Draft OGC API-Common - Part 1

Core

Open Geospatial Consortium

Submission Date: <yyyy-mm-dd>

Approval Date: <yyyy-mm-dd>

Publication Date: 2020-02-03

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

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

Version: 0.0.6

Category: OGC® Implementation Standard

Editor: Charles Heazel

Draft OGC API-Common - Part 1: Core

Document type:	OGC® Implementation Standard
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 across multiple OGC Web API standards. The purpose of this standard is to document these common practices. This standard also serves as a common foundation upon which all OGC Web API Standards will be built.

An OGC API provides a lightweight interface to access one or more resources. The resources addressed by OGC Web APIs fall into three categories; Foundation Resources, Spatial Resources, and Information Resources. These Resource Catagories are described in section 8, Requirement Class Core.

The API-Common standard defines resources and access paths for Foundation Resources. These resources and access paths are supported by all conformant OGC Web APIs. These are listed in Table 1.

In addition, the API-Common Standard defines an `stub` set of resources and access paths for Spatial Resources. This `stub` is sufficient to start the client down the path to spatial resource discovery. Developers of OGC Web API standards extend this `stub` with details specific to the resources they intend to expose. Since not all OGC Web APIs will host Spatial Resources, this capability is documented in a separate `Collections` Requirements Class. Only APIs which advertise conformance with the `Collections` Conformance Class have to support this capability.

The `Collections` paths and resources are listed in Table 2

Table 1. Overview of Foundation Resources

Resource	Path	HTTP Method	Document Reference
Landing page	`/`	GET	API Landing Page
API definition	`/api`	GET	API Definition
Conformance classes	`/conformance`	GET	Declaration of Conformance Classes

Table 2. Overview of Spatial Resources

Collections metadata	`/collections`	**GET**	**Collections Metadata**
Collection information	`/collections/{collectionId}`	GET	Collection Information
Collection resource	`/collections/{collectionId}/items`	GET	Collection Resource

The resources identified in Table 1 and Table 2 primarily support Discovery operations. Discovery operations allow clients the interrogate the API to determine its capabilities and retrieve information (metadata) about the hosted resources. 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 standard 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 standard addresses two fundamental operations; Discovery and Query.

Discovery operations allow the API to be interrogated to determine its capabilities and retrieve information (metadata) about the hosted resources. 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. These criteria are 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 claim conformance are specified in the OGC Compliance Testing Policies and Procedures and the OGC Compliance Testing web site.

The one Standardization Target for this standard is Web APIs.

OGC API-Common provides a common foundation for OGC API standards. It is anticipated that this standard will only be implemented through inclusion in other standards. Therefore, all the relevant abstract tests in Annex A shall be included or referenced in the Abstract Test Suite in each separate standard that normatively references this standard.

This standard identifies five conformance classes. The conformance classes implemented by an OGC API are advertised through the /conformance path on the landing page. Each conformance class is defined by one requirements class. The tests in Annex A are organized by Requirements Class. So an implementation of the *Core* conformance class must pass all tests specified in Annex A 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 and in OGC managed API extensions. Those standards and extensions build on the API-Common foundation to provide the full functionality required of any OGC 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 organization 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.

- Encodings
  - HTML
  - GeoJSON

Neither the *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 is mandatory. An implementor of the *API-Common* standard 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. 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.

- ISO 8601-1:2019, **Date and time - Representations for information interchange - Part 1: Basic rules**
- ISO 19107:2019, **Geographic information Spatial Schema**
- ISO 19108:2002/Cor 1:2006, **Geographic information — Temporal schema**
- ISO 19111:2019, **Geographic information Spatial referencing by coordinates**
- Herring, J.: OGC 06-104r4,**OpenGIS® Implementation Standard for Geographic information - Simple feature access - Part 2: SQL option**, http://portal.opengeospatial.org/files/?artifact_id=25354
- van den Brink, L., Portele, C., Vretanos, P.: OGC 10-100r3, **Geography Markup Language (GML) Simple Features Profile**, http://portal.opengeospatial.org/files/?artifact_id=42729
- 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
- Berners-Lee, T., Fielding, R., Masinter, L: **IETF RFC 3896, Uniform Resource Identifier (URI): Generic Syntax**, http://tools.ietf.org/rfc/rfc3896.txt
- Fielding, R., Reschke, JSchaub: **IETF RFC 7231, Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content**, https://tools.ietf.org/rfc/rfc7231.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
- Nottingham, M.: **IETF RFC 8288, Web Linking**, http://tools.ietf.org/rfc/rfc8288.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 Web Services Common (OGC 06-121r9), which is based on the ISO/IEC Directives, Part 2, Rules for the structure and drafting of International Standards. In particular, the word “shall” (not “must”) is the verb form used to indicate a requirement to be strictly followed to conform to this standard.

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

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

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

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

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

- **Coverage**
  feature that acts as a function to return values from its range for any direct position within its spatiotemporal domain, as defined in OGC Abstract Topic 6 (OGC 09-146r6)
- **Dataset**
  collection of data, published or curated by a single agent, and available for access or download in one or more formats (DCAT)
- **Distribution**
  represents an accessible form of a **dataset** (DCAT)

  EXAMPLE: a downloadable file, an RSS feed or a web service that provides the data.
- **Executable Test Suite (ETS)**
  A set of code (e.g. Java and CTL) that provides runtime tests for the assertions defined by the ATS. Test data required to do the tests are part of the ETS (OGC 08-134)
- **Extent**
  The area covered by something. Within this document, we always imply spatial extent; e.g. size or shape that may be expresses using coordinates. (W3C/OGC Spatial Data on the Web Best Practice)
- **Feature**
  abstraction of real world phenomena [ISO 19101-1:2014]

Note	For those unfamiliar with the term 'feature', the explanations on Spatial Things, Features and Geometry in the W3C/OGC Spatial Data on the Web Best Practice document provide more detail.

- **Foundation Resources**
  those resources which are provided by every OGC API.
- **Geometry**
  An ordered set of *n*-dimensional points in a given coordinate reference system. (W3C/OGC Spatial Data on the Web Best Practice)
- **Information Resources**
  are non-spatial resources which support the operation of the API or the access and use of the Spatial Resources
- **Recommendation**
  expression in the content of a document conveying that among several possibilities one is recommended as particularly suitable, without mentioning or excluding others, or that a certain course of action is preferred but not necessarily required, or that (in the negative form) a certain possibility or course of action is deprecated but not prohibited (OGC 08-131r3)

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

- **Requirement**
  expression in the content of a document conveying criteria to be fulfilled if compliance with the document is to be claimed and from which no deviation is permitted (OGC 08-131r3)
- **Requirements Class**
  aggregate of all requirement modules that must all be satisfied to satisfy a conformance test class (OGC 08-131r3)
- **Requirements Module**
  aggregate of requirements and recommendations of a specification against a single standardization target type (OGC 08-131r3)
- **Spatial Resources**
  the resources which we usually think of as Geospatial Data.
- **Standardization Target**
  entity to which some requirements of a standard apply (OGC 08-131r3)

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

- **Spatial Thing**
  anything with spatial extent, (i.e. size, shape, or position) and is a combination of the real-world phenomenon and its abstraction. (W3C/OGC Spatial Data on the Web Best Practice)

- **Temporal Coordinate System**
  temporal reference system based on an interval scale on which distance is measured as a multiple of a single unit of time. [ISO 19108]
- **Temporal Position**
  location relative to a temporal reference system [ISO 19108]
- **Temporal Reference System**
  reference system against which time is measured [ISO 19108]
- **Temporal Thing**
  Anything with temporal extent, i.e. duration. e.g. the taking of a photograph, a scheduled meeting, a GPS time-stamped track-point. (W3C Basic Geo)

Chapter 6. Conventions

6.1. Identifiers

The normative provisions in this draft standard are denoted by the URI `http://www.opengis.net/spec/ogcapi-common-1/1.0`.

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

The following registered link relation types are used in this document.

- **alternate**: Refers to a substitute for this context.
- **collection**: The target IRI points to a resource which represents the collection resource for the context IRI.
- **describedBy**: Refers to a resource providing information about the link's context.
- **item**: The target IRI points to a resource that is a member of the collection represented by the context IRI.
- **license**: Refers to a license associated with this context.
- **self**: Conveys an identifier for the link's context.
- **service-desc**: Identifies service description for the context that is primarily intended for consumption by machines.
    - API definitions are considered service descriptions.
- **service-doc**: Identifies service documentation for the context that is primarily intended for human consumption.

In addition the following link relation types are used for which no applicable registered link relation type could be identified.

- **items**: Refers to a resource that is comprised of members of the collection represented by the link's context.
- **conformance**: Refers to a resource that identifies the specifications that the link's context conforms to.
- **data**: Indicates that the link's context is a distribution of a dataset that is an API and refers to the root resource of the dataset in the API.

Each resource representation includes an array of links. Implementations are free to add additional links for all resources provided by the API. For example, an **enclosure** link could reference a bulk download of a collection. Or a **related** link on a feature could reference a related feature.

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. HTTP URIs

This document does not restrict the lexical space of URIs used in the API beyond the requirements of the HTTP and URI Syntax IETF RFCs. If URIs include reserved characters that are delimiters in the URI subcomponent, these have to be percent-encoded. See Clause 2 of RFC 3986 for details.

6.5. Geometry

6.5.1. Spatial Geometry

Standardized concepts for spatial characteristics are needed in order to share geographic information between applications. Concepts for shape (geometry) are key. These concepts are standardized in *ISO 19107*.

The spatial geometry used in the OGC API-Common Standard is documented in the GML Simple Features Profile Standard. This Profile defines a subset of the ISO 19107 geometry which is aligned with the OGC Simple Features for SQL Standard. That geometry includes: Point, Curve (LineString), Surface (Polygon), MultiPoint, MultiCurve, and MultiSurface.

6.5.2. Temporal Geometry

Standardized concepts are also needed for temporal characteristics. The temporal geometry used in the API-Common Standard is defined in ISO 19108. Only a subset of this geometry is used. Specifically:

- TM-Instant - a point representing position in time
- TM-Period - one dimensional geometric primitive representing extent in time and bounded by two different temporal positions (TM-Instant)

6.6. Reference Systems

6.6.1. 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 location 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.

OGC API-Common does not mandate use of a specific coordinate reference system. However, if no CRS is specified, the following default coordinate reference systems apply.

- CRS84 - WGS 84 longitude and latitude without elevation
- CRS84h - WGS 84 longitude and latitude with elipsoid elevation

ISO 19111 is the normative standard for Coordinate Reference Systems.

6.6.2. Temporal Reference Systems

Temporal positions are measured relative to an underlying temporal reference system. The OGC API-Common Standard does not mandate a specific Temporal Reference System. However, it does establish a default which is:

- For date: the Gregorian Calendar
- For time: a 24 hour local or Coordinated Universal Time (UTC)

This convention reflects the recomendations of ISO 8601 as documented in ISO 19108. A different temporal reference system may be used where appropriate.

ISO 19108 is the normative standard for Temporal Reference Systems.

ISO 8601 is the normative standard for representations of dates and times.

6.7. API definition

6.7.1. General remarks

So that developers can more easily learn how to use the API, good documentation is essential for every 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 OGC 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, and so on that are specific to the API or the software tool used to implement the API.

6.7.2. 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 conformance 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 conformance class for API definitions that follow the OpenAPI specification 3.0. Conformance classes for additional API definition languages will be added as the OGC API landscape continues to evolve.

In this document, fragments of OpenAPI definitions are shown in YAML. This is because YAML is

easier to format than JSON and is typically used by OpenAPI editors.

6.7.3. 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 is 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 OGC 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.

6.7.4. Reusable OpenAPI components

Reusable components for OpenAPI definitions for an 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/ogcapi_common/1.0/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, technology has evolved. New Resource-Oriented APIs provide an alternative to 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 the OGC API approach to services that follows the current Web architecture. In particular, the recommendations as defined in 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.

In addition to 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 common core requirements and more resource specific capabilities. The OGC API-Common Standard specifies the core or *common* requirements that may be relevant to almost anyone who wants to build an API for spatial resources. Additional capabilities will be specified as extensions to the Common API as they are developed.
- Technologies that change more frequently are decoupled and specified in separate modules ("conformance 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 can 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, this server 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 current or future OGC Web API standards.

7.2. Navigation

OGC API Common Standard provides clients with two ways to access to a conformant API, direct and hypermedia.

A direct access client has been designed to work with an API-Common conformant API. Standardizing paths, resource types, parameters, etc. equip a client with most of the information needed to use a conformant API. These clients can quickly access resources with minimal time spent on API discovery.

A hypermedia access client needs little a-prioi information beyond a URL to the landing page and the abilty to identify and follow the links found there. A common web browser should be sufficient.

Navigation is addressed in more detail in the Core Requirements Class.

7.3. Encodings

This standard does not mandate any encoding or format. However, the API-Common Standard does provide Requirements Classes for encodings which are commonly used in OGC APIs. HTML is the standard encoding for Web content. As such, failure to implement the HTML Requirement Class should be the exception rather than the rule. Additional Requirements Classes are provided for encodings which are commonly used for spatial data on the web. These include the GeoJSON and GML Requirements Classes.

Chapter 8. Requirement Class "Core"

Requirements Class	
http://www.opengis.net/spec/ogcapi_common/1.0/req/core	
Target type	Web API
Dependency	RFC 2616 (HTTP/1.1)
Dependency	RFC 2818 (HTTP over TLS)
Dependency	RFC 8288 (Web Linking)

8.1. Overview

8.1.1. Resources

An OGC API provides a lightweight 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 their unique requirements.

Spatial Resources are the resources which we usually think of as Geospatial Data. They include Features, Coverages, and Images. 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. The approach specified in this standard 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 likelihood that they will integrate at run-time is greatly enhanced.

A more detailed discussion of modular APIs can be found in the API-Common Best Practices document.

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 starts from a landing page, selects a link on that page and 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 use a set of common link relationships. These link relationships are described in Table 3.

Table 3. Link Relations

Link Relation	Purpose
`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`

OGC API hyperlinks are defined using the following 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 knowledge of the path to the resource prior to attempting access. Typically this knowledge 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 landing page (path /).

The purpose of the landing page is to provide clients with a starting point for using the API. It provides the basic information needed to understand the purpose, structure, and capabilities of the API. The landing page also serves as the starting point for navigation. Any resource exposed through an API can be accessed using a path or link starting from the landing page.

Operation

Requirement 1	**/req/core/root-op**
A	The server SHALL support the HTTP GET operation at the path `/`.

Response

Requirement 2	**/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 Json schema.

landingPage.json

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

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

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 standard can be used by developers to understand API-Common, by software clients to connect to the server, and by development tools to support the implementation of servers and clients.

Operation

Requirement 3	**/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 4	/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 conformance classes it implements and conforms to.

Operation

Requirement 5	/req/core/conformance-op
A	The API SHALL support the HTTP GET operation at the path `/conformance`.

Response

Requirement 6	/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 the Collections section.

8.4. Information Resources

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

Information Resources can be 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 4 provides a mapping of the known Information Resource types to the standard where they are defined.

Table 4. Information Resource Types

Resource Type	API Standard
TBD	TBD

8.5. General Requirements

The following general requirements and recommendations 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 7	**/req/core/http**
A	Any OGC Web 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

NOTE ISSUE: Should we expand our explanation of HTTP status codes? See issues 75, and 73.

Table 5 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 5. Typical HTTP status codes

Status code	Description
`200`	A successful request.

Status code	Description
304	An entity tag was provided in the request and the resource has not 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.
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 authorized to perform the requested operation on the resource.
404	The requested resource does not exist on the server. For example, a path parameter had an incorrect value.
405	The request method is not supported. For example, a POST request was submitted, but the resource only supports GET requests.
406	Content negotiation failed. For example, the `Accept` header submitted in the request did not support any of the media types supported by the server for the requested resource.
500	An internal error occurred in the server.

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 status codes in addition to those listed in Table 5.

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

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

Recommendation 3	/rec/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

NOTE	Note to the Editor: Is this discussion of content negotiation sufficient? If not, what else do we need? See issues 57 and 28.

While the OAPI Common standard does not specify any mandatory encoding, the following encodings are recommended. See Clause 7 (Overview) for a discussion of this issue.

HTML encoding recommendation:

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

GeoJSON encoding recommndation:

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

Requirement `/req/core/http` implies that the encoding of a response is determined using content negotiation as specified in IETF RFC 7231.

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

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, such as to alternate representations of the same resource. This standard does not mandate how this mechanism shall be supported by a conformant 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, manipulating ("hack") 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. 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 features 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 API-Common uses WGS 84 longitude and latitude as the default coordinate reference system.

Requirement 8	**/req/core/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.

8.5.7. Link Headers

Recommendation 6	**/rec/core/link-header**
A	Links included in the payload of a response SHOULD also be included as `Link` headers in the HTTP response according to RFC 8288, Clause 3.

B	This recommendation does not apply when 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/ogcapi_common/1.0/req/collections	
Target type	Web API
Dependency	Requirements Class "OAPI Core"
Dependency	RFC 3339 (Date and Time on the Internet: Timestamps)

9.1. Overview

Spatial Resources are resources which we usually think of as Geospatial Data. They include Features and Coverages. This Conformance Class defines basic patterns for accessing Spatial Resources. Additional OGC API Standards have been or are being developed to address specific OGC 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 8. For Spatial Resources, hypermedia navigation is enabled through the links included in each schema defined by the Requirement Class defined in that Clause.

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 goal of ensuring 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 any 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 be collections on the same OGC API end-point.
3. Providing a tie point for OGC API modules to connect to and access spatial resources hosted by an OGC API.

Spatial Resources are exposed using the path template.

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

The resources returned from each node in this template are described in Table 6.

Table 6. Spatial Resource Paths

Path Template	Resource
/collections	Metadata describing the spatial collections available from this API.
/collections/{collectionId}	Metadata describing the collection with the unique identifier `{collectionId}`
/collections/{collectionId}/items	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.json JSON schema. Examples of collections metadata are provided in Collections Metadata Examples.

collections.json

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

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

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

Requirement 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.

Recommendation 7	**/rec/collections/rc-md-descriptions**

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

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.

The Collections Metadata should describe all of the collections accessible through the API. However, in some cases that is impractical. As long as they provide a way to retrieve the remaining metadata as well, developers have an option to only return a subset.

Permission 2	**/per/collections/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": "extent.json"},
        "itemType": {
            "type": "string",
            "default": "unknown"
            },
        "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 recommendations.

To support hypermedia navigation, the `links` property must be populated 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.

Recommendation 8	**/rec/core/rc-md-items-descriptions**
A	If external schemas or descriptions exist that provide additional information about the structure or semantics of the collection, a `200`-response SHOULD include links to each of those resources in the `links` property of the response (relation: `describedBy`).
B	The `type` link parameter SHOULD be provided for each link.

Additional requirements and recommendations apply to the `extent` property 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/collections/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, such as thermal or pressure ranges.
B	The Core only supports • Spatial extents in CRS84 or CRS84h and • Temporal extents in the Gregorian calendar These are the only *enum* values in extent.yaml).
C	Extensions 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 5).

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 purview 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 7.

Table 7. Information Resource Paths

Path Template	Resource
/collections	The root resource describing the spatial collections available 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 4 provides a mapping of the known Information Resource types to the standard where they are defined.

9.4. Parameter Modules

NOTE	ISSUE: Should we add a limit parameter. See GitHub issues 87, 83, and 82.
NOTE	ISSUE: Should we be more explicit about the scope of parameters and other elements of the API definition? See issues 67 and 88. Also see API-Coverages issue 53.

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)
E	The values for the CRS axis 1 and 2 SHALL be interpreted as WGS84 longitude/latitude (http://www.opengis.net/def/crs/OGC/1.3/CRS84) unless a different coordinate reference system is specified in a parameter `bbox-crs`.
F	The coordinate values SHALL be within the extent specified for the coordinate reference system.

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

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

For CRS84 the bounding box is in most cases the sequence of minimum longitude, minimum latitude, maximum longitude and maximum latitude. However, in cases where the box spans the anti-meridian (180th meridian) the first value (west-most box edge) is larger than the third value (east-most box edge).

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

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

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

D	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 2. A date-time

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

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

For resources with a temporal property that is a timestamp (like `lastUpdate` 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 3. Intervals

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

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

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

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

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

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

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

9.5. General Requirements

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

9.5.1. Coordinate Reference Systems (CRS)

As discussed in Chapter 9 of the W3C/OGC Spatial Data on the Web (SDW) 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. therefore, clearly stating the CRS rules for the coordinates is very important.

For the reasons discussed in the SDW Best Practice, OGC APIs use WGS84 longitude and latitude as the default CRS.

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

A	Unless the client explicitly requests a different coordinate reference system, all spatial geometries SHALL be in the CRS84 (WGS 84 longitude/latitude) CRS for geometries without height information and 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 CRS 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 CRS 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 four 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
- Geography Markup Language (GML), Simple Features Profile, Level 0
- Geography Markup Language (GML), Simple Features Profile, Level 2

None of these encodings are mandatory and an implementation of the Core requirements class may implement some, all, or none of them.

10.2. Requirement Class "HTML"

Geographic information that is only accessible in formats such as GeoJSON or GML have two issues:

- The data is not discoverable using Web crawlers and search engines,
- 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, this publication should be done in a way that enables users and search engines to discover and access all of the data.

This is discussed in detail in the W3C/OGC SDW Best Practice. Therefore, the OGC API-Common Standard recommends supporting HTML as an encoding.

Requirements Class	
http://www.opengis.net/spec/ogcapi_common/1.0/req/html	
Target type	Web API
Dependency	Requirements 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 with the caveat that the resource can be represented in GeoJSON for the intended use.

Requirements Class	
http://www.opengis.net/spec/ogcapi_common/1.0/req/geojson	
Target type	Web API
Dependency	Requirements Class "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 JSON Schema definitions are available at collections.yaml and collectionInfo.yaml.

These are generic schemas that do not include any application schema information about specific resource types or their properties.

10.4. Requirement Class GMLSF0

"Geography Markup Language (GML), Simple Features Profile, Level 0"

NOTE To be provided

10.5. Requirement Class GMLSF2

"Geography Markup Language (GML), Simple Features Profile, Level 2"

NOTE To be provided

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/ogcapi_common/1.0/req/oas30	
Target type	Web API
Dependency	Requirements 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 APIs that, for example, are access-controlled (see Security), support web cache validation, support 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 4. 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

OpenAPI uses two constructs to describe the security features of an API; Security Requirements and Security Schemes. Security Requirements are packaged in an array. Only one of the Security Requirements in the array must be met in-order to authorize a request. Security Requirements are associated with one or more Security Schemes. Each Security Scheme describes a security control (ex. HTTP authentication). All of the security schemes associated with a Security Requirement must be satisfied in order for that Security Requirement to be met.

Security Requirements can be defined on following levels:

- Root - applicable to the whole API unless overriden
- Operation - only applicable to this operation. Overrides any requirements defined at the Root level.

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.

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.

11.5. Using OpenAPI (Informative)

The following section describes the main components of an OpenAPI document and how they should be used to construct API requests. The authoritative source is the OpenAPI 3.0 standard available here.

11.5.1. OpenAPI Document (root)

The OpenAPI document (Root) contains descriptive information about the API and serves as the root for the other parts of the document.

11.5.2. Paths

All API resources are accessed through a path. The Paths field of the OpenAPI document defines all of the paths available through this API. The Paths field is a collection of Paths Objects. Each Paths Object includes the URL or URL template for this path, any Server Objects specific to this Path, Parameters which are applicable to this Path, and an Operation Object for each of the HTTP Verbs applicable to this Path.

11.5.3. Operations

Operation Objects provide the details needed to create an HTTP request and response. Specifically, they provide definitions of the request message (including parameters) and all possible responses. In addition, they define any operation specific Server Objects or Security Requirements.

11.5.4. Parameters

Parameter Objects describe parameters which can be used in an API. These objects provide the parameter name, where it is passed (query, header, path, or cookie), and a detailed description of its' structure (if needed).

Parameter Objects can be defined at the following levels:

- Path - applicable to all operations on this path.
- Operation - only applicable to this operation. Overrides any parameters defined at the Path level which have the same name.

11.5.5. Servers

An API is not restricted to a single server. Furthermore, the set of valid servers may be different for different sections of the API. An OpenAPI Server Object describes a single server. Most important, it provides the URL to that server. Server Objects are grouped into arrays. These arrays provide a list of the servers which can be used to access a section of the API.

Example Array of Server Objects

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

Server Objects can be defined on following levels:

- Root - applicable to the whole API unless overriden
- Path - applicable to all operations on this path. Servers defined at the Rool level are still valid.
- Operation - only applicable to this operation. Overrides any servers defined at the Path or Root level.

Building URLs

An OpenAPI document can describe a large number of URLs. Extracting all of the URLs is a non-trivial task. The OpenAPI objects used to construct URLs are:

- Server Objects (URL template for the root and variables to populate it)
- Paths Objects (URL template for the path and parameters)
- Operation Objects (including parameters)

They are organized as follows:

{Server Object}/{Path Object}/?{Parameters}

Server Objects may be found at the OpenAPI document, Path Object, and Operation Object level.

Given this potentially large number of servers, how do you create the valid paths?

We can assume that the authors of a OAS document are not doing it for their personal enjoyment. Therefore, if a Server Object is included, there must be a reason for its' presence. So the Server Objects with the most restrictive scope are the ones we should use. Clients should look for Server Objects in the following order:

1. The Operation Object,
2. Then Path Item,
3. The root.

The first scope where a Server Object is found dictates the behavior completely.

```
IF Server Objects are supplied
    THEN save them for latter
    ELSE create a Server Object for the host of the OpenAPI document
DO FOR each Path
    IF Server Objects are included, THEN
        Use them instead of those previously identified
    IF Parameter Objects are included, THEN
        save them for latter
    DO FOR Each Operation
        IF Server Objects are included, THEN
            Use them instead of those previously identified
        IF Parameter Objects are included THEN
            IF this parameter was previously defined
                THEN replace the previous definition
                ELSE add this parameter to the set.
        DO FOR Each Server Object
            Extract all URL roots
            DO FOR each URL root
                Concatenate the URL root and Path to create a working URL
                Concatenate the working URL with the Parameters
                Save the completed URL for future use
            CONTINUE
        DONE
    DONE
DONE
```

Chapter 12. Media Types

JSON media types that would typically be used in an 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 an 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)

A.1. Introduction

OGC Web APIs are not a Web Services in the traditional sense. Rather, they define the behavior and content of a set of Resources exposed through a Web Application Programing Interface (Web API). Therefore, an API may expose resources in addition to those defined by the standard. A test engine must be able to traverse the API, identify and validate test points, and ignore resource paths which are not to be tested.

A.2. Conformance Class Core

Conformance Class	
http://www.opengis.net/spec/ogcapi-common/1.0/conf/core	
Target type	Web API
Requirements Class	http://www.opengis.net/spec/ogcapi_common/1.0/req/core

A.2.1. General Tests

HTTP

Abstract Test 1	**/ats/core/http**
Test Purpose	Validate that the resource paths advertised through the API conform with HTTP 1.1 and, where approprate, TLS.
Requirement	/req/core/http
Test Method	1. All compliance tests shall be configured to use the HTTP 1.1 protocol exclusively. 2. For APIs which support HTTPS, all compliance tests shall be configured to use HTTP over TLS (RFC 2818) with their HTTP 1.1 protocol.

A.2.2. Landing Page {root}/

Abstract Test 2	**/ats/core/root-op**
Test Purpose	Validate that a landing page can be retrieved from the expected location.

Requirement	/req/core/root-op
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 /ats/core/root-success.

Abstract Test 3	**/ats/core/root-success**
Test Purpose	Validate that the landing page complies with the require structure and contents.
Requirement	/req/core/root-success
Test Method	Validate the landing page for all supported media types using the resources and tests identified in Table 8 For formats that require manual inspection, perform the following: a. Validate that the landing page includes a "service-desc" and/or "service-doc" link to an API Definition b. Validate that the landing page includes a "conformance" link to the conformance class declaration c. Validate that the landing page includes a "data" link to the Feature contents.

The landing page may be retrieved in a number of different formats. The following table identifies the applicable schema document for each format and the test to be used to validate the landing page against that schema. All supported formats should be exercised.

Table 8. Schema and Tests for Landing Pages

Format	**Schema Document**	**Test ID**
HTML	landingPage.json	/ats/html/content
JSON	landingPage.json	/ats/geojson/content

A.2.3. API Definition Path {root}/api (link)

Abstract Test 4	**/ats/core/api-definition-op**
Test Purpose	Validate that the API Definition document can be retrieved from the expected location.

Requirement	/req/core/api-definition-op
Test Purpose	Validate that the API Definition document can be retrieved from the expected location.
Test Method	1. Construct a path for each API Definition link on the landing page 2. Issue a HTTP GET request on each path 3. Validate that a document was returned with a status code 200 4. Validate the contents of the returned document using test /ats/core/api-definition-success.

Abstract Test 5	**/ats/core/api-definition-success**
Test Purpose	Validate that the API Definition complies with the required structure and contents.
Requirement	/req/core/api-definition-success
Test Method	Validate the API Definition document against an appropriate schema document.

A.2.4. Conformance Path {root}/conformance

Abstract Test 6	**/ats/core/conformance-op**
Test Purpose	Validate that a Conformance Declaration can be retrieved from the expected location.
Requirement	/req/core/conformance-op
Test Method	1. Construct a path for each "conformance" link on the landing page as well as for the {root}/conformance path. 2. Issue an HTTP GET request on each path 3. Validate that a document was returned with a status code 200 4. Validate the contents of the returned document using test /ats/core/conformance-success.

Abstract Test 7	**/ats/core/conformance-success**

Test Purpose	Validate that the Conformance Declaration response complies with the required structure and contents.
Requirement	/req/core/conformance-success
Test Method	1. Validate the response document against OpenAPI 3.0 schema confClasses.yaml 2. Validate that the document includes the conformance class "http://www.opengis.net/spec/ogcapi-features-1/1.0/conf/core" 3. Validate that the document list all OGC API conformance classes that the API implements.

A.3. Conformance Class Collections

Conformance Class	
http://www.opengis.net/spec/ogcapi-common/1.0/conf/collections	
Target type	Web API
Requirements Class	http://www.opengis.net/spec/ogcapi_common/1.0/req/collections
Dependency	Conformance Class "OAPI Core"

A.3.1. General Tests

CRS 84

Abstract Test 8	**/ats/collections/crs84**
Test Purpose	Validate that all spatial geometries provided through the API are in the CRS84 or CRS84h spatial reference system unless otherwise requested by the client.
Requirement	/req/collections/crs84
Test Method	1. Do not specify a coordinate reference system in any request. All spatial data should be in the default coordinate reference system. 2. If the retrieved spatial data includes elevations, validate that the data is in the CRS84h reference system. 3. If the retrieved spatial data does not include elevations, validate that the data is in the CRS84 reference system.

A.3.2. Feature Collections {root}/collections

Abstract Test 9	**/ats/collections/rc-md-op**
Test Purpose	Validate that information about the Collections can be retrieved from the expected location.
Requirement	/req/collections/rc-md-op
Test Method	1. Issue an HTTP GET request to the URL {root}/collections 2. Validate that a document was returned with a status code 200 3. Validate the contents of the returned document using test /ats/collections/rc-md-success.

Abstract Test 10	**/ats/collections_rc-md-success**
Test Purpose	Validate that the Collections content complies with the required structure and contents.
Requirement	/req/collections/rc-md-success, /req/collections/crs84
Test Method	1. Validate that all response documents comply with /ats/collections/rc-md-links 2. In case the response includes a "crs" property, validate that the first value is either: "http://www.opengis.net/def/crs/OGC/1.3/CRS84" or "http://www.opengis.net/def/crs/OGC/0/CRS84h" 3. Validate the collections content for all supported media types using the resources and tests identified in Table 9

The Collections content may be retrieved in a number of different formats. The following table identifies the applicable schema document for each format and the test to be used to validate against that schema. All supported formats should be exercised.

Table 9. Schema and Tests for Collections content

Format	**Schema Document**	**Test ID**
HTML	collections.json	/ats/html/content
JSON	collections.json	/ats/geojson/content

A.3.3. Feature Collection {root}/collections/{collectionId}

Abstract Test 11	**/ats/collections/src-md-op**

Test Purpose	Validate that the Collection content can be retrieved from the expected location.
Requirement	/req/collections/src-md-op
Test Method	For every Feature Collection described in the Collections content, issue an HTTP GET request to the URL `/collections/{collectionId}` where `{collectionId}` is the `id` property for the collection. 1. Validate that a Collection was returned with a status code 200 2. Validate the contents of the returned document using test /ats/collections/src-md-success.

Abstract Test 12	**/ats/collections/src-md-success**
Test Purpose	Validate that the Collection content complies with the required structure and contents.
Requirement	/req/collections/src-md-success
Test Method	Verify that the content of the response is consistent with the content for this Resource Collection in the `/collections` response. That is, the values for `id`, `title`, `description` and `extent` are identical.

A.3.4. Features {root}/collections/{collectionId}/items

NOTE This test is too Feature centric. Will need to be greatly reduced in scope.

Abstract Test 13	**/ats/collections/rc-op**
Test Purpose	Validate that resources can be identified and extracted from a Collection using query parameters.
Requirement	/req/collections/rc-op

Test Method	1. For every resource collection identified in Collections, issue an HTTP GET request to the URL `/collections/{collectionId}/items` where `{collectionId}` is the `id` property for a Collection described in the Collections content. 2. Validate that a document was returned with a status code 200. Repeat these tests using the following parameter tests: **Bounding Box:** • Parameter /ats/collections/rc-bbox-definition • Response /ats/collections/rc-bbox-response **DateTime:** • Parameter /ats/collections/rc-time-definition • Response /ats/collections/rc-time-response Execute requests with combinations of the "bbox" and "datetime" query parameters and verify that only features are returned that match both selection criteria.

Abstract Test 14	**/ats/collections/rc-bbox-definition**
Test Purpose	Validate that the bounding box query parameters are constructed correctly.
Requirement	/req/collections/rc-bbox-definition

Test Method	Verify that the `bbox` query parameter complies with the following definition (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`
	
	Use a bounding box with four numbers in all requests:
	
	• Lower left corner, WGS 84 longitude
	• Lower left corner, WGS 84 latitude
	• Upper right corner, WGS 84 longitude
	• Upper right corner, WGS 84 latitude

Abstract Test 15	**/ats/collections/rc-bbox-response**
Test Purpose	Validate that the bounding box query parameters are processed corrrectly.
Requirement	/req/collections/rc-bbox-response
Test Method	1. Verify that only resources that have a spatial geometry that intersects the bounding box are returned as part of the result set. 2. Verify that the `bbox` parameter matched all resources in the collection that were not associated with a spatial geometry (this is only applicable for datasets that include resources without a spatial geometry). 3. Verify that the coordinate reference system of the geometries is WGS 84 longitude/latitude ("http://www.opengis.net/def/crs/OGC/1.3/CRS84" or "http://www.opengis.net/def/crs/OGC/0/CRS84h") since no parameter `bbox-crs` was specified in the request.

Abstract Test 16	**/ats/collections/rc-time-definition**
Test Purpose	Validate that the dateTime query parameters are constructed correctly.
Requirement	/req/collections/rc-time-definition

Test Method	Verify that the `datetime` query parameter complies with the following definition (using an OpenAPI Specification 3.0 fragment):
	`name: datetime`
	`in: query`
	`required: false`
	`schema:`
	`  type: string`
	`style: form`
	`explode: false`

Abstract Test 17	**/ats/collections/rc-time-response**
Test Purpose	Validate that the dataTime query parameters are processed correctly.
Requirement	/req/collections/rc-time-response
Test Method	1. Verify that only resources that have a temporal geometry that intersects the temporal information in the `datetime` parameter were included in the result set 2. Verify that all resources in the collection that are not associated with a temporal geometry are included in the result set 3. Validate that the datetime parameter complies with the syntax described in /req/collections/rc-time-response.

Abstract Test 18	**/ats/collections/rc-response**
Test Purpose	Validate that the Resource Collection complies with the require structure and contents.
Requirement	/req/collections/rc-response

Test Method	The test method is specific to the resource type returned.

A.3.5. Second Tier Tests

These tests are invoked by other tests.

Extent

Abstract Test 19	**/ats/coollections/rc-md-extent**
Test Purpose	Validate that the `extent` property if it is present
Requirement	/req/collections/rc-md-extent
Test Method	1. Verify that the `extent` provides bounding boxes that include all spatial geometries 2. Verify that if the `extent` provides time intervals that include all temporal geometries in this collection. 3. A temporal extent of `null` indicates an open time interval.

Items

Abstract Test 20	**/ats/collections/rc-md-items**
Test Purpose	Validate that each collection provided by the server is described in the Collections Metadata.
Requirement	/req/collections/rc-md-items
Test Method	1. Verify that there is an entry in the `collections` array of the Collections Metadata for each feature collection provided by the API. 2. Verify that each collection entry includes an identifier. 3. Verify that each collection entry includes links in accordance with /collections/rc-md-items-links. 4. Verify that if the collection entry includes an extent property, that that property complies with /collections/rc-md-extent 5. Validate each collection entry for all supported media types using the resources and tests identified in Table 10

The collection entries may be encoded in a number of different formats. The following table identifies the applicable schema document for each format and the test to be used to validate

against that schema. All supported formats should be exercised.

Table 10. Schema and Tests for Collection Entries

Format	Schema Document	Test ID
HTML	collectionInfo.json	/ats/html/content
JSON	collectionInfo.json	/ats/geojson/content

Abstract Test 21	**/ats/collections/rc-md-items-links**
Test Purpose	Validate that each Feature Collection metadata entry in the Collections Metadata document includes all required links.
Requirement	/req/collections/rc-md-items-links
Test Method	1. Verify that each Collection item in the Collections Metadata document includes a `link` property for each supported encoding. 2. Verify that the `links` properties of the collection includes an item for each supported encoding with a link to the features resource (relation: `items`). 3. Verify that all links include the `rel` and `type` link parameters.

Links

Abstract Test 22	**/ats/collections/rc-md-links**
Test Purpose	Validate that the required links are included in the Collections Metadata document.
Requirement	/req/collections/rc-md-links
Test Method	Verify that the response document includes: 1. a link to this response document (relation: `self`), 2. a link to the response document in every other media type supported by the server (relation: `alternate`). Verify that all links include the `rel` and `type` link parameters.

A.4. Conformance Class GeoJSON

Conformance Class

http://www.opengis.net/spec/ogcapi-common/1.0/conf/geojson	
Target type	Web API
Requirements Class	http://www.opengis.net/spec/ogcapi_common/1.0/req/geojson
Dependency	Conformance Class "OAPI Core"

A.4.1. GeoJSON Definition

Abstract Test 23	**/ats/geojson/definition**
Test Purpose	Verify support for JSON and GeoJSON
Requirement	/req/geojson/definition
Test Method	1. A resource is requested with response media type of `application/geo+json` 2. All `200`-responses SHALL support the following media types: ◦ `application/geo+json` for resources that include feature content, and ◦ `application/json` for all other resources.

A.4.2. GeoJSON Content

Abstract Test 24	**/ats/geojson/content**
Test Purpose	Verify the content of a GeoJSON document given an input document and schema.
Requirement	/req/geojson/content
Test Method	1. Validate that the document is a GeoJSON document. 2. Validate the document against the schema using a JSON Schema validator.

A.5. Conformance Class HTML

Conformance Class	
http://www.opengis.net/spec/ogcapi-common/1.0/conf/html	
Target type	Web API

Requirements Class	http://www.opengis.net/spec/ogcapi_common/1.0/req/html
Dependency	Conformance Class "OAPI Core"

A.5.1. HTML Definition

Abstract Test 25	**/ats/html/definition**
Test Purpose	Verify support for HTML
Requirement	/req/html/definition
Test Method	Verify that every `200`-response of every operation of the API where HTML was requested is of media type `text/html`

A.5.2. HTML Content

Abstract Test 26	**/ats/html/content**
Test Purpose	Verify the content of an HTML document given an input document and schema.
Requirement	/req/html/content
Test Method	1. Validate that the document is an HTML 5 document 2. Manually inspect the document against the schema.

A.6. Conformance Class OpenAPI 3.0

Conformance Class	
http://www.opengis.net/spec/ogcapi-common/1.0/conf/oas3	
Target type	Web API
Requirements Class	http://www.opengis.net/spec/ogcapi_common/1.0/req/oas3
Dependency	Conformance Class "OAPI Core"

Abstract Test 27	**/ats/oas30/completeness**
Test Purpose	Verify the completeness of an OpenAPI document.

Requirement	/req/oas30/completeness
Test Method	Verify that for each operation, the OpenAPI document describes all HTTP Status Codes and Response Objects that the API uses in responses.

Abstract Test 28	**/ats/oas30/exceptions-codes**
Test Purpose	Verify that the OpenAPI document fully describes potential exception codes.
Requirement	/req/oas30/exceptions-codes
Test Method	Verify that for each operation, the OpenAPI document describes all HTTP Status Codes that may be generated.

Abstract Test 29	**/ats/oas30/oas-definition-1**
Test Purpose	Verify that JSON and HTML versions of the OpenAPI document are available.
Requirement	/req/oas30/oas-definition-1
Test Method	1. Verify that an OpenAPI definition in JSON is available using the media type `application/vnd.oai.openapi+json;version=3.0` and link relation `service-desc` 2. Verify that an HTML version of the API definition is available using the media type `text/html` and link relation `service-doc`.

Abstract Test 30	**/ats/oas30/oas-definition-2**
Test Purpose	Verify that the OpenAPI document is valid JSON.
Requirement	/req/oas30/oas-definition-2
Test Method	Verify that the JSON representation conforms to the OpenAPI Specification, version 3.0.

Abstract Test 31	**/ats/oas30/oas-impl**

Test Purpose	Verify that all capabilities specified in the OpenAPI definition are implemented by the API.
Requirement	/req/oas30/oas-impl
Test Method	1. Construct a path from each URL template including all server URL options and all enumerated path parameters. 2. For each path defined in the OpenAPI document, validate that the path performs in accordance with the API definition and the API-Features standard.

Abstract Test 32	**/ats/oas30/security**
Test Purpose	Verify that any authentication protocols implemented by the API are documented in the OpenAPI document.
Requirement	/req/oas30/security
Test Method	1. Identify all authentication protocols supported by the API. 2. Validate that each authentication protocol is described inthe OpenAPI document by a Security Schema Object and its' use specified by a Security Requirement Object.

Annex B: Examples (Informative)

B.1. Example Landing Pages

Example 5. 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. API Description Examples

NOTE include::examples/tbd.adoc[]

B.3. Conformance Examples

Example 6. Conformance Response

This example response in JSON is for an implementation of the OGC API-Features Standard 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.4. Collections Metadata Examples

Example 7. Collection metadata response document

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

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

Representations of this 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".

A bulk download of all the features in the dataset is referenced using link relation type "enclosure"

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" }
      ]
    }
  ]
}
```


B.5. Collection Information Examples

NOTE include::examples/tbd.adoc[]

Annex C: Revision History

Date	Release	Editor	Primary clauses modified	Description
2019-10-31	October 2019 snapshot	C. Heazel	all	Baseline update

Annex D: Bibliography

- Open Geospatial Consortium: The Specification Model — A Standard for Modular specifications, OGC 08-131
- 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 (DCAT) - Version 2, W3C Recommendation 04 February 2020, https://www.w3.org/TR/vocab-dcat-2/
- IANA: Link Relation Types, https://www.iana.org/assignments/link-relations/link-relations.xml