - coordinatesRef
              properties:
                type:
                  enum:
                  - DisplacementAxisNestTypeRef
                id:
                  type: string
                axisLabel:
                  type: string
                srsName:
                  type: string
                  format: uri
```

```
                axisLabels:
                  type: array
                  items:
                    type: string
                uomLabels:
                  type: array
                  items:
                    type: string
                coordinatesRef:
                  type: string
                  format: uri
          model:
            title: Sensor model
            description: A Transformation By Sensor Model is a transformation
definition
              which is given by a SensorML 2.0 transformation specification.
            type: object
            required:
            - type
            - sensorModelRef
            properties:
              type:
                enum:
                - TransformationBySensorModelType
              id:
                type: string
              axisLabels:
                type: array
                items:
                  type: string
              uomLabels:
                type: array
                items:
                  type: string
              sensorModelRef:
                type: string
                format: uri
              sensorInstanceRef:
                type: string
                format: uri
          gridLimits:
            title: Grid limits
            description: This is the boundary of the array underlying the grid, given
              by its diagonal corner points in integer _60_3D. The grid limits can
              be omitted in case all axes are of type index axis, because then it
              repeats the grid information in a redundant way. The purpose of the
              axisLabels attribute, which lists the axis labels of all axisExtent
              elements in proper sequence, is to enforce axis sequence also in XML
              systems which do not preserve document order.
            type: object
            required:
```

```
            - type
            properties:
              indexAxis:
                title: Index Axis
                description: An Index Axis is an axis with only integer positions
                  allowed.
                type: object
                required:
                - type
                - lowerBound
                - upperBound
                additionalProperties: false
                properties:
                  type:
                    enum:
                    - IndexAxisType
                  id:
                    type: string
                  axisLabel:
                    type: string
                  lowerBound:
                    type: number
                  upperBound:
                    type: number
              srsName:
                type: string
                format: uri
              axisLabels:
                type: array
                items:
                  type: string
  - required:
    - type
    - directMultiPoint
    properties:
      type:
        enum:
        - DomainSetType
      directMultiPoint:
        oneOf:
        - required:
          - type
          - coordinates
          properties:
            type:
              enum:
              - DirectMultiPointType
            coordinates:
              type: array
              items:
                type: array
```

```
                items:
                  type:
                  - number
                  - string
                  - boolean
        - required:
          - type
          - coordinatesRef
          properties:
            type:
              enum:
              - DirectMultiPointTypeRef
            coordinatesRef:
              type: string
              format: uri
  - required:
    - type
    - fileReference
    properties:
      type:
        enum:
        - DomainSetRefType
      id:
        type: string
        format: uri
      fileReference:
        type: string
        format: uri
```

The following JSON fragment is an example of a response to a Coverage DomainSet request.

Coverage DomainSet Example

```
{
  "TBD": [
    "filler1",
    "filler2"
  ]
}
```

10.2.1.3. Error situations

The requirements for handling unsuccessful requests are provided in [http-response]. General guidance on HTTP status codes and how they should be handled is provided in HTTP Status Codes.

10.2.2. Coverage Range Type

The `Coverage Range Type` operation returns the coverage's range type information (i.e., a description of the data semantics)

10.2.2.1. Operation

The `Coverage Range Type` operation is defined by the following requirement.

Requirement 51	**/req/geodata-coverage/rangetype-op**
A	The API SHALL include a link with the relation type `http://www.opengis.net/def/rel/ogc/1.0/coverage-rangetype` within the links of the coverage (collection) resource, to a resource supporting the HTTP GET operation and returning a CIS JSON representation including a `RangeType` property. This link can either be a link to a separate resource (e.g., recommended path `/collections/{coverageid}/coverage/rangetype`) or a link to the coverage resource itself pointing to the property using RFC 6901 (JSON Pointer) e.g., `#/RangeType`. • `coverageid` is the local identifier for a `Coverage`. It serves the same role and is subject to the same requirements as the `collectionid` parameter defined in API-Common.

10.2.2.2. Response

A successful response to the `Coverage Range Type` operation shall meet the following requirement.

Requirement 52	**/req/geodata-coverage/rangetype-success**
A	A successful execution of the operation SHALL be reported as a response with a HTTP status code `200`.
B	The content of that response SHALL describe the `Range Type` of the coverage equivalent to that defined in the JSON schema coverage_rangetype.json.
C	The response SHALL be encoded using the format(s) negotiated through the HTTP protocol.
D	If no format is negotiated, then the response SHALL be encoded using CIS JSON, which SHALL also be the format associated with the media type described in the link object which links to this resource, contained within the coverage (collection) resource.

Coverage Range Type Response Schema

```
$schema: http://json-schema.org/draft-07/schema#
rangeType:
  title: rangeType
  description: The rangeType element describes the structure and semantics of a
```

```
coverage's
    range values, including (optionally) restrictions on the interpolation allowed
    on such values.
  type: object
  oneOf:
  - required:
    - type
    - field
    properties:
      type:
        enum:
        - DataRecordType
      field:
        type: array
        items:
          title: quantity
          description: quantity
          type: object
          required:
          - type
          properties:
            type:
              enum:
              - QuantityType
            id:
              type: string
              format: uri
            name:
              type: string
            definition:
              type: string
              format: uri
            uom:
              title: units of measure
              description: units of measure
              type: object
              required:
              - type
              - code
              properties:
                type:
                  enum:
                  - UnitReference
                id:
                  type: string
                  format: uri
                code:
                  type: string
            constraint:
              title: Constraint
              description: Constraint
```

```
              type: object
              required:
              - type
              properties:
                type:
                  enum:
                  - AllowedValues
                id:
                  type: string
                  format: uri
                interval:
                  type: array
                  items:
                    type:
                    - number
                    - string
                    - boolean
      interpolationRestriction:
        title: interpolationRestriction
        description: Interpolation restriction
        type: object
        required:
        - type
        properties:
          type:
            enum:
            - InterpolationRestrictionType
          id:
            type: string
            format: uri
          allowedInterpolation:
            type: array
            items:
              type: string
              format: uri
  - required:
    - type
    - fileReference
    properties:
      type:
        enum:
        - RangeTypeRefType
      id:
        type: string
        format: uri
      fileReference:
        type: string
        format: uri
```

The following JSON fragment is an example of a response to a Coverage RangeType request.

Coverage RangeType Example

```
{
  "TBD": [
    "filler1",
    "filler2"
  ]
}
```

10.2.2.3. Error situations

The requirements for handling unsuccessful requests are provided in [http-response]. General guidance on HTTP status codes and how they should be handled is provided in HTTP Status Codes.

10.2.3. Coverage Range Set

The `Coverage Range Set` operation returns the coverage's range set, i.e., the actual values in the coverage's Native Format (see Media Types for ways to retrieve inspecific formats)

10.2.3.1. Operation

The `Coverage Range Set` operation is defined by the following requirement.

Requirement 53	**/req/geodata_coverage/rangeset-op**
A	The API MAY support the HTTP GET operation at the path `/collections/{coverageid}/coverage/rangeset`.
B	The API SHOULD support it for selected formats which can describe the raw data values without any additional information. • `coverageid` is the local identifier for a `Coverage`. It serves the same role and is subject to the same requirements as the `collectionid` parameter defined in API-Common.

10.2.3.2. Response

A successful response to the `Coverage Range Set` operation shall meet the following requirement.

Requirement 54	**/req/geodata_coverage/rangeset-success**
A	A successful execution of the operation SHALL be reported as a response with a HTTP status code `200`.

B	The content of that response SHALL contain the `Range Set` of the coverage equivalent to that defined in the JSON schema coverage_rangeset.json.
C	The response SHALL be encoded using the format(s) negotiated through the HTTP protocol.
D	If no format is negotiated, then the response SHALL be encoded using the format associated with the media type described in the link object which links to this resource, contained within the coverage (collection) resource.

Coverage Range Set Response Schema

```
$schema: http://json-schema.org/draft-07/schema#
rangeSet:
  title: rangeSet
  description: 'The rangeSet lists a value for each of the coverage''s direct
positions.
    Values resemble the *payload* information of some particular direct positions.
    Values can be composite (with a single nesting level, i.e.: composites always
    consist of atomics), or atomic (emulated through single-component composites)
    whereby the sequence, structure, and meaning of every value is defined through
    the rangeType. Values can be represented in-line or by reference to an external
    file which may have any suitable encoding.'
  type: object
  oneOf:
  - required:
    - type
    - dataBlock
    properties:
      type:
        enum:
        - RangeSetType
      dataBlock:
        title: dataBlock
        description: Data block objects
        type: object
        required:
        - type
        - values
        properties:
          type:
            enum:
            - VDataBlockType
            - CVDataBlockType
          values:
            type: array
            items:
```

```
              type:
              - number
              - string
              - 'null'
              - boolean
  - required:
    - type
    - fileReference
    properties:
      type:
        enum:
        - RangeSetRefType
      fileReference:
        type: array
        items:
          type: string
          format: uri
```

The following JSON fragment is an example of a response to a Coverage RangeSet request.

Coverage RangeSet Example

```
{
  "TBD": [
    "filler1",
    "filler2"
  ]
}
```

10.2.3.3. Error situations

The requirements for handling unsuccessful requests are provided in [http-response]. General guidance on HTTP status codes and how they should be handled is provided in HTTP Status Codes.

10.2.4. Coverage Metadata

The `Coverage Metadata` operation returns the coverage's metadata (may be empty)

10.2.4.1. Operation

The `Coverage Metadata` operation is defined by the following requirement.

Requirement 55	**/req/geodata-coverage/metadata-op**
A	The API MAY support the HTTP GET operation at the path `/collections/{coverageid}/coverage/metadata`.

B	The API SHOULD support it, if available, for selected formats which can describe it. • `coverageid` is the local identifier for a `Coverage`. It serves the same role and is subject to the same requirements as the `collectionid` parameter defined in API-Common.

10.2.4.2. Response

A successful response to the `Coverage Metadata` operation shall meet the following requirement.

Requirement 56	**/req/geodata-coverage/metadata-success**
A	A successful execution of the operation SHALL be reported as a response with a HTTP status code `200`.
B	The content of that response SHALL contain metadata which describes the coverage equivalent to that defined in the JSON schema coverage_metadata.json.
C	The response SHALL be encoded using the format(s) negotiated through the HTTP protocol.
D	If no format is negotiated, then the response SHALL be encoded using the format associated with the media type described in the link object which links to this resource, contained within the coverage (collection) resource.

Coverage Metadata Response Schema

```
$schema: http://json-schema.org/draft-07/schema#
metadata:
  description: The metadata element is a container of any (not further specified)
    information which should be transported along with the coverage on hand, such
    as domain-specific metadata.
  type: object
  properties: {}
```

The following JSON fragment is an example of a response to a Coverage Metadata request.

Coverage Metadata Example

```
{
  "TBD": [
    "filler1",
    "filler2"
  ]
}
```

10.2.4.3. Error situations

The requirements for handling unsuccessful requests are provided in [http-response]. General guidance on HTTP status codes and how they should be handled is provided in HTTP Status Codes.

Chapter 11. Media Types

This standard does not mandate any particular encoding or format. However, it does provide extensions for encodings which are commonly used in OGC APIs. These extensions include:

- CIS JSON
- HTML

Neither of these encodings are mandatory. An implementor of this standard may choose to implement neither of them, selecting different encodings instead.

In addition to the Requirements Classes, there are additional coverage formats which implementors should be aware of. These formats apply to encodings of pixel data. Since this data is typically binary, it is largely opaque to the API.

11.1. HTML Encoding

Support for HTML is recommended. HTML is the core language of the World Wide Web. An API that supports HTML will support browsing the spatial resources with a web browser and will also enable search engines to crawl and index those resources.

11.2. CIS JSON Encoding

Support for CIS JSON is recommended. JSON is a commonly used format that is simple to understand and well supported by tools and software libraries.

JSON structures documented in this standard are defined using JSON Schema. These schema are available in JSON and YAML formats from http://schemas.opengis.net/tbd

When coverage resources (the coverage as a whole, range set, domain set, range type, meta data) advertise the `application/json` media type, it refers to the CIS JSON encoding.

This API-Coverages standard is built around the OGC Coverage Implementation Schema (CIS). CIS content often includes multi-dimensional coordinates and coordinate reference systems in sensor and analytic space. These "Engineering" coordinate reference systems cannot be represented using WGS-84.

The OGC JSON Schema for CIS standard addresses that need by defining a JSON schema for the CIS standard. This format should be used when the `application/json` media type is specified to encode all of the `{datasetAPI}/collections/{coverageid}/coverage*` resources. A CIS JSON representation of the range type and domain set for the coverage is also required to be either embedded in properties or linked from the /collections/{collectionId} resource.

Other encodings such as CoverageJSON must use an alternate media type.

11.3. Binary

A coverage can also be distributed in binary form, for which a number of formats are commonly used:

- CIS RDF
- NetCDF
- GeoTIFF
- PNG
- JPEG
- JPEG-2000

OGC CIS also defines multipart encoding of the different components of a coverage. This allows the result to have a "canonical" header while components can be factored out and represented in some (more efficient) binary format. Any suitable container format (such as zip, multipart/mime, SAFE, etc.) can "bundle" these components into one coverage file ready for shipping.

With OGC API - Coverages, given that the description of the coverage is easily available at the /collections/{collectionId} resource, with the range type and domain set either embedded or linked from that resource, an implementation may prefer to keep single-part binary formats encoding all components to the best it can, as they might be more directly interoperable.

11.4. Media Types

A description of the media types is mandatory for any OGC standard which involves data encodings. The list of suitable media types for the API-Coverages standard in provided in Table 8.

Coverages can be encoded in any suitable data format, including formats as GML, CIS JSON, GeoTIFF, and NetCDF. Further, coverages can be represented by a single document (stream or file) or by a hierarchically organized set of documents, each of which can be encoded individually – for example, the domain set, range type, and metadata may be encoded in easily parseable GML, CIS JSON, or RDF while the range set is encoded in some compact binary format like NetCDF or JPEG2000. Such partitioning allows for coverages tiled in space, time, or mixed, thereby enabling mosaics, time-interleaved coverages, and efficiently subsettable datacubes.

Table 8. API-Coverages media Types

Encoding	media type
HTML	text/html
JSON / CIS JSON	application/json
CIS RDF	TODO:
CoverageJSON	TODO:
GeoTIFF	image/tiff; application=geotiff
PNG	image/png

JPEG	image/jpeg
JPEG-2000	image/jp2
NetCDF	application/x-netcdf

NOTE Consider adding a table showing valid media types for each CIS component.

11.5. Default Encodings

The media type used to encode a response to a request shall be determined through the HTTP content negotiation protocol as specified in API-Common. If not using content negotiation, the encoding must follow the media type described in the link to the resource from the collection.

Chapter 12. Requirements Class Coverage Subset

The Subset Requirements Class defines parameters for sub-setting n-dimensional Range Sets. Subsetting parameters may be mixed with other parameters, in no particular order, in the query part of a URL.

Requirements Class	
http://www.opengis.net/spec/ogcapi-coverages-1/1.0/req/coverage-subset	
Target type	Web API
Dependency	http://www.opengis.net/spec/ogcapi_coverage-1/1.0/req/geodata-coverage
Dependency	OGC Coverage Implementation Schema (CIS)

The subset parameter is defined in the following Requirement:

Requirement 57	**/req/coverage-subset/definition**

A	The operation SHALL support a parameter `subset` with the following characteristics (using an Extended Backus Naur Form (EBNF) fragment):
	`SubsetSpec: "subset"=axisName(intervalOrPoint)`
	`axisName: {text}`
	`intervalOrPoint: interval | point`
	`interval: low : high`
	`low: point | *`
	`high: point | *`
	`point: {number} | "{text}"`
	`Where:`
	`\" = double quote = ASCII code 0x42,`
	`{number} is an integer or floating-point number, and`
	`{text} is some general ASCII text (such as a time and date notation in ISO 8601).`

B	The axis name SHALL be the same as one of the `axisLabels` defined in the DomainSet or else return a 400 status code.
C	If the intervalOrPoint values fall entirely outside the range of valid values defined by the DomainSet for the identified axis,a 204 status code SHALL be returned

D	For a CRS where an axis can wrap around, such as subsetting across the dateline (anti-meridian) in a geographic CRS, a `low` value greater than `high` SHALL be supported to indicate an extent crossing that wrapping point.

NOTE When the intervalOrPoint values fall partially outside of the range of valid values defined by the DomainSet for the identified axis, the service is expected to return the non-empty portion of the coverage resulting from the subset. For subsetting on the range set, and for coverage media types with no geo-referencing mechanisms (e.g. PNG), NO_DATA values or transparency should be used. If a georeferencing mechanism is available within the negotiated media type, the service could decide whether to use NO_DATA values or simply return the properly geo-referenced values within the domain set.

The following permission is granted to facilitate slicing data sparsely populated across a dimension, such as Earth Observation imagery:

Permission 8	**/per/coverage-subset/slice-sparse-dimension**
A	The empty portions in a coverage resulting from a slice operation on an axis (e.g. time), combined with a trimming operations on other axes (e.g. latitude and longitude) which would either be empty or not cover the full extent of the trim operation MAY be filled with data values from the same trim operation combined with a slicing operation on a different value of the slicing axis which would return non-empty values. For example, the closest or last previous time for which data is available for a certain geospatial extent may be returned. An Earth Observation use case for this permission is to allow retrieving a slice of the last available imagery on or before a certain date, taking into account that a certain geographic area may only be observed every few days.
B	This permission applies to both explicit slice operations using subset, as well as to implicit slicing from requesting an output format only supporting a lower dimensionality than the data (e.g. requesting a 2D image from a 3D coverage as PNG or GeoTIFF).
C	A query parameter defined by a custom or standardized extension MAY be made available to enable, disable or alter that behavior.

The results of using a subset parameter are defined by the following Requirement:

Requirement 58	/req/coverage-subset/success
A	The `subset` parameter SHALL be applied to the coverage (and rangeset, if available as a separate resource).
B	Only that part of the coverage addressed SHALL be returned that falls within the bounds of the subset expression whereby a * (asterisk) in the position of a lower or upper limit along an axis denotes the coverage's minimum or maximum extent along this axis, respectively. The DomainSet shall be adjusted accordingly to the new boundaries (in case of trimming) and the reduced dimension (in case of slicing).

12.1. Subsetting Examples

These examples return a subset of a coverage in the negotiated format, including domain set, range type (and metadata if applicable) to the extent than they can be described in that format:

- {datasetAPI}/collections/{coverageid}/coverage?subset=Lat(40:50),Lon(10:20) — returns a cutout (trim) from the coverage for the extent between corner coordinates (Lat: 40, Lon: 10) and (Lat: 50, Lon: 20)
- {datasetAPI}/collections/{coverageid}/coverage?subset=time("2019-03-27") — returns a coverage slice at the timestamp (if the coverage is has 2D spatial dimensions plus time, the result will be a 2D image with its full spatial extent)

A `rangeset` resource, also supporting subsetting, may be available in some implementations, for some formats/media types. The following examples using this resource return only the sample values for a subset of a coverage in the negotiated format, without domain set, range type or metadata:

- {datasetAPI}/collections/{coverageid}/coverage/rangeset?subset=Lat(40:50),Lon(10:20) — returns a coverage cutout/trim between (Lat: 40, Lon: 10) and (Lat: 50, Lon: 20)
- {datasetAPI}/collections/{coverageid}/coverage/rangeset?subset=time("2019-03-27") — returns a coverage slice at the timestamp given, with all data available in the other dimensions (if the coverage is has 2D spatial dimensions plus time, the result will be a 2D image with its full spatial extent)

Chapter 13. Requirements Class Coverage Scaling

The Scaling Requirements Class defines parameters for retrieving data from n-dimensional Range Sets at a resolution different than the original. Scaling parameters may be mixed with other parameters, in no particular order, in the query part of a URL.

Requirements Class	
http://www.opengis.net/spec/ogcapi-coverages-1/1.0/req/coverage-scaling	
Target type	Web API
Dependency	http://www.opengis.net/spec/ogcapi_coverage-1/1.0/req/geodata-coverage
Dependency	OGC Coverage Implementation Schema (CIS)

The scaling parameters are defined in the following Requirement:

Requirement 59	**/req/coverage-scaling/definition**

A	The operation SHALL support a parameter `scale-size` with the following characteristics (using an Extended Backus Naur Form (EBNF) fragment):
	`ScalingSpec: "scale-size"=axisName({number})[,axisName({number})]*`
	`axisName: {text}`
	Where:
	{number} is an integer or floating-point number, and

B	The operation SHALL support `scale-factor` numeric parameter.

C	The operation SHALL support a parameter `scale-axes` with the following characteristics (using an Extended Backus Naur Form (EBNF) fragment):
	
	`ScalingSpec: "scale-axes"=axisName({number})[,axisName({number})]*`
	`axisName: {NCName}`
	
	`Where:`
	`{number} is an integer or floating-point number, and`

D	The axis name SHALL be the same as one of the `axisLabels` defined in the DomainSet, and shall match the abbreviations defined by the CRS (as defined bythe <gml:axisAbbrev> tags). or else return a 400 status code.

The results of using a scaling parameter are defined by the following Requirement:

Requirement 60	**/req/coverage-scaling/success**
A	The `scale-size=AxisName(count)`, `scale-factor=factor` and `scale-axes=AxisName(factor)` parameters SHALL be supported for the coverage and range set resources.
B	When `scale-size` is used, the returned coverage SHALL contain exactly the specified number of sample values along each axis which is specified, and the original number of sample values for unspecified axes.
C	When `scale-factor` is used, for each axis, the returned coverage SHALL contain the number of original sampled values, divided by *factor*.
D	When `scale-axes` is used, for each axis specified, the returned coverage SHALL contain the number of original sampled values, divided by the *factor* specified for that axis, and the original number of sample values for unspecified axes.
E	When `scale-size` is used together with subsetting, the requested number of samples SHALL be mapped to the subset dimensions specified, even if those dimensions fall partially outside the extent of the coverage.

13.1. Scaling Examples

- {datasetAPI}/collections/{coverageid}/coverage?scale-size=Lat(300),Lon(400) — returns the entire coverage re-scaled to 300 pixels along its latitude axis, and 400 samples along its longitude axis in the negotiated format
- {datasetAPI}/collections/{coverageid}/coverage/rangeset?subset=Lat(40:50),Lon(10:20)&scale-size=Lat(300),Lon(400) — returns a coverage cutout (sample values only) between (Lat: 40, Lon: 10) and (Lat: 50, Lon: 20), re-scaled to 300 pixels along its latitude axis, and 400 samples along its longitude axis in the negotiated format
- {datasetAPI}/collections/{coverageid}/coverage?scale-factor=1.5 — returns a coverage re-scaled so as to contain 1.5 times less sample values along all of its axes
- {datasetAPI}/collections/{coverageid}/coverage?scale-axes=Lat(2) — returns a coverage re-scaled so as to contain 2 times less sample values along its latitude axis, and all original values along all its other dimensions

Chapter 14. Requirements Class Coverage Range Subset

The Range Subset Requirements Class defines parameters for selecting a subset of the bands (defined in the Range Type) to retrieve from Range Sets. Range subsetting parameters may be mixed with other parameters, in no particular order, in the query part of a URL.

Requirements Class	
http://www.opengis.net/spec/ogcapi-coverages-1/1.0/req/coverage-rangesubset	
Target type	Web API
Dependency	http://www.opengis.net/spec/ogcapi_coverage-1/1.0/req/geodata-coverage
Dependency	OGC Coverage Implementation Schema (CIS)

The range-subset parameter is defined in the following Requirement:

Requirement 61	**/req/coverage-rangesubset/definition**

A	The operation SHALL support a parameter `range-subset` with the following characteristics (using an Extended Backus Naur Form (EBNF) fragment):
	`RangeSubsetSpec: "range-subset"=band[,bandName]*`
	`band: {bandName}|{bandIndex}|"*"`
	`bandName: {text}`
	`bandIndex: {number}`
	
	`Where:`
	`{number} is an integer number, and`
	`{text} is some general ASCII text.`

B	The band name SHALL be the same as one of the `id` defined in the RangeType DataRecord fields, or else return a 400 status code.
C	The band index SHALL be an integer between 0 and the number of bands - 1 defined in the RangeType DataRecord fields, or else return a 400 status code.
D	If a valid band index conflicts with a band name, it SHALL be interpreted as a band name.

The results of using a range-subset parameter are defined by the following Requirement:

Requirement 62	/req/coverage-rangesubset/success
A	The `range-subset` parameter SHALL be applied to the coverage (and rangeset, if available as a separate resource).
B	Only the bands of the coverage addressed SHALL be returned that correspond to one of the bands selected by the range subset expression whereby a * (asterisk) at the begining or at the end of the comma-separated list denotes all other bands before or after the listed bands, respectively. The RangeType shall be adjusted accordingly to the new selected bands.

14.1. Range Subsetting Examples

These examples return a range subset of a coverage in the negotiated format, including domain set, range type (and metadata if applicable) to the extent than they can be described in that format:

- {datasetAPI}/collections/{coverageid}/coverage?range-subset=B02,B03,B04 — returns only the bands with IDs B02, B03 and B04 from the coverage
- {datasetAPI}/collections/{coverageid}/coverage?range-subset=3,4,5 — returns the 4th, 5th and 6th band (0-based indexing, as listed in the Range Type) from the coverage
- {datasetAPI}/collections/{coverageid}/coverage?range-subset=B07,* — returns the band B07 and all subsequent bands from the coverage

A `rangeset` resource, also supporting subsetting, may be available in some implementations, for some formats/media types. The following examples using this resource return only the sample values for a subset of a coverage in the negotiated format, without domain set, range type or metadata:

- {datasetAPI}/collections/{coverageid}/rangeset?range-subset=B02,B03,B04 — returns only the bands with IDs B02, B03 and B04 from the coverage
- {datasetAPI}/collections/{coverageid}/rangeset?range-subset=3,4,5 — returns the 4th, 5th and 6th band (0-based indexing, as listed in the Range Type) from the coverage
- {datasetAPI}/collections/{coverageid}/rangeset?range-subset=B07,* — returns the band B07 and all subsequent bands from the coverage

Chapter 15. Requirements Class Coverage Tiles

The Tiles Requirements Class defines how to combine the OGC API - Tiles building blocks with the Coverages API to request coverage tiles. Coverages parameters such as for subsetting additional dimensions not covered by the Tiles 2D Tile Matrix Set (such as time or elevation), or for range subsetting, can be used together with the path for requesting individual tiles.

Requirements Class	
http://www.opengis.net/spec/ogcapi-coverages-1/1.0/req/coverage-tiles	
Target type	Web API
Dependency	http://www.opengis.net/spec/ogcapi_coverage-1/1.0/req/geodata-coverage
Dependency	http://www.opengis.net/spec/ogcapi-tiles-1/1.0/conf/core
Dependency	http://www.opengis.net/spec/ogcapi-tiles-1/1.0/conf/tileset
Dependency	http://www.opengis.net/spec/ogcapi-tiles-1/1.0/conf/tilesets-list
Dependency	http://www.opengis.net/spec/ogcapi-tiles-1/1.0/conf/geodata-tilesets
Dependency	OGC Coverage Implementation Schema (CIS)

The capability to request tiles is defined in the following Requirement:

Requirement 63	**/req/coverage-tiles/definition**
A	The coverage SHALL have an associated list of at least one coverage tilesets accessible at `…/{collectionId}/coverage/tiles`, conforming to OGC API - Tiles - Part 1: Core
B	The coverage's collection description document at `…/{collectionId}` SHALL include a link with relation type http://www.opengis.net/def/rel/ogc/1.0/tilesets-coverage linking to that resource listing available coverage tilesets
C	This list of coverage tilesets SHALL include a link to the full descriptions of these tilesets accessible at `…/{collectionId}/coverage/tiles/{tileMatrixSetId}`
D	The tiles making up the tilesets SHALL be accessible from `…/{collectionId}/coverage/tiles/{tileMatrixSetId}/{tileMatrix}/{tileRow}/{tileCol}`

The results of requesting coverage tiles are defined by the following Requirement:

Requirement 64	**/req/coverage-tiles/success**
A	The responses for the list of tilesets, tileset metadata and tiles SHALL comply to all requirements of the OGC API - Tiles specifications
B	The response to a request for an individual tile SHALL return a subset of the coverage trimmed on the axes defined by the 2D Tile Matrix Set to cover the exact geospatial extent of the tile.
C	The response to a request for an individual tile SHALL be scaled down to a number of cells corresponding to the tile's pixel size, plus one if the coverage cells do not fully span the distance between them.

As an example, considering that the *WorldCRS84Quad* Tile Matrix Set has tile pixel sizes defined as 256 x 256, a request for tile
`…/coverage/tiles/WorldCRS84Quad/1/0/0` of a point coverage is equivalent to:

`…/coverage?subset=Lat(0:90),Lon(-180:-90)&scale-size=257,257`

and returns a tile containing one or more slices of 257 x 257 cells, while for an area coverage it is equivalent to:

`…/coverage?subset=Lat(0:90),Lon(-180:-90)&scale-size=256,256`

and returns a tile containing one or more slices of 256 x 256 cells.

15.1. Coverage Tiles Examples

Chapter 16. Requirements Class DomainSet Subset

The DomainSet Subset Requirements Class defines the use of parameters for sub-setting the domainset of the coverage.

Requirements Class	
http://www.opengis.net/spec/ogcapi-coverages-1/1.0/req/domainset-subset	
Target type	Web API
Dependency	http://www.opengis.net/spec/ogcapi_coverage-1/1.0/req/geodata-coverage
Dependency	OGC Coverage Implementation Schema (CIS)

The subset parameter is defined in the following Requirement:

Requirement 65	**/req/domainset-subset/definition**

A	The operation SHALL support a parameter `subset` with the following characteristics (using an Extended Backus Naur Form (EBNF) fragment):
	
	`SubsetSpec: "subset"=axisName(intervalOrPoint)`
	`axisName: {text}`
	`intervalOrPoint: interval | point`
	`interval: low : high`
	`low: point | *`
	`high: point | *`
	`point: {number} | "{text}"`
	
	`Where:`
	`\" = double quote = ASCII code 0x42,`
	`{number} is an integer or floating-point number, and`
	`{text} is some general ASCII text (such as a time and date notation in ISO 8601).`

B	The axis name SHALL be the same as one of the axes defined in the coverage's envelope as part of its collection description document, or else the server SHALL return a 400 status code.
C	If the intervalOrPoint values fall entirely outside the range of the DomainSet for the identified axis, a 204 status code SHALL be returned

D	For a CRS where an axis can wrap around, such as subsetting across the dateline (anti-meridian) in a geographic CRS, a `low` value greater than `high` SHALL be supported to indicate an extent crossing that wrapping point.

The results of using a subset parameter are defined by the following Requirement:

Requirement 66	**/req/domainset-subset/success**
A	The `subset` parameter SHALL be applied to the domainset
B	Only that part of the domainset addressed SHALL be returned that falls within the bounds of the subset expression whereby a * (asterisk) in the position of a lower or upper limit along an axis denotes the DomainSet's minimum or maximum extent along this axis, respectively. The DomainSet shall be adjusted accordingly to the new boundaries (in case of trimming) and the reduced dimension (in case of slicing).

16.1. DomainSet Subsetting Examples

Chapter 17. Requirements Class HTML

The following requirements apply to an OGC API-Coverage implementation when the following conditions apply:

1. The API advertises conformance to the HTML Conformance Class
2. The client negotiates an HTML format

The HTML Requirements Class restricts requirements defined in the `GeoData-Coverage` Requirements Class by imposing encoding-specific requirements. At this time, these additional requirements only apply to the HTTP response payloads. The sections below identify the scope of each new requirement and the `GeoData-Coverage` requirements which lay within each scope.

Requirements Class	
http://www.opengis.net/spec/ogcapi-coverages-1/1.0/req/html	
Target type	Web API
Dependency	Conformance Class "Core"
Dependency	API-Common HTML
Dependency	HTML5
Dependency	Schema.org

17.1. Common

This section covers the requirements inherited from the API-Common standard. Its scope includes responses for the following operations:

- `{datasetAPI}/`: Dataset API Landing Page
- `{datasetAPI}/api`: API Description
- `{datasetAPI}/conformance`: Conformance Classes
- `{datasetAPI}/collections`: Collections
- `{datasetAPI}/collections/{coverageid}`: Collection Information

Requirement 67	**/req/html/api-common**
Extends	/req/core/api-common
The API SHALL demonstrate conformance with the following Requirements Class of the OGC API-Common version 1.0 Standard.	
A	http://www.opengis.net/spec/ogcapi_common-1/1.0/req/html

It is also necessary to advertise conformance with this Requirements Class.

Requirement 68	/req/html/conformance
The list of Conformance Classes advertised by the API SHALL include:	
A	http://www.opengis.net/spec/ogcapi-coverages-1/1.0/conf/html

17.2. Coverage

This section covers the Coverage response for the `{datasetAPI}/collections/{coverageid}/coverage` operation.

Requirement 69	/req/html/coverage-success
Restricts	/req/core/coverage-success
A	A successful execution of the operation SHALL be reported as a response with a HTTP status code `200`.
B	The response SHALL be a valid HTML document
C	The response SHALL include content equivalent to that defined in JSON schema coverage.json.

17.3. Coverage Domain Set

This section covers the Coverage Domain Set response for the `{datasetAPI}/collections/{coverageid}/coverage/domainset` operation.

Requirement 70	/req/html/domainset-success
Restricts	/req/core/domainset-success
A	A successful execution of the operation SHALL be reported as a response with a HTTP status code `200`.
B	The response SHALL be a valid HTML document
C	The response SHALL include content equivalent to that defined in JSON schema coverage_domainset.json.

17.4. Coverage Range Type

This section covers the Coverage Range Type response for the

{datasetAPI}/collections/{coverageid}/coverage/rangetype operation.

Requirement 71	/req/html/rangetype-success
Restricts	/req/core/rangetype-success
A	A successful execution of the operation SHALL be reported as a response with a HTTP status code 200.
B	The response SHALL be a valid HTML document
C	The response SHALL include content equivalent to that defined in JSON schema coverage_rangetype.json.

17.5. Coverage Range Set

This section covers the Coverage Range Set response for the {datasetAPI}/collections/{coverageid}/coverage/rangeset operation.

Requirement 72	/req/html/rangeset-success
Restricts	/req/core/rangeset-success
A	A successful execution of the operation SHALL be reported as a response with a HTTP status code 200.
B	The response SHALL be a valid HTML document
C	The response SHALL include content equivalent to that defined in JSON schema coverage_rangeset.json.

17.6. Coverage Metadata

This section covers the Coverage Metadata response for the {datasetAPI}/collections/{coverageid}/coverage/metadata operation.

Requirement 73	/req/html/metadata-success
Restricts	/req/core/metadata-success
A	A successful execution of the operation SHALL be reported as a response with a HTTP status code 200.

B	The response SHALL be a valid HTML document
C	The response SHALL include content equivalent to that defined in JSON schema coverage_metadata.json.

Chapter 18. Requirements Class CIS JSON

The following requirements apply to an OGC API-Coverage implementation when the following conditions apply:

1. The API advertises conformance to the CIS JSON Conformance Class
2. The client negotiates a JSON or CIS JSON format

The CIS JSON Requirements Class restricts requirements defined in the `GeoData-Coverage` Requirements Class by imposing encoding-specific requirements. At this time, these additional requirements only apply to the HTTP response payloads. The sections below identify the scope of each new requirement and the `GeoData-Coverage` requirements which lay within each scope.

Requirements Class	
http://www.opengis.net/spec/ogcapi_coverages-1/1.0/req/cisjson	
Target type	Web API
Dependency	Requirements Class "API-Common Core"
Dependency	API-Common JSON
Pre-conditions	1) The API advertises conformance to the CIS JSON Conformance Class 2) The client negotiates use of the JSON or CIS JSON encoding.

18.1. Common

This section covers the requirements inherited from the API-Common standard. Its scope includes responses for the following operations:

- `{datasetAPI}/`: Dataset API Landing Page
- `{datasetAPI}/api`: API Description
- `{datasetAPI}/conformance`: Conformance Classes
- `{datasetAPI}/collections`: Collections
- `{datasetAPI}/collections/{coverageid}`: Collection Information

Requirement 74	**/req/cisjson/api-common**
Extends	/req/core/api-common
The API SHALL demonstrate conformance with the following Requirements Class of the OGC API-Common version 1.0 Standard.	
A	http://www.opengis.net/spec/ogcapi_common-1/1.0/req/json

It is also necessary to advertise conformance with this Requirements Class.

Requirement 75	/req/cisjson/conformance
The list of Conformance Classes advertised by the API SHALL include:	
A	http://www.opengis.net/spec/ogcapi-coverages-1/1.0/conf/cisjson

18.2. Coverage

This section covers the Coverage response for the `{datasetAPI}/collections/{coverageid}/coverage` operation.

Requirement 76	/req/cisjson/coverage-success
Restricts	/req/core/coverage-success
A	A successful execution of the operation SHALL be reported as a response with a HTTP status code `200`.
B	The response SHALL be a CIS JSON document which validates against the JSON schema coverage.json.

18.3. Coverage Domain Set

This section covers the Coverage Domain Set response for the `{datasetAPI}/collections/{coverageid}/coverage/domainset` operation.

Requirement 77	/req/cisjson/domainset-success
Restricts	/req/core/domainset-success
A	A successful execution of the operation SHALL be reported as a response with a HTTP status code `200`.
B	The response SHALL be a CIS JSON document which validates against the JSON schema coverage_domainset.json.

18.4. Coverage Range Type

This section covers the Coverage Range Type response for the `{datasetAPI}/collections/{coverageid}/coverage/rangetype` operation.

Requirement 78	/req/cisjson/rangetype-success

Restricts	/req/core/rangetype-success
A	A successful execution of the operation SHALL be reported as a response with a HTTP status code `200`.
B	The response SHALL be a CIS JSON document which validates against the JSON schema coverage_rangetype.json.

18.5. Coverage Range Set

This section covers the Coverage Range Set response for the `{datasetAPI}/collections/{coverageid}/coverage/rangeset` operation.

Requirement 79	**/req/cisjson/rangeset-success**
Restricts	/req/core/rangeset-success
A	A successful execution of the operation SHALL be reported as a response with a HTTP status code `200`.
B	The response SHALL be a CIS JSON document which validates against the JSON schema coverage_rangeset.json.

18.6. Coverage Metadata

This section covers the Coverage Metadata response for the `{datasetAPI}/collections/{coverageid}/coverage/metadata` operation.

Requirement 80	**/req/cisjson/metadata-success**
Restricts	/req/core/metadata-success
A	A successful execution of the operation SHALL be reported as a response with a HTTP status code `200`.
B	The response SHALL be a JSON document which validates against the JSON schema coverage_metadata.json.

Chapter 19. Requirements class "OpenAPI 3.0"

Requirements Class	
http://www.opengis.net/spec/ogcapi-coverages/1.0/req/oas30	
Target type	Web API
Dependency	Conformance Class "Core"
Dependency	OGC API-Common Standard 1.0
Dependency	OpenAPI Specification 3.0.2

The OpenAPI 3.0 Requirements Class is applicable to API-Coverages as well. So an implementation of API-Coverages which supports OpenAPI 3.0 as an API Description format must also comply with the API-Common oas30 Conformance Class.

Requirement 81	**/req/oas30/oas-common**
Extends	/req/core/api-common
A	The API SHALL demonstrate conformance with the following Requirements Class of the OGC API-Common version 1.0 Standard. http://www.opengis.net/spec/ogcapi-common-1/1.0/req/oas30.

Implementations must also advertise conformance with this Requirements Class.

Requirement 82	**/req/oas30/conformance**
The list of Conformance Classes advertised by the API SHALL include:	
A	http://www.opengis.net/spec/ogcapi-coverages-1/1.0/conf/oas30

Annex A: Conformance Class Abstract Test Suite (Normative)

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

A.1. Conformance Class A

A.1.1. Requirement 1

Test id:	/conf/conf-class-a/req-name-1
Requirement:	/req/req-class-a/req-name-1
Test purpose:	Verify that…
Test method:	Inspect…

A.1.2. Requirement 2

Annex B: Examples (Informative)

B.1. Example Landing Pages

Example 4. JSON Landing Page

This example Landing Page response in JSON is for an implementation of the OGC API-Common Standard that supports:

- HTML
- JSON
- API-Common Part 2 (Geospatial Data)

This example also illustrates the `self` and `alternate` association types.


```
{
  "title": "Example API Landing Page",
  "description": "This is an example of an API landing page in JSON format",
  "attribution": "<a href='www.ign.es' rel=' '>IGN</a> <a href='www.govdata.de/dl-
de/by-2-0'>(c)</a>",
  "links": [
    {
      "rel": "service-desc",
      "type": "application/vnd.oai.openapi+json;version=3.0",
      "title": "API definition for this endpoint as JSON",
      "href": "http://www.example.com/oapi-
c/api?f=application/vnd.oai.openapi+json;version=3.0"
    },
    {
      "rel": "service-doc",
      "type": "text/html",
      "title": "API definition for this endpoint as HTML",
      "href": "http://www.example.com/oapi-c/api?f=text/html"
    },
    {
      "rel": "http://www.opengis.net/def/rel/ogc/1.0/conformance",
      "type": "application/json",
      "title": "Conformance Declaration as JSON",
      "href": "http://www.example.com/oapi-c/conformance?f=application/json"
    },
    {
      "rel": "http://www.opengis.net/def/rel/ogc/1.0/conformance",
      "type": "text/html",
      "title": "Conformance Declaration as HTML",
      "href": "http://www.example.com/oapi-c/conformance?f=text/html"
    },
    {
      "rel": "http://www.opengis.net/def/rel/ogc/1.0/data",
      "type": "application/json",
      "title": "Collections Metadata as JSON",
      "href": "http://www.example.com/oapi-c/collections?f=application/json"
    },
    {
```

```
      "rel": "http://www.opengis.net/def/rel/ogc/1.0/data",
      "type": "text/html",
      "title": "Collections Metadata as HTML",
      "href": "http://www.example.com/oapi-c/collections?f=text/html"
    },
    {
      "rel": "alternate",
      "type": "text/html",
      "title": "This Document as HTML",
      "href": "http://www.example.com/oapi-c?f=text/html"
    },
    {
      "rel": "self",
      "type": "application/json",
      "title": "This Document",
      "href": "http://www.example.com/oapi-c?f=application/json"
    }
  ]
}
```

B.2. Conformance Examples

Example 5. Conformance Response

This example response in JSON is for an implementation of the OGC API-Common Standard that supports OpenAPI 3.0 for the API definition and HTML and JSON as encodings for resources.

```
{
  "conformsTo": [
    "http://www.opengis.net/spec/ogcapi-common-1/1.0/req/core",
    "http://www.opengis.net/spec/ogcapi-common-1/1.0/req/oas30",
    "http://www.opengis.net/spec/ogcapi-common-1/1.0/req/html",
    "http://www.opengis.net/spec/ogcapi-common-1/1.0/req/json"
  ]
}
```

B.3. API Definition Examples

Example 6. JSON API Definition

This is an example of an API Definition response in JSON. It describes an implementation of the OGC API-Common Part 1 - Core Standard.

This example also illustrates:

1. Extended metadata (x-keywords),
2. Multiple Servers,
3. The use of tags to associate external documentation,
4. Responses which reference the appropriate JSON schema

```
{
  "openapi" : "3.0.2",
  "info" : {
    "title" : "A sample API conforming to the draft standard OGC API - Common -
Part 1 - Core",
    "version" : "1.0.0",
    "description" : "This is a sample OpenAPI definition of the OGC API - Common -
Part 1 - Core Standard. This example is a generic OGC API definition that
describes the Common Core of OGC Web APIs. This generic OpenAPI definition does
not provide any details on the hosted content.",
    "contact" : {
      "name" : "Acme Corporation",
      "email" : "info@example.org",
      "url" : "http://example.org/"
    },
    "license" : {
      "name" : "CC-BY 4.0 license",
      "url" : "https://creativecommons.org/licenses/by/4.0/"
    },
    "x-keywords" : [ "geospatial", "data", "api" ]
  },
  "servers" : [ {
    "url" : "https://data.example.org/",
    "description" : "Production server"
  }, {
    "url" : "https://dev.example.org/",
    "description" : "Development server"
  } ],
  "tags" : [ {
    "name" : "capabilities",
    "description" : "essential characteristics of this API"
  }, {
    "name" : "data",
    "description" : "access to data"
  }, {
    "name" : "server",
    "description" : "Information about the server hosting this API",
```

```
    "externalDocs" : {
      "description" : "information",
      "url" : "https://example.com/sample_api/documentation"
    }
  } ],
  "paths" : {
    "/" : {
      "get" : {
        "description" : "The landing page provides links to the API definition and
the conformance statements for this API.",
        "operationId" : "getLandingPage",
        "parameters" : [ {
          "$ref" : "#/components/parameters/f"
        } ],
        "responses" : {
          "200" : {
            "description" : "successful operation",
            "content" : {
              "application/json" : {
                "schema" : {
                  "$ref" :
"https://github.com/opengeospatial/oapi_common/blob/master/core/openapi/schemas/la
ndingPage.json"
                }
              }
            }
          },
          "400" : {
            "$ref" : "#/components/responses/400"
          },
          "500" : {
            "$ref" : "#/components/responses/500"
          }
        },
        "summary" : "Landing page",
        "tags" : [ "server" ]
      }
    },
    "/api" : {
      "get" : {
        "description" : "This document",
        "operationId" : "getAPIDefinition",
        "parameters" : [ {
          "$ref" : "#/components/parameters/f"
        } ],
        "responses" : {
          "200" : {
            "$ref" : "#/components/responses/200"
          },
          "400" : {
            "$ref" : "#/components/responses/400"
```

```
          },
          "default" : {
            "$ref" : "#/components/responses/400"
          }
        },
        "summary" : "This document",
        "tags" : [ "server" ]
      }
    },
    "/conformance" : {
      "get" : {
        "description" : "A list of all conformance classes that the server
conforms to.",
        "operationId" : "getConformanceClasses",
        "parameters" : [ {
          "$ref" : "#/components/parameters/f"
        } ],
        "responses" : {
          "200" : {
            "description" : "successful operation",
            "content" : {
              "application/json" : {
                "schema" : {
                  "$ref" :
"https://github.com/opengeospatial/oapi_common/blob/master/core/openapi/schemas/co
nfClasses.json"
                }
              }
            }
          },
          "400" : {
            "$ref" : "#/components/responses/400"
          },
          "500" : {
            "$ref" : "#/components/responses/500"
          }
        },
        "summary" : "API conformance definition",
        "tags" : [ "server", "capabilities" ]
      }
    }
  },
  "components" : {
    "parameters" : {
      "f" : {
        "description" : "The optional f parameter indicates the output format
which the server shall provide as part of the response document.  The default
format is JSON.",
        "explode" : false,
        "in" : "query",
        "name" : "f",
```

```
        "required" : false,
        "schema" : {
          "default" : "json",
          "enum" : [ "json", "html" ],
          "type" : "string"
        },
        "style" : "form"
      }
    },
    "responses" : {
      "200" : {
        "description" : "successful operation"
      },
      "400" : {
        "description" : "error response",
        "content" : {
          "application/json" : {
            "schema" : {
              "$ref" :
"https://github.com/opengeospatial/oapi_common/blob/master/core/openapi/schemas/ex
ception.yaml"
            }
          }
        }
      },
      "500" : {
        "description" : "server errors",
        "content" : {
          "application/json" : {
            "schema" : {
              "$ref" :
"https://github.com/opengeospatial/oapi_common/blob/master/core/openapi/schemas/ex
ception.yaml"
            }
          }
        }
      }
    }
  }
}
```

B.4. Service Metadata Examples

Example 7. Service Metadata

This is an example of how to extend the OpenAPI `info` object to include identifying metadata about both the service and the service provider.

```
{
  "info" : {
    "title" : "My Web API",
    "version" : "1.0.0",
    "description" : "This example shows population of an OpenAPI Info element with
identifying metadata for both the service and the service provider.",
    "contact" : {
      "name" : "Acme Corporation",
      "email" : "info@example.org",
      "url" : "http://example.org/",
      "x-OGC-serviceContact" : {
        "individualName" : "John Smith",
        "positionName" : "System Administrator",
        "role" : "pointOfcontact",
        "hoursOfService" : "24 Hours",
        "contractInstructions" : "None",
        "onlineResource" : "http://example.org/contact",
        "address" : {
          "deliveryPoint" : "123 Any Street",
          "city" : "Boston",
          "administrativeArea" : "MA",
          "postalCode" : "12345",
          "country" : "USA",
          "electronicMailAddress" : "smith.j@example.org"
        },
        "telephone" : {
          "voice" : "+1.123.456.7890",
          "facsimile" : "+1.123.456.7890 "
        }
      }
    },
    "license" : {
      "name" : "CC-BY 4.0 license",
      "url" : "https://creativecommons.org/licenses/by/4.0/"
    },
    "x-serviceType" : "http://www.opengis.net/doc/IS/ogcapi-common-1/1.0",
    "x-serviceTypeVersion" : "1.0",
    "x-profile" : "DGIWG",
    "x-keywords" : [ "geospatial", "data", "api" ],
    "x-fees" : "None",
    "x-accessConstraints" : "None"
  }
}
```

B.5. Collections Metadata Example

This collections metadata example response in JSON is for a dataset with a single "buildings" feature collection.

There is a link to the collections response itself (link relation type: "self").

Representations of this resource in other formats are referenced (link relation type: "alternate").

An additional link is to a GML application schema for the collection (link relation type: "describedBy").


```
{
  "links": [
    { "href": "http://data.example.org/collections.json",
      "rel": "self", "type": "application/json", "title": "this document" },
    { "href": "http://data.example.org/collections.html",
      "rel": "alternate", "type": "text/html", "title": "this document as HTML" },
    { "href": "http://schemas.example.org/1.0/foobar.xsd",
      "rel": "describedBy", "type": "application/xml", "title": "XML schema for Acme
Corporation data" }
  ],
  "collections": [
    {
      "id": "buildings",
      "title": "Buildings",
      "description": "Buildings in the city of Bonn.",
      "extent": {
        "spatial": [ 7.01, 50.63, 7.22, 50.78 ],
        "temporal": [ "2010-02-15T12:34:56Z", "2018-03-18T12:11:00Z" ]
      },
      "links": [
        { "href": "http://example.org/concepts/building.html",
          "rel": "describedBy", "type": "text/html",
          "title": "Feature catalogue for buildings" }
      ]
    }
  ]
}
```


B.6. Collection Description Examples

B.6.1. Building Collection

This Collection Description example response in JSON is for a single "buildings" collection.

The basic descriptive information includes:

- "id": an identifier for this collection

- "title": the title of this collection
- "description": self evident
- "attribution": markup providing attribution (owner, producer, logo, etc.) of this collection

The response includes links to:

- There is a link to the response itself (link relation type: "self").
- Representations of this response in other formats are referenced using link relation type "alternate".
- An additional link is to a GML application schema for the collection - using:https://www.iana.org/assignments/link-relations/link-relations.xhtml[link relation type] "describedBy".

Finally, this response includes both spatial and temporal extents.

Reference system information is not provided as the service provides geometries only in the default system (spatial: WGS 84 longitude/latitude; temporal: Gregorian calendar).

```
{
  "id": "1234567890",
  "title": "Example Collection Description Response",
  "description": "This is an example of a Collection Description in JSON format",
  "attribution": "<a href='www.ign.es' rel=' '>IGN</a> <a href='www.govdata.de/dl-
de/by-2-0'>(c)</a>",
  "links": [
    { "href": "http://data.example.org/collections.json",
      "rel": "self", "type": "application/json", "title": "this document" },
    { "href": "http://data.example.org/collections.html",
      "rel": "alternate", "type": "text/html", "title": "this document as HTML" },
    { "href": "http://schemas.example.org/1.0/foobar.xsd",
      "rel": "describedBy", "type": "application/xml", "title": "XML schema for Acme
Corporation data" }
  ],
  "extent": {
    "spatial": [ 7.01, 50.63, 7.22, 50.78 ],
    "temporal": [ "2010-02-15T12:34:56Z", "2018-03-18T12:11:00Z" ]
    }
}
```

Annex C: Revision History

Date	Release	Editor	Primary clauses modified	Description
2019-03-06	Template	C. Heazel	all	initial template

Annex D: Glossary

- **Conformance Test Module**
  set of related tests, all within a single conformance test class (OGC 08-131r3)

NOTE:	When no ambiguity is possible, the word `test` may be omitted. i.e. conformance test module is the same as conformance module. Conformance modules may be nested in a hierarchical way. This term and those associated to it are included here for consistency with ISO 19105.

- **Conformance Test Class; Conformance Test Level**
  set of conformance test modules that must be applied to receive a single **certificate of conformance**. (OGC 08-131r3)

NOTE:	When no ambiguity is possible, the word `test` may be left out, so conformance test class may be called a conformance class.

- **Executable Test Suite (ETS)**
  A set of code (e.g. Java and CTL) that provides runtime tests for the assertions defined by the ATS. Test data required to do the tests are part of the ETS (OGC 08-134)

- **Recommendation**
  expression in the content of a document conveying that among several possibilities one is recommended as particularly suitable, without mentioning or excluding others, or that a certain course of action is preferred but not necessarily required, or that (in the negative form) a certain possibility or course of action is deprecated but not prohibited (OGC 08-131r3)

NOTE:	"Although using normative language, a recommendation is not a requirement. The usual form replaces the `shall` (imperative or command) of a requirement with a `should` (suggestive or conditional)." (ISO Directives Part 2)

- **Requirement**
  expression in the content of a document conveying criteria to be fulfilled if compliance with the document is to be claimed and from which no deviation is permitted (OGC 08-131r3)

- **Requirements Class**
  aggregate of all requirement modules that must all be satisfied to satisfy a conformance test class (OGC 08-131r3)

- **Requirements Module**
  aggregate of requirements and recommendations of a specification against a single standardization target type (OGC 08-131r3)

- **Standardization Target**
  entity to which some requirements of a standard apply (OGC 08-131r3)

NOTE:	The standardization target is the entity which may receive a certificate of conformance for a requirements class.

Annex E: Bibliography

- W3C/OGC: **Spatial Data on the Web Best Practices**, W3C Working Group Note 28 September 2017, https://www.w3.org/TR/sdw-bp/
- W3C: **Data on the Web Best Practices**, W3C Recommendation 31 January 2017, https://www.w3.org/TR/dwbp/
- W3C: **Data Catalog Vocabulary**, W3C Recommendation 16 January 2014, https://www.w3.org/TR/vocab-dcat/
- IANA: **Link Relation Types**, https://www.iana.org/assignments/link-relations/link-relations.xml
- International Telecommunication Union, **ITU-T.800 : Information technology - JPEG 2000 image coding system: Core coding system**, June, 2019, https://www.itu.int/rec/T-REC-T.800-201906-I/en
- W3C, **RDF 1.1 Semantics**, February 2014, https://www.w3.org/TR/rdf11-mt/
- OGC: OGC 13-102r2, **Name type specification – Time and index coordinate reference system definitions** (OGC Policy Document), version 1, 2014