OGC API-Common

Open Geospatial Consortium

Submission Date: <yyyy-mm-dd>

Approval Date: <yyyy-mm-dd>

Publication Date: 2019-09-04

External identifier of this OGC® document: http://www.opengis.net/doc/IS/ogcapi-common/0.3

Internal reference number of this OGC® document: 19-072

Version: 0.0.3

Category: OGC® Implementation Specification

Editor: Charles Heazel

OGC API-Common

Warning

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

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

Document type: OGC®ImplementationSpecification
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

Chapter 1. Introduction

i. Abstract

The OGC has extended their suite of standards to include Resource Oriented Architectures and Web APIs. In the course of developing these standards, some practices proved to be common accross all OGC API standards. The purpose of this standard is to document those practices. It also serves as a common foundation upon which all OGC APIs will be built. As such, this OGC API Common standard serves as the "OWS Common" standard for OGC Resource Oriented APIs.

An OGC API provides a lighweight interface to access one or more resources. The resources addressed by OGC APIs fall into three categories; Foundation Resources, Spatial Resources, and Information Resources. These Resource Catagories are described in section 7 Requirement Class "Core".

The API-Common standard defines resources and access paths that are supported by all OGC APIs. These are listed in Table 1.

Table 1. Overview of Resources

Resource	Path	HTTP Method	Document Reference
Landing page	`/`	GET	7.2.1 API Landing Page
API definition	`/api`	GET	7.2.2 API Definition
Conformance classes	`/conformance`	GET	7.2.3 Declaration of Conformance Classes
Collections metadata	`/collections`	GET	8.2.1 Collections Metadata

The resources identified in Table 1 primarily support Discovery operations. Discovery operations allow clients the interrogate the API to determine its capabilities and retrieve information (metadata) about this distribution of the resource. This includes the API definition of the server(s) as well as metadata about the resources provided by those servers.

This standard also defines common Query operations for OGC APIs. Query operations allow resources or values extracted from those resources to be retrieved from the underlying data store. The information to be returned is based upon selection criteria (query string) provided by the client. This standard only defines simple query parameters which should be applicable to all resource types. Other OGC API standards may define additional query capabilities specific to their resource type.

ii. Keywords

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

ogcdoc, OGC document, property, geographic information, spatial data, spatial things, dataset, distribution, API, geojson, html, OpenAPI, AsyncAPI, REST, Common

iii. Preface

OGC Declaration

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 Inc. 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):

- Heazeltech LLC
- others TBD

v. Submitters

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

Name	**Affiliation**
Chuck Heazel *(editor)*	Heazeltech
others	TBD

Chapter 2. Scope

This specification identifies resources, captures compliance classes, and specifies requirements which are applicable to all OGC API standards. It should be included as a normative reference by all such standards.

This specification addresses two fundamental operations; discovery and query.

Discovery operations allow the API to be interrogated to determine its capabilities and retrieve information (metadata) about this distribution of a resource. This includes the API definition of the server as well as metadata about the spatial resources provided by the server.

Query operations allow spatial resources to be retrieved from the underlying data store based upon simple selection criteria, defined by the client.

Chapter 3. 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 be achieved to claim conformance are specified in the OGC Compliance Testing Policies and Procedures and the OGC Compliance Testing web site.

This standard defines five requirements / conformance classes. The conformance classes implemented by an API are advertised through the /conformance path on the landing page. Conformant implementations must pass all of the tests for each advertised conformance class.

There is a one-to-one correspondance between conformance classes and requirements classes. The tests in Annex A are organized by Requirements Class. So an implementation of the *Core* conformance class must pass all tests specified for the *Core* requirements class.

The requirements classes for OGC API-Common are:

- Core.

The *Core Requirements Class* is the minimal useful service interface for an OGC API. The requirements specified in this requirements class are mandatory for all OGC APIs

Additional capabilities such as support for transactions, complex data structures, and rich queries are specified in additional OGC API standards or in OGC managed API extensions. Those standards are extensions build on the API-Common foundation to provide the full functionality required of the API implementation.

- Collections.

The *Collections Requirements Class* extends the *Core* to enable fine-grained access to spatial resources. This requirements class is mandatory for all OGC APIs which expose spatial resources.

The structure and organziation of a collection of spatial resources is very much dependent on the nature of that resource and the expected access patterns. This is information which cannot be specified in a common manner. The *Collections Requirements Class* specifies the requirements necessary to discover and understand that structure and organization. Requirements governing the resource collections themselves are specified in the resource-specific OGC API standards.

- HTML,
- GeoJSON

Neither the nor *Core* nor *Collections* requirements class mandate a specific encoding or format for representing resources. The *HTML* and *GeoJSON* requirements classes specify representations for these resources in commonly used encodings for spatial data on the web.

Neither of these encodings are mandatory. An implementation of the *API-Common* may decide to implement another encoding instead of, or in addition to, these two.

- OpenAPI 3.0.

The *API-Common* does not mandate any encoding or format for the formal definition of the API either. The prefered option is the OpenAPI 3.0 specification. The *OpenAPI 3.0* requirements class has been specified for APIs implementing OpenAPI 3.0.

Chapter 4. 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.

- Open API Initiative: **OpenAPI Specification 3.0.2**, https://github.com/OAI/OpenAPI-Specification/blob/master/versions/3.0.2.md
- Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T.: **IETF RFC 2616, HTTP/1.1**, http://tools.ietf.org/rfc/rfc2616.txt
- Rescorla, E.: **IETF RFC 2818, HTTP Over TLS**, http://tools.ietf.org/rfc/rfc2818.txt
- Klyne, G., Newman, C.: **IETF RFC 3339, Date and Time on the Internet: Timestamps**, http://tools.ietf.org/rfc/rfc3339.txt
- Nottingham, M.: **IETF RFC 8288, Web Linking**, http://tools.ietf.org/rfc/rfc8288.txt
- Butler, H., Daly, M., Doyle, A., Gillies, S., Hagen, S., Schaub, T.: **IETF RFC 7946, The GeoJSON Format**, https://tools.ietf.org/rfc/rfc7946.txt
- W3C: **HTML5, W3C Recommendation**, http://www.w3.org/TR/html5/
- **Schema.org**: http://schema.org/docs/schemas.html

Chapter 5. Terms and Definitions

This document uses the terms defined in Sub-clause 5.3 of [OGC 06-121r8], 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.

5.1. dataset

collection of data, published or curated by a single agent, and available for access or download in one or more formats (DCAT)

5.2. distribution

represents an accessible form of a **dataset** (DCAT)

EXAMPLE: a downloadable file, an RSS feed or a web service that provides the data.

Chapter 6. Conventions

6.1. Identifiers

The normative provisions in this draft standard are denoted by the URI `http://www.opengis.net/spec/OAPI_Common/1.0`.

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

6.2. Link relations

To express relationships between resources, RFC 8288 (Web Linking) and registered link relation types are used.

6.3. Use of HTTPS

For simplicity, this document in general only refers to the HTTP protocol. This is not meant to exclude the use of HTTPS and simply is a shorthand notation for "HTTP or HTTPS". In fact, most servers are expected to use HTTPS, not HTTP.

6.4. API definition

6.4.1. General remarks

Good documentation is essential for every API so that developers can more easily learn how to use the API. In the best case, documentation would be available both in HTML for human consumption and in a machine readable format that can be processed by software for run-time binding.

This standard specifies requirements and recommendations for APIs that share spatial resources and want to follow a standard way of doing so. In general, APIs will go beyond the requirements and recommendations stated in this standard. They will support additional operations, parameters, etc. that are specific to the API or the software tool used to implement the API.

6.4.2. Role of OpenAPI

This document uses OpenAPI 3.0 fragments as examples and to formally state requirements. Using OpenAPI 3.0 is not required for implementing an OGC API. Other API definition languages may be used along with, or instead of OpenAPI. However, any API definition language used should have an associated requirements class advertised through the /conformance path.

This approach is used to avoid lock-in to a specific approach to defining an API. This standard includes a requirements class for API definitions that follow the OpenAPI specification 3.0. Requirements classes for additional API definition languages will be added as the API landscape continues to evolve.

In this document, fragments of OpenAPI definitions are shown in YAML since YAML is easier to

format than JSON and is typically used in OpenAPI editors.

6.4.3. References to OpenAPI components in normative statements

Some normative statements (requirements, recommendations and permissions) use a phrase that a component in the API definition of the server must be "based upon" a schema or parameter component in the OGC schema repository.

In this case, the following changes to the pre-defined OpenAPI component are permitted:

- If the server supports an XML encoding, `xml` properties may be added to the relevant OpenAPI schema components.
- The range of values of a parameter or property may be extended (additional values) or constrained (if a subset of all possible values are applicable to the server). An example for a constrained range of values is to explicitly specify the supported values of a string parameter or property using an enum.
- Additional properties may be added to the schema definition of a Response Object.
- Informative text may be changed or added, like comments or description properties.

For API definitions that do not conform to the OpenAPI Specification 3.0 the normative statement should be interpreted in the context of the API definition language used.

6.4.4. Paths in OpenAPI definitions

All paths in an OpenAPI definition are relative to the base URL of a server. Unlike Web Services, an API is decoupled from the server(s). Some ramifications of this are:

- An API may be hosted (replicated) on more than one server.
- Parts of an API may be distributed across multiple servers.

Example 1. URL of the OpenAPI definition

If the OpenAPI Server Object looks like this:

```
servers:
  - url: https://dev.example.org/
    description: Development server
  - url: https://data.example.org/
    description: Production server
```

The path "/mypath" in the OpenAPI definition of the API would be the URL `https://data.example.org/mypath` for the production server.

6.4.5. Reusable OpenAPI components

Reusable components for OpenAPI definitions for a OGC API are referenced from this document.

CAUTION	During the development phase, these components use a base URL of "https://raw.githubusercontent.com/opengeospatial/oapi_common/master/", but eventually they are expected to be available under the base URL "http://schemas.opengis.net/wfs/3.0/openapi/".

Chapter 7. Overview

7.1. Evolution from OGC Web Services

OGC Web Service (OWS) standards implement a Remote-Procedure-Call-over-HTTP architectural style using XML for payloads. This was the state-of-the-art when OGC Web Services (OWS) were originally designed in the late 1990s. However, times have changed. New Resource-Oriented APIs have begun to replace Service-Oriented Web Services. And new OGC API standards are under development to provide API alternatives to the OWS standards.

OGC API (OAPI) Common specifies the common kernel of this API approach to services that follows the current Web architecture. In particular, the W3C/OGC best practices for sharing Spatial Data on the Web as well as the W3C best practices for sharing Data on the Web.

Beside the general alignment with the architecture of the Web (e.g., consistency with HTTP/HTTPS, hypermedia controls), another goal for OGC API standards is modularization. This goal has several facets:

- Clear separation between core requirements and more advanced capabilities. This document specifies the core or *common* requirements that are relevant for almost everyone who wants to build a spatial API. Additional capabilities that several communities are using today will be specified as extensions to the Common API.
- Technologies that change more frequently are decoupled and specified in separate modules ("requirements classes" in OGC terminology). This enables, for example, the use/re-use of new encodings for spatial data or API descriptions.
- Modularization is not just about a single "service". OGC APIs will provide building blocks that can be reused in APIs in general. In other words, a server supporting the OGC-Feature API should not be seen as a standalone service. Rather it should be viewed as a collection of API building blocks which together implement API-Feature capabilities. A corollary of this is that it should be possible to implement an API that simultaneously conforms to conformance classes from the Feature, Coverage, and other OGC Web API standards.

Implementations of OGC API Common are intended to support two different approaches for how clients can use the API.

In the first approach, clients are implemented with knowledge about this standard and its resource types. The clients navigate the resources based on this knowledge and based on the responses provided by the API. The API definition may be used to determine details, e.g., on filter parameters, but this may not be necessary depending on the needs of the client. These are clients that are in general able to use multiple APIs as long as they implement OGC API Common.

The other approach targets developers that are not familiar with the OGC API standards, but want to interact with spatial data provided by an API that happens to implement OGC API Common. In this case the developer will study and use the API definition, typically an OpenAPI document, to understand the API and implement client code to interact with the API. This assumes familiarity with the API definition language and the related tooling, but it should not be necessary to study the OGC API standards.

7.2. Encodings

This standard does not mandate any encoding or format. But it does provide extensions for encodings which are commonly used in OGC APIs. In addition to HTML as the standard encoding for Web content, rules for commonly used encodings for spatial data on the web are provided (GeoJSON).

None of these encodings is mandatory. An implementation of the *Core* requirements class does not have to support any of them. It may instead implement an entirely different set of encodings.

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.

GeoJSON is a commonly used format that is simple to understand and well supported by tools and software libraries. Since most Web developers are comfortable with using a JSON-based format, it is recommended for APIs which expose feature data if the feature data can be represented in GeoJSON for the intended use.

Some examples of cases that are out-of-scope for GeoJSON are:

- When solids are used for geometries (e.g. in a 3D city model),
- Geometries that include non-linear curve interpolations that cannot be simplified (e.g., use of arcs in authoritative geometries),
- Geometries that have to be represented in a coordinate reference system that is not based on WGS 84 longitude/latitude (e.g. an authoritative national reference system),
- Features that have more than one geometric property.

The recommendations for using HTML and GeoJSON reflect the importance of HTML and the current popularity of JSON-based data formats. As the practices in the Web community evolve, these recommendations will likely be updated in future versions of this standard to provide guidance on using other encodings.

This part of the OAPI standard does not provide any guidance on other encodings. The supported encodings, or more precisely the media types of the supported encodings, can be determined from the API definition. The desired encoding is selected using HTTP content negotiation.

For example, if the server supports GeoJSON Text Sequences an encoding that is based on JSON text sequences and GeoJSON to support streaming by making the data incrementally parseable, the media type `application/geo+json-seq` would be used.

In addition, HTTP supports compression. Therefore the standard HTTP compression mechanisms can be used to reduce the size of messages between the server and the client.

Chapter 8. Requirement Class "Core"

Requirements Class	
http://www.opengis.net/spec/OAPI_Common/1.0/req/core	
Target type	Web API
Dependency	RFC 2616 (HTTP/1.1)
Dependency	RFC 2818 (HTTP over TLS)
Dependency	RFC 5988 (Web Linking)

8.1. Overview

8.1.1. Resources

An OGC API provides a lighweight interface to access one or more resources. The resources addressed by OGC APIs fall into three categories; Foundation Resources, Spatial Resources, and Information Resources.

Foundation Resources are those resources which are common across all OGC APIs. Those resources are defined in this OGC API-Common standard. Other OGC API standards re-use these resources and, where necessary, extend them to address unique requirements.

Spatial Resources are the resource which we usually think of as Geospatial Data. They include Features, Coverages, and Maps. This Standard defines basic patterns for accessing Spatial Resources. Additional OGC API Standards have been developed to address specific API requirements for each Spatial Resource type.

Information Resources are non-spatial resources which support the operation of the API or the access and use of the Spatial Resources.

8.1.2. Modular APIs

A goal of OGC API standards is to provide rapid and easy access to spatial resources. To meet this goal, the needs of both the resource provider and the resource consumer must be considered. Our approach is to provide a modular framework of API components. This framework provides a consistent "look and feel" across all OGC APIs. When API servers and clients are built from the same set of modules, the likelyhood that they will integrate at run-time is greatly enhanced.

8.1.3. Navigation

OGC APIs are designed to support two access patterns; Hypermedia Access, and Direct Access. OGC APIs support both access patterns through the use of API Definition documents, standardized paths, and standardized hypermedia schemas.

Hypermedia Access

Hypermedia Access is the use of hypermedia links to navigate from one resource to another. This pattern is typical of the Web Browser environment. A resource consumer (typically a human) starts from a landing page, selects a link on that page, then moves on to the referenced resource.

Navigation of hyperlinks is facilitated if the hyperlink includes information about the resource type at the link destination. Therefore, OGC APIs will use a set of common link relationships.

Requirement 1	**/req/core/link-relations**
A	Links used in OGC APIs SHALL use the following link relations: • `alternate`: links to this resource in another media type (the media type is specified in the `type` link attribute) • `conformance`: links to conformance information • `data`: links to an information resource • `describedBy`: links to external resources which further describe the subject resource • `items`: links to each individual resource which is included in a collection resource • `self`: links to this resource, • `service-desc`: links to the API Definition • `service-doc`: an alternative to `service-desc`
B	This requirement does not constrain the use of other link relations for resource types not addressed by this requirement.

OGC API hyperlinks are defined using the following Hyperlink Schema.

Hyperlink Schema

```
{
    "$schema": "http://json-schema.org/draft-07/schema#",
    "title": "Link Schema",
    "description": "Schema for external references",
    "type": "object",
    "required": [
        "href"
        ],
    "properties": {
        "href": {
            "type": "string"
            },
        "rel": {
            "type": "string"
            },
        "hreflang": {
            "type": "string"
            },
        "title": {
            "type": "string"
            }
        }
    }
```

Direct Access

Direct Access requires that the resource consumer possesses knowlege of the path to the resouce prior to attempting access. Typically this knowlege comes from the use of standard paths, receiving the path from another entity, or by processing an API definition resource. Direct access is particularly applicable to software analytics where there is no human in the loop.

Direct access is facilitated by the use of standard URL paths. The requirements in this Requirements Class are organized around these standard paths.

8.2. Foundation Resources

Foundation resources are those resources which are provided by every OGC API.

The standard paths defined in this Standard for Foundation Resources are:

1. "/" - the landing page
2. "/api" - the API Definition document for this API
3. "/conformance" - the conformance information for this API

8.2.1. API landing page

Each OGC API has a single `LandingPage` (path `/`).

The purpose of the landing page is to provide users with the basic information they need to use this API as well as links to the resources exposed through the API.

Operation

Requirement 2	**/req/core/root-op**
A	The server SHALL support the HTTP GET operation at the path `/`.

Response

Requirement 3	**/req/core/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: • the API definition (relation type 'service-desc' or 'service-doc') • `/conformance` (relation type 'conformance') • one or more information resources (relation type 'data')

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

The landing page returned by this operation is based on the following Landing Page Schema. Examples of OGC landing pages are provided in Example Landing Pages.

Landing Page Schema

```
{
"$schema": "http://json-schema.org/draft-07/schema#",
"title": "Landing Page Schema",
"description": "JSON schema for the OGC API-Common landing page",
"type": "object",
"required": [
    "links"
    ],
"properties": {
    "title": {
        "description": "The title of the API",
        "type": "string"
        },
    "description": {
        "description": "A textual description of the API",
        "type": "string"
        },
    "links": {
        "description": "Links to the resources exposed through this API.",
        "type": "array",
        "items": {"$href": "link.json"}
        }
    },
    "patternProperties": {
        "^x-": {}
    },
    "additionalProperties": true
}
```

Error Situations

See HTTP Status Codes for general guidance.

8.2.2. API Definition

Every API is expected to provide a definition that describes capabilities provided by the API. This document 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.

Operation

Requirement 4	**/req/core/api-definition-op**
A	The URIs of all API definitions referenced from the landing page SHALL support the HTTP GET method.

Response

Requirement 5	/req/core/api-definition-success
A	A GET request to the URI of an API definition linked from the landing page (link relations `service-desc` or `service-doc`) with an `Accept` header with the value of the link property `type` SHALL return a document consistent with the requested media type.

Recommendation 1	/rec/core/api-definition-oas
A	If the API definition document uses the OpenAPI Specification 3.0, the document SHOULD conform to the OpenAPI Specification 3.0 requirements class.

If multiple API definition formats are supported, use content negotiation to select the desired representation.

Error Situations

See HTTP Status Codes for general guidance.

8.2.3. Declaration of Conformance Classes

To support "generic" clients that want to accessing OGC APIs in general - and not "just" a specific API / server, the API has to declare the requirements classes it implements and conforms to.

Operation

Requirement 6	/req/core/conformance-op
A	The API SHALL support the HTTP GET operation at the path `/conformance`.

Response

Requirement 7	/req/core/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 OpenAPI 3.0 schema confClasses.json and list all OGC API conformance classes that the API conforms to.

The conformance resource returned by this operation is based on the following Conformance Schema. Examples of OGC conformance resources are provided in Conformance Examples.

Conformance Schema

```
{
"$schema": "http://json-schema.org/draft-07/schema#",
"title": "Conformance Classes 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/OAPI_Common/1.0/req/core"
            }
        }
    }
}
```

Error situations

See HTTP Status Codes for general guidance.

8.3. Spatial Resources

There is no requirement that every OGC API support Spatial Resources. Therefore, Spatial Resources are addressed in a separate `Collections` Requirement Class. This class is described in Section 8.

8.4. Information Resources

Information Resources are non-spatial resources which support the operation of the API or the access and use of the Spatial Resources. These resources are usually specific to a spatial resource type and will be defined in the appropriate API standards.

Information Resources can exposed using two path templates:

- /collections/{collectionId}/{resourceType}
- /{resourceType}

Where

{collectionId} = a unique identifier for a Spatial Resource collection.

{resourceType} = a text string identifying the Information Resource type.

Information Resources associated with a specific collection should be accessed through the `/collections` path. Those which are not associated with a specific collection should use the `/{resourceType}` template.

The OGC API-Common standard does not define any Information Resource types. However Table 2 provides a maping of the know Information Resource types to the standard where they are defined.

Table 2. Information Resource Types

Resource Type	API Standard
TBD	TBD

8.5. General Requirements

The following general requirements and recomendations apply to all OGC APIs.

8.5.1. 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 8	**/req/core/http**
A	The API SHALL conform to HTTP 1.1.
B	If the API supports HTTPS, then the API SHALL also conform to HTTP over TLS.

8.5.2. HTTP Status Codes

Table 3 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 3. Typical HTTP status codes

Status code	Description
`200`	A successful request.
`304`	An entity tag was provided in the request and the resource has not been changed since the previous request.
`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.

Status code	Description
401	The request requires user authentication. The response includes a `WWW-Authenticate` header field containing a challenge applicable to the requested resource.
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 authorised 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	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.

More specific guidance is provided for each resource, where applicable.

Permission 1	**/per/core/additional-status-codes**
A	Servers MAY support other capabilities of the HTTP protocol and, therefore, MAY return other status codes than those listed in [status_codes], too.

8.5.3. 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 2616).

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

8.5.4. Support for Cross-Origin Requests

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

Recommendation 3	**/rec/core/cross-origin**

A	If the server is intended to be accessed from the 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)

8.5.5. Encodings

While the OAPI Common standard does not specify any mandatory encoding, the following encodings are recommended. See Clause 6 (Overview) for a discussion.

Recommendation 4	**/rec/core/html**
A	To support browsing a API with a web browser and to enable search engines to crawl and index the dataset, implementations SHOULD consider to support an HTML encoding.

Recommendation 5	**/rec/core/geojson**
A	If the resource can be represented for the intended use in GeoJSON, implementations SHOULD consider to support GeoJSON as an encoding.

Requirement `/req/core/http` implies that the encoding of a 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 API 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 API.

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") in the browser address bar, can study the API definition.

NOTE

Two common approaches are:

- 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.

8.5.6. Link Headers

Recommendation 6	**/rec/core/link-header**
A	Links included in payload of responses SHOULD also be included as `Link` headers in the HTTP response according to RFC 8288, Clause 3.
B	This recommendation does not apply, if there are a large number of links included in a response or a link is not known when the HTTP headers of the response are created.

Chapter 9. Requirement Class "Collections"

Requirements Class	
http://www.opengis.net/spec/OAPI_Common/1.0/req/collections	
Target type	Web API
Dependency	RFC 3339 (Date and Time on the Internet: Timestamps)

9.1. Overview

Spatial Resources are the resources which we usually think of as Geospatial Data. They include Features, Coverages, and Maps. This Conformance Class defines basic patterns for accessing Spatial Resources. Additional OGC API Standards have been developed to address specific API requirements for each Spatial Resource type.

OGC APIs are designed to support two access patterns; Hypermedia Access, and Direct Access. OGC APIs support both access patterns through the use of API Definition documents, standardized paths, and standardized hypermedia schemas.

Hypermedia Access was described in the Navigation section of Clause 7. For Spatial Resources, hypermedia navigation is enabled through the links included in each schema defined by this Reqirement Class.

Direct access is the use of know URL paths to access a resource directly. The requirements in this Requirement Class are organized around the standard paths for Spatial Data.

9.2. Spatial Resources

Detailed requirements for each Spatial Resource type are dealt with in the resource-specific API standards. However, this API Common standard has the responsiblity to see that all OGC API standards work together by:

1. Providing specifications for the description of each collection (/collections/{collectionId}), and the list of collections (/collections)
2. Providing a consistent framework for serving spatial data from the OGC API, regardless of the type. Consistent means that #1 works exactly the same (potentially with type-specific additional properties) and that the different types of data can all be collections on the same OGC API end-point.
3. Providing a tie point for other OGC API modules to connect to and reference (processes inputs & outputs, cataloging, searching and filtering collections, detailed metadata, tiles, styles, clipping and intersecting bounding boxes in common) Just by virtue of understanding that /collections/{collectionId} points to a spatial **data layer**.

Spatial Resources are exposed using the path template.

```
/collections/{collectionId}/items
```

The resources returned from each node in this template are described in Table 4.

Table 4. Spatial Resource Paths

Path Template	Resource
/collections	Metadata describing the spatial collections avalable from this API.
/collections/{collectionId}	Metadata describing the collection with the unique identifier `{collectionId}`
/collections/{collectionId}/item	The spatial collection resource identified by the `{collectionId}` parameter.

9.2.1. Collections Metadata

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

Operation

Requirement 9	/req/collections/rc-md-op
A	The API SHALL support the HTTP GET operation at the path `/collections`.

Response

Requirement 10	/req/collections/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 Metadata Schema. Examples of collections metadata are provided in Collections Metadata Examples.

Collections Metadata Schema

```
{
    "$schema": "http://json-schema.org/draft-07/schema#",
    "title": "Collections Schema",
    "description": "This schema defines the metadata resource returned from
/collections.",
    "type": "object",
    "required": [
        "links",
        "collections"
        ],
    "properties": {
        "links": {
            "type": "array",
            "items": {"$href": "link.json"}
            },
        "collections": {
            "type": "array",
            "items": {"$href": "collectionInfo.json"}
            }
        }
    }
```

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

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

Requirement 11	**/req/collections/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.

Recomendation 7	**/rec/collections/rc-md-desciptions**

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	This applies to resources that describe to the whole dataset.
C	The `type` link parameter SHOULD be provided for each link.

The `collections` property of the Collections Metadata provides a description of each collection. These descriptions are based on the Collection Information Schema. This schema is described in detail in the Collection Information section of this Standard. The following requirements and recommendations govern the use of Collection Information in the Collections Metadata.

Requirement 12	**/req/collections/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 Information resource.

While it is prefered that the Collections Metadata describe all of the collections accessible through the API, in some cases that is impractical. Developers have an option to only return a subset, as long as they provide a way to retrieve the remaining metadata as well.

Permission 2	**/per/core/rc-md-items**
A	To support servers with many collections, servers MAY limit the number of items included in the `collections` property.

Error situations

See HTTP Status Codes for general guidance.

9.2.2. Collection Information

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.

Operation

Requirement 13	/req/collections/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`).

Response

Requirement 14	/req/collections/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 Information is based on the Collection Information Schema. Examples of Collection Information are provided in [collection-information-examples].

Collection Information Schema

```
{
    "$schema": "http://json-schema.org/draft-07/schema#",
    "title": "Collection Information Schema",
    "description": "This schema defines metadata resource returned from
/collections/{collectionId}.",
    "type": "object",
    "required": [
        "id",
        "links"
        ],
    "properties": {
        "id": {
          "type": "string"
          },
        "title": {
            "type": "string"
            },
        "description": {
            "type": "string"
            },
        "links": {
            "type": "array",
            "items": {"$href": "link.json"}
            },
        "extent": {"$href": "bbox.yaml"},
        "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"
                ]
            }
        }
    }
```

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

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

Requirement 15	/req/collections/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.

Recomendation 8	/rec/core/rc-md-items-desciptions
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.

Additional requirements and recomendations apply to the `extent` propery of the Collection Information.

Requirement 16	/req/collections/rc-md-extent
A	For each spatial resource collection, the `extent` property, if provided, SHALL provide bounding boxes that include all spatial geometries and time intervals that include all temporal geometries 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 9	/rec/core/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 3	/per/core/rc-md-extent-extensions
A	The Core only specifies requirements for spatial and temporal extents. However, the `extent` object MAY be extended with additional members to represent other extents, for example, thermal or pressure ranges.
B	The Core only supports spatial extents in WGS84 longitude/latitude andtemporal extents in the calendar (these are the only enum values in extent.yaml).
C	Extensions to the Core MAY add additional reference systems to the `extent` object.

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 3).

9.2.3. 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 purvue of the API standard for that resource type.

Operation

Requirement 17	/req/collections/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`).

Response

Requirement 18	**/req/collections/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.

Error situations

See HTTP Status Codes for general guidance.

9.3. Information Resources

Information Resources are non-spatial resources which support the operation of the API or the access and use of the Spatial Resources. They are described in the Information Resources section.

Information Resources related to Spatial Resources can exposed using the path template:

- /collections/{collectionId}/{resourceType}

The resources returned from each node in this template are described in Table 5.

Table 5. Information Resource Paths

Path Template	Resource
/collections	The root resource describing the spatial collections avalable from this API.
/collections/{collectionId}	Identifies a collection with the unique identifier `{collectionId}`
/collections/{collectionId}/{resourceType}	Identifies an Information Resource of type {resourceType} associated with the `{collectionId}` collection.

The OGC API-Common standard does not define any Information Resource types. However Table 2 provides a maping of the know Information Resource types to the standard where they are defined.

9.4. Parameter Modules

Query parameters are used in URLs to limit the resources which are returned on a GET request. The API Common standard defines two standard parameters for use in OGC API standards.

9.4.1. Parameter bbox

Requirement 19	**/req/collections/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`

Requirement 20	**/req/collections/rc-bbox-response**
A	If the `bbox` parameter is provided, 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 • Lower left corner, coordinate axis 3 (optional) • Upper right corner, coordinate axis 1 • Upper right corner, coordinate axis 2 • Upper right corner, coordinate axis 3 (optional)
C	The coordinate reference system of the values on 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`.
D	The coordinate values SHALL be within the extent specified for the coordinate reference system.

"Intersects" means that the rectangular area specified in the parameter `bbox` includes a coordinate that is part of the (spatial) geometry of the resource. This includes the boundaries of the geometries (e.g. for curves the start and end position and for surfaces the outer and inner rings).

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

For WGS84 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 the first value (west-most box edge) is larger than the third value (east-most box edge).

Example 2. 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`.

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

9.4.2. Parameter datetime

Requirement 21	**/req/collections/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`

Requirement 22	**/req/collections/rc-time-response**
A	If the `datetime` parameter is provided, 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 resourcee has multiple temporal properties, it is the decision of the API 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	The temporal information is either a date-time 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.

Example 3. 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` in the building features), 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 4. 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.

9.5. General Requirements

The following general requirements and recomendations apply to all OGC APIs which host Spatial Resources.

9.5.1. Coordinate Reference Systems

As discussed in Chapter 9 of the W3C/OGC Spatial Data on the Web Best Practices document, how to express and share the location of resources in a consistent way is one of the most fundamental aspects of publishing geographic data and it is important to be clear about the coordinate reference system that coordinates are in.

For the reasons discussed in the Best Practices, OGC APIs use WGS84 longitude and latitude as the default coordinate reference system.

Requirement 23	**/req/collections/crs84**

A	Unless the client explicitly requests a different coordinate reference system, all spatial geometries SHALL be in the coordinate reference system http://www.opengis.net/def/crs/OGC/1.3/CRS84 (WGS 84 longitude/latitude) for geometries without height information and http://www.opengis.net/def/crs/OGC/0/CRS84h (WGS 84 longitude/latitude plus ellipsoidal height) for geometries with height information.

The implementations compliant with the Core are not required to support publishing geometries in coordinate reference systems other than http://www.opengis.net/def/crs/OGC/1.3/CRS84. The Core also does not specify a capability to request geometries in a different reference system than the native one of the published resource. Such a capability will be specified in other OGC API standards.

Chapter 10. Requirements classes for encodings

10.1. Overview

This clause specifies two pre-defined requirements classes for encodings to be used by an OGC API implementation. These encodings are commonly used encodings for spatial data on the web:

- HTML
- GeoJSON

Neiter of these encodings are mandatory and an implementation of the Core requirements class may implement either, both, or none of them. Clause 6 (Overview) includes a discussion about recommended encodings.

10.2. Requirement Class "HTML"

Geographic information that is only accessible in formats like GeoJSON or GML has two issues:

- The data is not discoverable using the most common mechanism for discovering information, that is the search engines of the Web,
- The data can not be viewed directly in a browser - additional tools are required to view the data.

Therefore, sharing data on the Web should include publication in HTML. To be consistent with the Web, it should be done in a way that enables users and search engines to access all data.

This is discussed in detail in Best Practice 2: Make your spatial data indexable by search engines [SDWBP]. This standard therefore recommends supporting HTML as an encoding.

Requirements Class	
http://www.opengis.net/spec/OAPI_Common/1.0/req/html	
Target type	Web API
Dependency	Conformance Class "OAPI Core"
Dependency	HTML5
Dependency	Schema.org

Requirement 24	**/req/html/definition**
A	Every `200`-response of an operation of the API SHALL support the media type `text/html`.

Requirement 25	/req/html/content
A	Every `200`-response of the API with the media type "text/html" SHALL be a HTML 5 document that includes the following information in the HTML body: • all information identified in the schemas of the Response Object in the HTML `<body/>`, and • all links in HTML `<a/>` elements in the HTML `<body/>`.

Recommendation 10	/rec/html/schema-org
A	A `200`-response with the media type `text/html`, SHOULD include Schema.org annotations.

10.3. Requirement Class "GeoJSON"

GeoJSON is a commonly used format that is simple to understand and well supported by tools and software libraries. Since most Web developers are comfortable with using a JSON-based format, supporting GeoJSON is recommended if the resource can be represented in GeoJSON for the intended use.

Requirements Class	
http://www.opengis.net/spec/OAPI_Common/1.0/req/geojson	
Target type	Web API
Dependency	OAPI Core
Dependency	GeoJSON

Requirement 26	/req/geojson/definition
A	`200`-responses of the server SHALL support the following media types: • `application/geo+json` for resources that include feature content, and • `application/json` for all other resources.

Requirement 27	/req/geojson/content

A	Every `200`-response with the media type `application/geo+json` SHALL be • a GeoJSON FeatureCollection Object for feature collections, and • a GeoJSON Feature Object for features.
B	The schema of all responses with the media type `application/json` SHALL conform with the JSON Schema specified for that resource.

Templates for the definition of the schemas for the GeoJSON responses in OpenAPI definitions are available at featureCollectionGeoJSON.yaml and featureGeoJSON.yaml. These are generic schemas that do not include any application schema information about specific resource types or their properties.

Chapter 11. Requirements class "OpenAPI 3.0"

11.1. Basic requirements

APIs conforming to this requirements class document themselves by an OpenAPI Document.

Requirements Class	
http://www.opengis.net/spec/OAPI_Common/1.0/req/oas30	
Target type	Web API
Dependency	Conformance Class "OAPI Core"
Dependency	OpenAPI Specification 3.0.2

Requirement 28	**/req/oas30/oas-definition-1**
A	An OpenAPI definition in JSON using the media type `application/vnd.oai.openapi+json;version=3.0` and a HTML version of the API definition using the media type `text/html` SHALL be available.

CAUTION	ISSUE 117 The OpenAPI media type has not been registered yet with IANA and will likely change. We need to update the media type after registration.

Requirement 29	**/req/oas30/oas-definition-2**
A	The JSON representation SHALL conform to the OpenAPI Specification, version 3.0.

Two example OpenAPI documents are included in Annex B.

Requirement 30	**/req/oas30/oas-impl**
A	The API SHALL implement all capabilities specified in the OpenAPI definition.

11.2. Complete definition

Requirement 31	**/req/oas30/completeness**

A	The OpenAPI definition SHALL specify for each operation all HTTP Status Codes and Response Objects that the API uses in responses.
B	This includes the successful execution of an operation as well as all error situations that originate from the server.

Note that APIs that, for example, are access-controlled (see Security), support web cache validation, CORS or that use HTTP redirection will make use of additional HTTP status codes beyond regular codes such as `200` for successful GET requests and `400`, `404` or `500` for error situations. See [http_status_codes].

Clients have to be prepared to receive responses not documented in the OpenAPI definition. For example, additional errors may occur in the transport layer outside of the server.

11.3. Exceptions

Requirement 32	**/req/oas30/exceptions-codes**
A	For error situations that originate from an API server, the API definition SHALL cover all applicable HTTP Status Codes.

Example 5. An exception response object definition

```
description: An error occurred.
content:
  application/json:
    schema:
      $ref:
https://raw.githubusercontent.com/opengeospatial/OAPI/openapi/schemas/exception.ya
ml
  text/html:
    schema:
      type: string
```

11.4. Security

Requirement 33	**/req/oas30/security**
A	For cases, where the operations of the API are access-controlled, the security scheme(s) and requirements SHALL be documented in the OpenAPI definition.

The OpenAPI specification currently supports the following security schemes:

- HTTP authentication,
- an API key (either as a header or as a query parameter),
- OAuth2's common flows (implicit, password, application and access code) as defined in RFC6749, and
- OpenID Connect Discovery.

Chapter 12. Media Types

JSON media types that would typically be used in on OGC API that supports JSON are

- `application/geo+json` for feature collections and features, and
- `application/json` for all other resources.

XML media types that would typically occur in on OGC API that supports XML are

- `application/gml+xml;version=3.2` for any GML 3.2 feature collections and features,
- `application/gml+xml;version=3.2;profile=http://www.opengis.net/def/profile/ogc/2.0/gml-sf0` for GML 3.2 feature collections and features conforming to the GML Simple Feature Level 0 profile,
- `application/gml+xml;version=3.2;profile=http://www.opengis.net/def/profile/ogc/2.0/gml-sf2` for GML 3.2 feature collections and features conforming to the GML Simple Feature Level 2 profile, and
- `application/xml` for all other resources.

The typical HTML media type for all "web pages" in an OGC API would be `text/html`.

The media types for an OpenAPI definition are `vnd.oai.openapi+json;version=3.0` (JSON) and `application/vnd.oai.openapi;version=3.0` (YAML).

NOTE	The OpenAPI media type has not been registered yet with IANA and may change.

Annex A: Abstract Test Suite (Normative)

CAUTION	ISSUE 112 Currently, only the Core and OpenAPI 3.0 conformance classes are addressed by the Abstract Test Suite. The other four conformance classes (HTML, GeoJSON, GML-SF0, GML-SF2) are not yet covered. How they will be addressed is the subject of on-going discussions within the WFS 3.0 SWG.

A.1. A.1 Overview

WFS 3.0 is not a Web Service in the traditional sense. Rather, it defines the behavior and content of a set of Resources exposed through an RESTful Application Programing Interface (API). Compliance testing for WFS 3.0 and similar standards must answer three questions:

1. Are the capabilities advertised through the API Description compliant with the standard?
2. Does the API implement those capabilities as advertised?
3. Do the resources returned by the micro-services meet the structure and content requirements of the standard?

Further complicating the issue, an API may expose resources in addition to those defined by the standard. A test engine must be able to traverse the API description document, identify test points, and ignore resource paths which are not to be tested. The process for identifying test points is provided in Section A.4.3.

A.2. A.2 Conventions

The following conventions apply to this Abstract Test Suite:

A.2.1. A.2.1 Path Templates

Path templates are used throughout these test suites. Path templating refers to the usage of curly braces “{}” to mark a section of a URL path that can be replaced using path parameters. The terms used to describe portions of these templates are based on the URL syntax described in RFC 3986.

- scheme: http | https
- authority: DNS name of the server with optional port number
- path: The slash delimited identifier for a resource on the server
- query: query parameters following the "?" character
- fragment: identifies an element within the resource. Preceded by the "#" character

A.2.2. A.2.2 API Description Document

The WFS 3.0 standard does not mandate a standard format for the API Description Document. However, some form of standard is needed if tests are to be accurately described and implemented. Therefore, this Abstract Test Suite asserts that the API Description document is compliant with

OpenAPI 3.0. This Test Suite will be updated if and when an alternative is commonly adopted.

A.2.3. A.2.3 Resource Encodings

The WFS 3.0 standard does not mandate a standard encoding for resources returned by the API. Yet a compliance test requires some minimal level of expected behavior. Therefore, this Abstract Test Suite asserts that the API returns resources encoded in HTML and GeoJSON. Since no compliance test suite exists for these encodings at this time, the resources shall be presented to the test operator for human inspection.

A.2.4. A.2.4 Processing Security Objects

OpenAPI does not provide a standard way to associate a security requirement with a single server URI. Therefore, WFS 3.0 compliance tests will have to make that association through the runtime challenge-response transaction. At this time the role of the Security Objects should be considered advisory.

Security Requirements can be defined at both the OpanAPI root level and at the Operation Object level. The following rules should be followed to understand the scope of a Security Requirement:

- The Security Requirements defined at the root level are the default requirements for all operations and servers.
- If Security Requirements are defined at the Operation level, then those Requirements, and not the ones defined at the OpenAPI level, shall be used with that operation.
- An empty set of Security Requirements at the Operation level indicates that there are no security requirements for that operation.

Note: this allows operations to opt-out of security requirements defined at the OpenAPI level.

A.2.5. A.2.4 Parameters

The following observations apply for WFS 3.0 parameters:

1. WFS 3.0 does not use cookies.
2. Query parameters follow common Web practice
3. Header parameters are restricted to custom headers
4. For path parameters, the name of the parameter must match the name of the variable in the path template in the path object

Parameters are defined at the Path Item and Operation level. Parameters defined at the Path Item level must apply to all operations under that Path item. These parameters may be modified at the Operation level but they may not be removed.

A.2.6. A.2.5 Testable Paths

A testable path is a path which corresponds to one of the paths defined in the WFS 3.0 specification. There are three alternatives for making this determination:

1. The path URI matches – this is the simplest approach but may be subject to error
2. Use mandatory tags in the tags field of the Operation Object
3. Use standardized operation ids for the operationId field of the Operation Object

A testable path is validated against the rules for that path. At a minimum that includes:

1. Building a list of all parameters which are defined in the standard
2. Validate that the mandatory parameters are present and required
3. Validate type, format, etc. for each parameter in the list.
4. Validate that there are no mandatory parameters which are not on the list.

A.3. A.3 Requirements Trace Matrix

Requirement 1: API Landing Page Operation The server SHALL support the HTTP GET operation at the path /. Tests: A.4.2.1
Requirement 2: API Landing Page Response A successful execution of the operation SHALL be reported as a response with a HTTP status code 200. The content of that response SHALL be based upon the OpenAPI 3.0 schema root.yaml and include at least links to the following resources: - /api (relation type 'service') - /conformance (relation type 'conformance') - /collections (relation type 'data') Tests: A.4.2.2
Requirement 3: API Definition Operation The server SHALL support the HTTP GET operation at the path /api. Tests: A.4.2.3
Requirement 4: API Definition Response A successful execution of the operation SHALL be reported as a response with a HTTP status code 200. The server SHALL return an API definition document. Tests: A.4.2.3, A.4.2.4, A.4.4.1
Requirement 5: Conformance Class Operation The server SHALL support the HTTP GET operation at the path /conformance. Tests: A.4.4.2
Requirement 6: Conformance Class Response A successful execution of the operation SHALL be reported as a response with a HTTP status code 200. The content of that response SHALL be based upon the OpenAPI 3.0 schema req-classes.yaml and list all WFS 3.0 requirements classes that the server conforms to. Tests: A.4.4.3
Requirement 7: HTTP 1.1 The server SHALL conform to HTTP 1.1. If the server supports HTTPS, the server SHALL also conform to HTTP over TLS. Tests: A.4.1.1

Requirement 8: Coordinate Reference Systems
Unless the client explicitly requests a different coordinate reference system, all spatial geometries SHALL be in the coordinate reference system http://www.opengis.net/def/crs/OGC/1.3/CRS84 (WGS84 longitude/latitude)
Tests: A.4.1.2

Requirement 9: Feature Collections Metadata Operation
The server SHALL support the HTTP GET operation at the path /collections.
Tests: A.4.4.4

Requirement 10: Feature Collections Metadata Response
A successful execution of the operation SHALL be reported as a response with a HTTP status code 200. The content of that response SHALL be based upon the OpenAPI 3.0 schema content.yaml.
Tests: A.4.4.5

Requirement 11: Feature Collections Metadata 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 server (relation: alternate).
All links SHALL include the rel and type link parameters.
Tests: A.4.4.5

Requirement 12: Feature Collections Metadata Items
For each feature collection in this distribution of the dataset, an item SHALL be provided in the property collections.
Tests: A.4.4.5, A.4.4.6

Requirement 13: Feature Collections Metadata Items Links
For each feature collection in this distribution of the dataset, the links property of the collection SHALL include an item for each supported encoding with a link to the collection resource (relation: item).
All links SHALL include the rel and type properties.
Tests: A.4.4.6

Requirement 14: Feature Collections Metadata Extent
For each feature collection, the extent property, if provided, SHALL be a bounding box that includes all spatial and temporal geometries in this collection.
If a feature has multiple properties with spatial or temporal information, it is the decision of the server whether only a single spatial or temporal geometry property is used to determine the extent or all relevant geometries.
Tests: A.4.4.6

Requirement 15: Feature Collection Metadta Operation
The server SHALL support the HTTP GET operation at the path /collections/{collectionId}.
collectionId is the `id` property in the feature collections metadata (JSONPath: $.collections[*].id).
Tests: A.4.4.7

Requirement 16: Feature Collection Metadata Response
A successful execution of the operation SHALL be reported as a response with a HTTP status code 200. The content of that response SHALL be the same as the content for this feature collection in the /collections response.
Tests: A.4.4.8

Requirement 17: Feature Collection Operation
For every feature collection identified in the metadata about the feature collection (path /), the server SHALL support the HTTP GET operation at the path /collections/{collectionId}/items where {collectionId} is the `id` property in the feature collections metadata (JSONPath: $.collections[*].id).
Tests: A.4.4.9

Requirement 18: Feature Collection Operation Limit Parameter
Each feature collection operation SHALL support a parameter limit with the following characteristics (using an OpenAPI Specification 3.0 fragment):
Tests: A.4.4.11

Requirement 19: Feature Collection Operation Limit Parameter Response
The response SHALL not contain more features than specified by the optional limit parameter. If the API definition specifies a maximum value for limit parameter, the response SHALL not contain more features than this maximum value.
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.
Tests: A.4.4.11

Requirement 20: Feature Collection Operation BoundingBox Parameter
Each feature collection operation SHALL support a parameter bbox with the following characteristics (using an OpenAPI Specification 3.0 fragment):
Tests: A.4.4.12

Requirement 21: Feature Collection Operation BoundingBox Parameter Response
Only features that have a spatial geometry that intersects the bounding box SHALL be part of the result set, if the bbox parameter is provided.
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
- Lower left corner, coordinate axis 3 (optional)
- Upper right corner, coordinate axis 1
- Upper right corner, coordinate axis 2
- Upper right corner, coordinate axis 3 (optional)

The coordinate reference system of the values 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.
Tests: A.4.4.12

Requirement 22: Feature Collection Operation Time Parameter
Each feature collection operation SHALL support a parameter time with the following characteristics (using an OpenAPI Specification 3.0 fragment):
Tests: A.4.4.13

Requirement 23: Feature Collection Operation Time Parameter Response
Only features that have a temporal geometry that intersects the timestamp or time period SHALL be part of the result set, if the time parameter is provided.
The temporal information is either a date-time or a period string that adheres to RFC3339.
If a feature has multiple temporal properties, it is the decision of the server whether only a single temporal property is used to determine the extent or all relevant temporal properties.
Tests: A.4.4.13

Requirement 24: Feature Collection Response
A successful execution of the operation SHALL be reported as a response with a HTTP status code 200.
Tests: A.4.4.10
The response SHALL only include features selected by the request.

Requirement 25: Feature Collection Response Links
A 200-response SHALL include the following links:
- a link to this response document (relation: self),
- a link to the response document in every other media type supported by the service (relation: alternate).
Tests: A.4.4.10

Requirement 26: Feature Collection Response Links Parameters
All links SHALL include the rel and type link parameters.
Tests: A.4.4.10

Requirement 27: Feature Collection Response timeStamp
If a property timeStamp is included in the response, the value SHALL be set to the time stamp when the response was generated.
Tests: A.4.4.10

Requirement 28: Feature Collection Response numberMatched
If a property numberMatched is included in the response, the value SHALL be identical to the number of features in the feature collections that match the selection parameters like bbox, time or additional filter parameters.
A server MAY omit this information in a response, if the information about the number of matching features is not known or difficult to compute.
Tests: A.4.4.10

Requirement 29: Feature Collection Response numberReturned
If a property numberReturned is included in the response, the value SHALL be identical to the number of features in the response.
A server MAY omit this information in a response, if the information about the number of features in the response is not known or difficult to compute.
Tests: A.4.4.10

Requirement 30: Feature Operation
For every feature in a feature collection (path /collections/{collectionId}/items), the service SHALL support the HTTP GET operation at the path /collections/{collectionId}/items/{featureId}. The parameter {collectionId} is each `id` property in the feature collection metadata (JSONPath: $.collections[*].id). {featureId} is a local identifier of the feature.
Tests: A.4.4.14

Requirement 31: Feature Opearation Response A successful execution of the operation SHALL be reported as a response with a HTTP status code 200. Tests: A.4.4.15
Requirement 32: Feature Operation Response Links A 200-response SHALL include the following links in the response: - a link to the response document (relation: self), - a link to the response document in every other media type supported by the service (relation: alternate), and - a link to the feature collection that contains this feature (relation: collection). All links SHALL include the rel and type link parameters. Tests: A.4.4.15

A.4. A.4 Abstract Test

The Test Approach used in the WFS 3.0 Abstract Test Suite includes four steps:

1. Identify the test points
2. Verify that API descriptions of the test points comply with the WFS 3.0 standard
3. Verify that the micro-services at each test point behave in accordance with the WFS 3.0 standard.
4. Verify that the resources returned at each test point are in accordance with the WFS 3.0 standard and any referenced content standard.

Identification of test points is a new requirement with WFS 3.0. Since an API is not a Web Service, there may be RESTful endpoints advertised which are not intended to be targets of the compliance testing. Section A.4.2 describes the process for crawling the API Description document and extracting those URLs which should be tested as well as the path(s) they should be tested with. The concatenation of a Server URL with a path forms a test point.

Section A.4.3 describes how the test points are exercised to determine compliance with the WFS 3.0 standard.

A.4.1. A.4.1 General Tests

A.4.1.1 HTTP 1.1

a) Test Purpose:

Validate that the WFS services advertised through the API conform with HTTP 1.1.

b) Pre-conditions:

none

c) Test Method:

1. All compliance tests shall be configured to use the HTTP 1.1 protocol exclusively.

d) References:

Requirement 7

A.4.1.2 Coordinate Reference Systems

a) Test Purpose:

Validate that all spatial geometries provided through a WFS service are in the CRS84 spatial reference system unless otherwise requested by the client.

b) Pre-conditions:

none

c) Test Method:

1. Do not specify a coordinate reference system in any request. All spatial data should be in the CRS84 reference system.
2. Validate retrieved spatial data using the CRS84 reference system.

d) References:

Requirement 8

A.4.2. A.4.2 Retrieve the API Description

A.4.2.1 Landing Page Retrieval

a) Test Purpose:

Validate that a landing page can be retrieved from the expected location.

b) Pre-conditions:

- A URL to the server hosting the landing page is known.
- The test client can authenticate to the server.
- The test client has sufficient privileges to access the landing page.

c) Test Method:

1. Issue an HTTP GET request to the URL {root}/
2. Validate that a document was returned with a status code 200
3. Validate the contents of the returned document using test A.4.2.2

d) References:

Requirement 1

A.4.2.2 Landing Page Validation

a) Test Purpose:

Validate that the landing page complies with the require structure and contents.

b) Pre-conditions:

- The landing page has been retrieved from the server

c) Test Method:

1. Validate the landing page against the root.yaml schema
2. Validate that the landing page includes a “service” link to API Definition
3. Validate that the landing page includes a “conformance” link to the conformance class document
4. Validate that the landing page includes a “data” link to the WFS contents.

d) References:

Requirement 2

A.4.2.3 OpenAPI Document Retrieval

Note: The URI for the API definition is provided through the landing page. However, that does not mean that the API definition resides on the same server as the landing page. Test clients should be prepared for a WFS 3.0 implementation which is distributed across multiple servers.

a) Test Purpose:

Validate that the API Definition document can be retrieved from the expected location.

b) Pre-conditions:

- A URL to the server hosting the API Definition document is known.
- The test client can authenticate to the server.
- The test client has sufficient privileges to assess the API Definition document.

c) Test Method:

1. Issue an HTTP GET request to the URL {server}/api
2. Validate that a document was returned with a status code 200
3. Validate the contents of the returned document using test A.4.2.4

d) References:

Requirements 3 and 4

A.4.2.4 API Definition Validation

a) Test Purpose:

Validate that the API Definition page complies with the require structure and contents.

b) Pre-conditions:

- The API Definition document has been retrieved from the server

c) Test Method:

1. Validate the API Definition document against the OpenAPI 3.0 schema
2. Identify the Test Points as described in test A.4.3
3. Process the API Definition document as described in test A.4.4

d) References:

Requirement 4

A.4.3. A.4.3 Identify the Test Points

Identification of the test points is a pre-condition to performing a compliance test. Thisprocess starts with A.4.3.1.

A.4.3.1 Identify Test Points:

a) Purpose:

To identify the test points associated with each Path in the OpenAPI document

b) Pre-conditions:

- An OpenAPI document has been obtained
- A list of URLs for the servers to be included in the compliance test has been provided
- A list of the paths specified in the WFS 3.0 specification

c) Method:

FOR EACH paths property in the OpenAPI document If the path name is one of those specified in the WFS 3.0 specification Retrieve the Server URIs using A.4.3.2. FOR EACH Server URI Concatenate the Server URI with the path name to form a test point. Add that test point to the list.

d) References:

None

A.4.3.2 Identify Server URIs:

a) Purpose:

To identify all server URIs applicable to an OpenAPI Operation Object

b) Pre-conditions:

- Server Objects from the root level of the OpenAPI document have been obtained
- A Path Item Object has been retrieved
- An Operation Object has been retrieved
- The Operation Object is associated with the Path Item Object
- A list of URLs for the servers to be included in the compliance test has been provided

c) Method:

1) Identify the Server Objects which are in-scope for this operation

- IF Server Objects are defined at the Operation level, then those and only those Server Objects apply to that Operation.
- IF Server Objects are defined at the Path Item level, then those and only those Server Objects apply to that Path Item.
- IF Server Objects are not defined at the Operation level, then the Server Objects defined for the parent Path Item apply to that Operation.
- IF Server Objects are not defined at the Path Item level, then the Server Objects defined for the root level apply to that Path.
- IF no Server Objects are defined at the root level, then the default server object is assumed as described in the OpenAPI specification.

2) Process each Server Object using A.4.3.3.

3) Delete any Server URI which does not reference a server on the list of servers to test.

d) References:

None

A.4.3.3 Process Server Object:

a) Purpose:

To expand the contents of a Server Object into a set of absolute URIs.

b) Pre-conditions:

- A Server Object has been retrieved

c) Method:

Processing the Server Object results in a set of absolute URIs. This set contains all of the URIs that can be created given the URI template and variables defined in that Server Object.

1. If there are no variables in the URI template, then add the URI to the return set.
2. For each variable in the URI template which does not have an enumerated set of valid values:
    - generate a URI using the default value,
    - add this URI to the return set,
    - flag this URI as non-exhaustive
3. For each variable in the URI template which has an enumerated set of valid values:
    - generate a URI for each value in the enumerated set,
    - add each generated URI to the return set.
4. Perform this processing in an iterative manner so that there is a unique URI for all possible combinations of enumerated and default values.
5. Convert all relative URIs to absolute URIs by rooting them on the URI to the server hosting the OpenAPI document.

d) References:

None

A.4.4. A.4.4 Processing the OpenAPI Document

A.4.4.1 Validate /api path

a) Test Purpose:

Validate API definition provided through the /api path it the athoritative definition of this API. Validate that this resource exists at the expected location and that it complies with the appropirate schema.

b) Pre-conditions:

- A URL to the server hosting the API definition document is known

c) Test Method:

1. Issue an HTTP GET request to the URL {server}/api
2. Validate that a document was returned with a status code of 200
3. Validate the returned document against the OpenAPI 3.0 schema

d) References:

Requirement 4

A.4.4.2 Validate Conformance Operation

a) Test Purpose:

Validate that Conformance Operation behaves as required.

b) Pre-conditions:

- Path = /conformance

c) Test Method:

DO FOR each /conformance test point

1. Issue an HTTP GET request using the test point URI
2. Go to test A.4.4.3.

d) References:

Requirement 5

A.4.4.3 Validate Conformance Operation Response

a) Test Purpose:

Validate the Response to the Conformance Operation.

b) Pre-conditions:

- Path = /conformance
- A Conformance document has been retrieved

c) Test Method:

1. Validate the retrieved document against the classes.yaml schema.
2. Record all reported compliance classes and associate that list with the test point. This information will be used in latter tests.

d) References:

Requirement 6

A.4.4.4 Validate the Feature Collections Metadata Operation

a) Test Purpose:

Validate that the Feature Collections Metadata Operation behaves as required

b) Pre-conditions:

- Path = /collections

c) Test Method:

DO FOR each /collections test point

- Issue an HTTP GET request using the test point URI
- Go to test A.4.4.5

d) References:

Requirement 9

A.4.4.5 Validate the Feature Collections Metadata Operation Response

a) Test Purpose:

Validate that response to the Feature Collection Metadata Operation.

b) Pre-conditions:

- A Feature Collection Metadata document has been retrieved

c) Test Method:

1. Validate the retrieved document against the content.yaml schema.
2. Validate that the retrieved document includes links for:
    - Itself
    - Alternate encodings of this document in every other media type as identified by the compliance classes for this server.
3. Validate that each link includes a rel and type parameter
4. Validate that the returned document includes a collections property for each collection in the dataset.
5. For each collections property, validate the metadata for that collection using test A.4.4.6

d) References:

Requirements 10, 11, and 12

A.4.4.6 Validate a Collections Metadata document

a) Test Purpose:

Validate a Collections Metadata document.

b) Pre-conditions:

- A Collection metadata document has been retrieved.

c) Test Method:

1. Validate the collection metadata against the collectionInfo.yaml schema

2. Validate that the collection metadata document includes links to the collection for each supported media type as identified by the compliance classes for this server.
3. Validate that each link includes a rel and type parameter
4. Validate the extent property if it is provided
5. Retrieve the collection using the id property and test A.4.4.7.

d) References:

Requirement 12, 13, 14

A.4.4.7 Validate the Feature Collection Metadata Operation

a) Test Purpose:

Validate that the Feature Collection Metadata Operation behaves as required

b) Pre-conditions:

- A feature collection name is provided by test A.4.4.6
- Path = /collections/{collectionId}

c) Test Method:

DO FOR each /collections{collectionId} test point

- Issue an HTTP GET request using the test point URI
- Go to test A.4.4.8

d) References:

Requirement 15

A.4.4.8 Validate the Feature Collection Metadata Operation Response

a) Test Purpose:

Validate that response to the Feature Collection Metadata Operation.

b) Pre-conditions:

- A Feature Collection Metadata document has been retrieved

c) Test Method:

1. Validate the retrieved document against the collectionInfo.yaml schema.
2. Validate that this is the same document as that processed in Test A.4.4.6

d) References:

Requirement 16

A.4.4.9 Validate the Get Features Operation

a) Test Purpose:

Validate that the Get Features Operation behaves as required.

b) Pre-conditions:

- A feature collection name is provided by test A.4.4.6
- Path = /collections/{collectionId}/items

c) Test Method:

DO FOR each /collections{collectionId}/items test point

- Issue an HTTP GET request using the test point URI
- Go to test A.4.4.10

d) References:

Requirement 17

A.4.4.10 Validate the Get Features Operation Response

a) Test Purpose:

Validate the Get Feature Operation Response.

b) Pre-conditions:

- A collection of Features has been retrieved

c) Test Method:

1. Validate the structure of the response as follows:
    - For HTML use Human inspection
    - For GeoJSON use featureCollectionGeoJSON.yaml
    - For GML use featureCollectionGML.yaml
2. Validate that the following links are included in the response document:
    - To itself
    - Alternate encodings of this document in every other media type as identified by the compliance classes for this server.
3. Validate that each link includes a rel and type parameter.
4. If a property timeStamp is included in the response, validate that it is close to the current time.
5. If a property numberReturned is included in the response, validate that the number is equal to the number of features in the response.
6. If a property numberMatched is included in the response, iteratively follow the next links until

no next link is included and count the aggregated number of features returned in all responses during the iteration. Validate that the value is identical to the numberMatched stated in the initial response.

d) References:

Requirements 24, 25, 26, 27, 28 and 29

A.4.4.11 Limit Parameter

a) Test Purpose:

Validate the proper handling of the limit parameter.

b) Pre-conditions:

- Tests A.4.4.9 and A.4.4.10 have completed successfully.

c) Test Method:

1. Verify that the OpenAPI document correctly describes the limit parameter for the Get Features operation.
2. Repeat Test A.4.4.9 using different values for the limit parameter.
3. For each execution of Test A.4.4.9, repeat Test A.4.4.10 to validate the results.

d) References:

Requirements 18 and 19

A.4.4.12 Bounding Box Parameter

a) Test Purpose:

Validate the proper handling of the bbox parameter.

b) Pre-conditions:

- Tests A.4.4.9 and A.4.4.10 have completed successfully.

c) Test Method:

1. Verify that the OpenAPI document correctly describes the bbox parameter for the Get Features operation.
2. Repeat Test A.4.4.9 using different values for the bbox parameter. These should include test cases which cross the meridian, equator, 180^{O} longitude, and polar regions.
3. For each execution of Test A.4.4.9, repeat Test A.4.4.10 to validate the results.

d) References:

Requirements 20 and 21

A.4.4.13 Time Parameter

a) Test Purpose:

Validate the proper handling of the time parameter.

b) Pre-conditions:

- Tests A.4.4.9 and A.4.4.10 have completed successfully.

c) Test Method:

1. Verify that the OpenAPI document correctly describes the time parameter for the Get Features operation.
2. Repeat Test A.4.4.9 using different values for the time parameter.
3. For each execution of Test A.4.4.9, repeat Test A.4.4.10 to validate the results.

d) References:

Requirements 22 and 23

A.4.4.14 Get Feature Operation

a) Test Purpose:

Validate that the Get Feature Operation behaves as required.

b) Pre-conditions:

- A feature collection name is provided by test A.4.4.6
- A feature identifier is provided by test A.4.4.10
- Path = /collections/{collectionId}/items/(featureId} where {featureId} = the feature identifier

c) Test Method:

DO FOR each /collections{collectionId}/items/{featureId} test point

- Issue an HTTP GET request using the test point URI
- Go to test A.4.4.15

d) References:

Requirement 30

A.4.4.15 Validate the Get Feature Operation Response

a) Test Purpose:

Validate the Get Feature Operation Response.

b) Pre-conditions:

- The Feature has been retrieved from the server.

c) Test Method:

1. Validate the structure of the response as follows:
    - For HTML use Human Inspection
    - For GeoJSON use featureGeoJSON.yaml
    - For GML use featureGML.yaml
2. Validate that the following links are included in the response document:
    - To itself
    - To the Feature Collection which contains this Feature
    - Alternate encodings of this document in every other media type as identified by the compliance classes for this server.
3. Validate that all links include the rel and type link parameters.

d) References:

Requirements 31 and 32

Annex B: Examples (Informative)

B.1. Example Landing Pages

Example 6. JSON Landing Page

```
{
  "links": [
    { "href": "http://data.example.org/",
      "rel": "self", "type": "application/json", "title": "this document" },
    { "href": "http://data.example.org/api",
      "rel": "service", "type": "application/openapi+json;version=3.0", "title":
"the API definition" },
    { "href": "http://data.example.org/conformance",
      "rel": "conformance", "type": "application/json", "title": "OGC conformance
classes implemented by this API" },
    { "href": "http://data.example.org/collections",
      "rel": "data", "type": "application/json", "title": "Metadata about the
resource collections" }
  ]
}
```

B.2. Conformance Examples

Example 7. Conformance Response

This example response in JSON is for an OGC API Features that supports OpenAPI 3.0 for the API definition and HTML and GeoJSON as encodings for resources.

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

B.3. Collections Metadata Examples

Example 8. Collection metadata response document

This feature collection metadata example response in JSON is for a dataset with a single collection "buildings". It includes links to the collection resource in all formats that are supported by the API (link relation type: "items").

Representations of the metadata resource in other formats are referenced using link relation type "alternate".

An additional link is to a GML application schema for the dataset - using:https://www.iana.org/assignments/link-relations/link-relations.xhtml[link relation type] "describedBy".

Finally there are also links to the license information for the building data (using:https://www.iana.org/assignments/link-relations/link-relations.xhtml[link relation type] "license").

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

```
{
  "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://data.example.org/collections/buildings/items",
          "rel": "items", "type": "application/geo+json",
          "title": "Buildings" },
        { "href": "http://example.org/concepts/building.html",
          "rel": "describedBy", "type": "text/html",
          "title": "Feature catalogue for buildings" }
      ]
    }
  ]
}
```

Annex C: Revision History

Date	Release	Editor	Primary clauses modified	Description
2017-10-09	3.0.0-SNAPSHOT	C. Portele	all	initial version
2017-10-11	3.0.0-SNAPSHOT	C. Portele	all	changes discussed in SWG/PT call on 2017-10-09
2017-12-13	3.0.0-SNAPSHOT	C. Portele	all	address issues #2, #5, #6, #7, #8, #14, #15, #19
2018-01-22	3.0.0-SNAPSHOT	C. Portele	7	add description of the UML diagram
2018-02-01	3.0.0-SNAPSHOT	C. Portele	2, 3, 5, 7	add links to recent issues on GitHub; address issues #31, #32
2018-02-11	3.0.0-SNAPSHOT	C. Portele	2, 6, 7, 8	address issue #25
2018-02-27	3.0.0-SNAPSHOT	C. Portele	all	address issues #3, #9, #12, #22, #23, #24, #44; add links to issues #41, #45, #46, #47
2018-03-04	3.0.0-SNAPSHOT	T. Schaub	7, B	JSON schema fixes #54, #55
2018-03-12	3.0.0-SNAPSHOT (for ISO NWIP)	C. Portele	all	Updates after the WFS 3.0 Hackathon #59, #61, #62, #63, #64, #69, #72, #77, #78; resolve #4; editorial edits
2018-03-15	3.0.0-SNAPSHOT	J. Amara	7	Uniqueness of feature id #83
2018-03-21	3.0.0-SNAPSHOT	I. Rinne	7	Clarified the requirement /req/core/crs84 #92
2018-03-28	3.0.0-SNAPSHOT	C. Portele	3, 4, 7	Temporal support #57, bbox no longer restricted to CRS84 #60, clarify 'collection' #86, clarify feature id constraints #84
2018-04-02	3.0.0-SNAPSHOT	C. Portele	7, B	Clarify 'item' links #81, clean up OpenAPI example in Annex B
2018-04-03	3.0.0-SNAPSHOT	C. Portele	4 to 9	Clean-up asciidoc #100
2018-04-04	3.0.0-SNAPSHOT	P. Vretanos, C. Portele	8.4, 8.5, C	Clarify XML encoding #58
2018-04-05	3.0.0-SNAPSHOT	C. Heazel	A	Initial version of the Abstract Test Suite #112
2018-04-05	3.0.0-SNAPSHOT	C. Portele	C	Fix axis order in example #113
2018-04-07	3.0.0-SNAPSHOT	C. Portele	7, 9, 10	Add HTTP status code guidance #105, add warning about OpenAPI media type #117
2018-04-07	3.0.0-SNAPSHOT	C. Reed, C. Portele	all	Edits after review #119
2018-04-07	**3.0.0-draft.1**	C. Portele	iv, v	First draft release

Date	Release	Editor	Primary clauses modified	Description
2019-02-14	3.0.0-SNAPSHOT	C. Portele, C. Holmes	all	Bugfixes #149 and #176, change rel=item to rel=items #175, use {collectionId}, {featureId} and `id` consistently #171

Annex D: 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