Draft OGC API - Common - Part 1

Core

Open Geospatial Consortium

Submission Date: <yyyy-mm-dd>

Approval Date: <yyyy-mm-dd>

Publication Date: 2020-07-31

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: 0.0.7

Category: OGC® Implementation Standard

Editor: Charles Heazel

Draft OGC API - Common - Part 1: Core

Warning

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

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

Document type:	OGC® 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 more than one of those standards. These common practices are documented in the OGC API - Common suite of standards. API-Common standards serve as reusable building-blocks. Developers of OGC standards will use these building-blocks in the construction of OGC Web API Standards. The result is a modular suite of coherent API standards which can be adapted by an system designer for the unique requirements of their system.

The OGC API - Common - Part 1: Core Standard defines the resources and access mechanisms which are useful for a client seeking to understand the offerings and capabilities of an API. These resources and their access mechanisms are supported by all conformant OGC Web APIs and are described in Table 1.

Table 1. Common Core Resources

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

ii. Keywords

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

ogcdoc, OGC document, geographic information, spatial data, API, json, html, OpenAPI, 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, defines conformance 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 Discovery operations directed against the API itself. This includes general information about the API, the API definition, and the conformance classe implemented. Additional OGC Web API Standards will extend this core with resource-specific capabilities.

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 - Part 1: Core 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 four conformance classes. The conformance classes implemented by an OGC API are advertised through the `/conformance` resource 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 - Core* are:

3.1. Core Requirements Class

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.

The Core requirements class is specified in **Requirements Class Core** (Chapter 8).

3.2. Encoding Requirements Classes

The *Core* requirements class does not mandate a specific encoding or format for representations of resources. However, both *HTML* and *JSON* are commonly used encodings for spatial data on the web. The *HTML* and *JSON* requirements classes specify the encoding of resource representations using:

- HTML
- JSON

Neither of these encodings is mandatory. An implementor of the *API-Common* standard may decide to implement another encodings instead of, or in addition to, these two.

The Encoding Requirements Classes are specified in **Encoding Requirements Classes** (Chapter 9).

3.3. OpenAPI 3.0 Requirements Class

The *API-Common - core* 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.

The OpenAPI 3.0 Requirements Class is specified in **OpenAPI 3.0 Requirements Class** (Chapter 10).

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.

- json-schema-org, **JSON Schema,** September 2019, https://json-schema.org/specification.html
- Open API Initiative: **OpenAPI Specification 3.0.3**, https://github.com/OAI/OpenAPI-Specification/blob/master/versions/3.0.3.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 3986, Uniform Resource Identifier (URI): Generic Syntax**, http://tools.ietf.org/rfc/rfc3896.txt
- Greforio, J., Fielding, R., Hadley, M., Nottingham, M., Orchard, D.: **IETF RFC 6570, URI Template**, https://tools.ietf.org/html/rfc6570
- Fielding, R., Reschke, JSchaub: **IETF RFC 7231, Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content**, https://tools.ietf.org/rfc/rfc7231.txt
- Bray, T.: **IETF RFC 8259, The JavaScript Object Notation (JSON) Data Interchange Format**, http://tools.ietf.org/rfc/rfc8259.txt
- Nottingham, M.: **IETF RFC 8288, Web Linking**, http://tools.ietf.org/rfc/rfc8288.txt

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.

- **Control Data**
  1. defines the purpose of a message between components. (Fielding 2000)
  2. used to parameterize requests and override the default behavior of some connecting elements. (Fielding 2000)

- **Landing Page**
  This is the root resource of an API. It serves as the root node of the API Resource tree and provides the information needed to navigate all of the resources exposed through the API.

- **Representation**
  the current or intended state of a resource encoded for exchange between components. (based on Fielding 2000)

- **Resource**
  1. Any information that can be named. (Fielding 2000)
  2. The intended conceptual target of a hypertext reference. (Fielding 2000)

- **Resource Type**
  the definition of a type of resource. Resource types are re-usable components which are independent of where the resource resides in the API.

- **Uniform Resource Identifier (URI)**
  an identifier consisting of a sequence of characters matching the syntax rule named <URI>. (IETF RFC 3986)

- **Uniform Resource Locator (URL)**
  the subset of URIs that, in addition to identifying a resource, provide a means of locating the resource by describing its primary access mechanism (e.g., its network "location"). (IETF RFC 3986)

- **Web API**
  API using an architectural style that is founded on the technologies of the Web. (W3C Data on the Web Best Practices)

Chapter 6. Conventions

6.1. Web API Fundamentals

The following concepts are critical to understanding OGC Web API standards.

1. The purpose of a Web API is to provide a uniform interface to resources.
2. Resources are uniquely identified using Uniform Resource Identifiers (URI).
3. A user manipulates a resource through representations of that resource.
4. A representation is the current or intended state of a resource encoded for exchange between components.
5. The format used to encode a representation is negotiated between the components participating in the exchange.
6. Representations are exchanged between components using the HTTP protocol and the operations (GET, PUT, etc.) that HTTP supports.

6.2. Identifiers

The Architecture of the World Wide Web establishes the URI as the single global identification system for the Web. Therefore, URIs or URI Templates are used in OGC Web API standards to identify key entities in those standards.

In accordance with OGC policy, only the Uniform Resource Locator (URL) form of URIs is used.

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

All Requirements, Conformance Modules, and Conformance Classes that appear in this document are denoted by partial URIs that are relative to this base.

Resources described in this document are denoted by partial URIs that are relative to the `root` node of the API. This node serves as the head of the resource tree exposed through an API. In OpenAPI, the root node is identified by the `url` field of the Server Object. When used in a URI, the root node is designated as `{root}`.

The partial URIs used to identify Resources in this document are refered to as the resource `path`. The purpose of a resource `path` is to identify the referenced resource within the context of this standard.

This standard defines Resources which may appear in more than one place in the API. These Resource Types are identified by name rather than by URI.

Summary for Developers:

RFC 3986 defines a URI in Backus-Naur Form (BNF) as follows:

Backus-Naur Definition of URI

```
    URI = scheme ":" hier-part [ "?" query ] [ "#" fragment ]

        hier-part     = "//" authority path-abempty
                        / path-absolute
                        / path-rootless
                        / path-empty

        authority     = [ userinfo "@" ] host [ ":" port ]

        path-abempty  = *( "/" segment )

        path-absolute = "/" [ segment-nz *( "/" segment ) ]

        path-rootless = segment-nz *( "/" segment )

        path-empty    = 0<pchar>
```

The following rules should be used when interpreting the BNF:

- `scheme` is assumed to be `HTTP` or `HTTPS`
- `authority` is provided by the API developer
- `{root}` designates the `scheme`, `authority`, and `path` to the root node of the API implementation.
- only the `path-absolute` and `path-rootless` patterns are used
- parameters passed as part of an operation are encoded in the `query`.
- parameters passed in HTTP headers or as cookies are out of scope for this Standard.

The following example shows a URI categorised according to RFC 3986 and OGC Web API standards.

Example URI and Components

```
URI     foo://example.com:8042/myapi/mydata/?name=roads#centerline
        \__/ \______________/\____________/ \_________/ \________/
          |           |            |            |           |
3986    scheme     authority     path         query      fragment
        \_____________________________/\_____/\__________/
                      |                 |        |
OGC                {root}              path   parameters
```

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.

6.3. Link relations

RFC 8288 (Web Linking) is used to express relationships between resources. Link relation types

from the IANA Link Relations Registry are used wherever possible. Additional link relation types are registered with the OGC Link Relation Registry.

The link relationships used in API-Common Core are described in Table 2.

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.

6.4. Use of HTTPS

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

OGC Web API standards do not prohibit the use of any valid HTTP option. However, implementors should be aware that optional capabilities which are not in common use could be an impediment to interoperability.

6.5. API definition

6.5.1. 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.

6.5.2. 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.

6.5.3. 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.

6.5.4. Reusable OpenAPI components

Reusable components for OpenAPI definitions for an OGC API are referenced from this document. They are available from the OGC Schemas Registry at http://schemas.opengis.net/ogcapi/common/part1/1.0.

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. New OGC Web API standards are under development to provide API alternatives to the OWS standards.

The OGC API - Common suite of standards specify common modules for defining OGC Web API standards that follow 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:

- A common `core` which is the same for all OGC Web API standards. This OGC API - Common - Core Standard provides the information needed by a client to understand and use an OGC Web API.
- Clear separation between common requirements and more resource specific capabilities. The OGC API - Common suite of standards specify the *common* requirements that may be relevant to almost anyone who wants to build an API for spatial resources. Resource-specific requirements are addressed in resource-specific OGC standards.
- 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 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. 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.

7.3. 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. The link relationships used in API-Common Core are described in Table 2.

Table 2. Link Relations

Link Relation	Purpose
`alternate`	Refers to a substitute for this context [IANA]. Refers to a representation of the current resource which is encoded using another media type (the media type is specified in the `type` link attribute).
`collection`	The target IRI points to a resource which represents the collection resource for the context IRI. [IANA]
`current`	Refers to a resource containing the most recent item(s) in a collection of resources. [IANA]
`conformance`	Refers to a resource that identifies the specifications that the link's context conforms to. [OGC]
`data`	Refers to the root resource of a dataset in an API. [OGC]
`describedby`	Refers to a resource providing information about the link's context.[IANA] Links to external resources which further describe the subject resource
`first`	An IRI that refers to the furthest preceding resource in a series of resources.[IANA]
`item`	The target IRI points to a resource that is a member of the collection represented by the context IRI.[IANA]
`items`	Refers to a resource that is comprised of members of the collection represented by the link's context. [OGC]
`last`	An IRI that refers to the furthest following resource in a series of resources.
`license`	Refers to a license associated with this context. [IANA]
`next`	Indicates that the link's context is a part of a series, and that the next in the series is the link target. [IANA]
`prev`	Indicates that the link's context is a part of a series, and that the previous in the series is the link target. [IANA]

Link Relation	Purpose
`self`	Conveys an identifier for the link's context. [IANA] A link to another representation of this resource.
`service-desc`	Identifies service description for the context that is primarily intended for consumption by machines. [IANA] API definitions are considered service descriptions.
`service-doc`	Identifies service documentation for the context that is primarily intended for human consumption. [IANA]
`start`	Refers to the first resource in a collection of resources.[IANA]

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",
        "rel"
        ],
    "properties": {
        "href": {
            "type": "string",
            "example": "http://data.example.com/buildings/123"
            },
        "rel": {
            "type": "string",
            "example": "alternate"
            },
        "type": {
            "type": "string",
            "example": "application/geo+json"
            },
        "hreflang": {
            "type": "string",
            "example": "en"
            },
        "title": {
            "type": "string",
            "example": "Trierer Strasse 70, 53115 Bonn"
            },
        "length": {
            "type": "integer"
            }
        }
    }
```

Chapter 8. Requirement Class "Core"

Requirements Class	
http://www.opengis.net/spec/ogcapi-common-1/1.0/req/core	
Target type	Web API
Dependency	RFC 2616 (HTTP/1.1)
Dependency	RFC 2818 (HTTP over TLS)
Dependency	RFC 8288 (Web Linking)

The Core Conformance Class of API-Common describes how the core resources are accessed through an OGC conformant API. The resources described in this Requirements Class are organized around the following resources:

Table 3. Common Core Resources

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

In addition, requirements which are applicable to all OGC API standards and are not specific to a resource type are included in the General Requirements section.

8.1. API landing page

Each 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.

8.1.1. Operation

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

8.1.2. 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: • `/api` (relation type 'service-desc' or 'service-doc') • `/conformance` (relation type 'conformance')

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"}
        }
    },
    "additionalProperties": true
}
```

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

8.1.3. Error Situations

See HTTP Status Codes for general guidance.

8.2. 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.

8.2.1. 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.

8.2.2. 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 the `Accept` header populated 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, THEN 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.

8.2.3. Error Situations

See HTTP Status Codes for general guidance.

8.3. Declaration of Conformance Classes

The OGC Web API Standards define a collection of modules which can be assembled into a Web API. The first question a client will ask when accessing one of these APIs is "what are you?" In other words, what modules were used to create you? Since implementors have a choice on which

modules to use, there is no simple answer. The best that can be done is to provide a list of the modules implemented, the Conformance Classes.

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

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

8.3.1. Operation

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

8.3.2. 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-1/1.0/req/core"
            }
        }
    }
}
```

8.3.3. Error situations

See HTTP Status Codes for general guidance.

8.4. General Requirements

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

8.4.1. 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.

8.4.2. HTTP Status Codes

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

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

Table 4. Typical HTTP status codes

Status code	Description
`200`	A successful request.
`302`	The target resource was found but resides temporarily under a different URI. A 302 response is not evidence that the operation has been successfully completed.
`303`	The server is redirecting the user agent to a different resource. A 303 response is not evidence that the operation has been successfully completed.
`304`	An entity tag was provided in the request and the resource has not changed since the previous request.
`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 4.

8.4.3. Unknown or invalid query parameters

If a server wants to support query parameters, these have to be explicitly declared in the API definition.

Requirement 8	/req/core/query-param-known
A	IF a query parameter is specified in the API definition, THEN the server SHALL meet the requirements specified for that parameter.

Query parameters that have not been explicitly specified in the API definition for that operation will be treated as unknown.

Requirement 9	/req/core/query-param-unknown
A	IF the request URI includes a query parameter that is not specified in the API definition THEN The server SHALL respond with a response with the status code `400`,

If OpenAPI is used to represent the API definition, a capability exists to define query parameters without explicitly declaring them. That is,

OpenAPI schema for query parameters

```
in: query
name: queryParameters
schema:
  type: object
  additionalProperties: true
style: form
```

Note that the name of the parameter does not matter as the actual query parameters are the names of the object properties. For example, assume that the value of `queryParameters` is this object:

```
{
  "first_parameter": "some value",
  "other_parameter": 42
}
```

In the request URI this would be expressed as `&first_parameter=some%20value&other_parameter=42`.

Requirement 10	/req/core/query-param-invalid

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

This is a general rule that applies to all parameters, whether they are specified in this document or in additional parts. A value is invalid if it violates the API definition or any other constraint for that parameter stated in a requirement.

8.4.4. Web Caching

Entity tags are a mechanism for web cache validation and for supporting conditional requests to reduce network traffic. Entity tags are specified by HTTP/1.1 (RFC 2616).

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

8.4.5. Support for Cross-Origin Requests

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

Recommendation 3	**/rec/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)

8.4.6. Encodings

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

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

While the OAPI Common Core standard does not specify any mandatory encoding, the following encodings are recommended:

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.

JSON encoding recommndation:

Recommendation 5	**/rec/core/json**
A	To support processing of an API with a web applet, implementations SHOULD consider supporting a JSON encoding.

The section Media Types includes a discussion on encoding media types for this standard.

8.4.7. 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 11	**/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.

8.4.8. 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. Encoding Requirements Classes

9.1. Overview

This clause specifies two 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
- JSON

Neither of these encodings are mandatory and an implementation of the Core requirements class may implement some, all, or none of them.

9.2. 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-1/1.0/req/html	
Target type	Web API
Dependency	Requirements Class "OAPI Core"
Dependency	HTML5
Dependency	Schema.org

Requirement 12	**/req/html/definition**
A	Every `200`-response of an operation of the API SHALL support the media type `text/html`.

Requirement 13	**/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 7	**/rec/html/schema-org**
A	A `200`-response with the media type `text/html`, SHOULD include Schema.org annotations.

9.3. Requirement Class "JSON"

JSON is a text syntax that facilitates structured data interchange between programming languages. It commonly used for Web-based software-to-software interchanges. Most Web developers are comfortable with using a JSON-based format, so supporting JSON is recommended for machine-to-machine interactions.

Requirements Class	
http://www.opengis.net/spec/ogcapi-common-1/1.0/req/json	
Target type	Web API
Dependency	Requirements Class "OAPI Core"
Dependency	IETF RFC 8259: The JavaScript Object Notation (JSON) Data Interchange Format
Dependency	JSON Schema

Requirement 14	**/req/json/definition**
A	`200`-responses of the server SHALL support the `application/json` media type.

Requirement 15	**/req/json/content**
A	Every `200`-response with the media type `application/json` SHALL include, or link to, a payload encoded according to the JSON Interchange Format
B	The schema of all responses with the media type `application/json` SHALL conform with the JSON Schema specified for that resource.

An example JSON Schema for the landing page is available at landingPage.yaml.

Chapter 10. OpenAPI 3.0 Requirements Class

10.1. Basic requirements

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

Requirements Class	
http://www.opengis.net/spec/ogcapi-common-1/1.0/req/oas30	
Target type	Web API
Dependency	Requirements Class "OAPI Core"
Dependency	OpenAPI Specification 3.0.2

Requirement 16	**/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.

Requirement 17	**/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 18	**/req/oas30/oas-impl**
A	The API SHALL implement all capabilities specified in the OpenAPI definition.

10.2. Complete definition

Requirement 19	**/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.

10.3. Exceptions

Requirement 20	**/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 1. An exception response object definition

```
description: An error occurred.
content:
  application/json:
    schema:
      $ref:
http://schemas.opengis.net/ogcapi/common/part1/1.0/openapi/schemas/exception.yaml
  text/html:
    schema:
      type: string
```

10.4. 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.

10.4.1. 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.

10.4.2. 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.

10.4.3. 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.

10.4.4. 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.

10.4.5. 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
```

10.5. 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 21	**/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.

Chapter 11. Media Types

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

The media type that would typically be used in an OGC API for machine-to-machine exchanges would be `application/json`.

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.

Chapter 12. Security Considerations

The OAPI-Common Core Standard does not specify any specific security controls. However, it was constructed so that security controls can be added without impacting conformance.

See Clause 10, Security Section for a discussion of OpenAPI support for security controls.

add additional text as needed

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 required 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 5 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

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 5. 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 JSON

Conformance Class	
http://www.opengis.net/spec/ogcapi-common/1.0/conf/json	
Target type	Web API
Requirements Class	http://www.opengis.net/spec/ogcapi-common-1/1.0/req/json
Dependency	Conformance Class "OAPI Core"

A.3.1. JSON Definition

Abstract Test 8	**/ats/json/definition**
Test Purpose	Verify support for JSON
Requirement	/req/json/definition
Test Method	1. A resource is requested with response media type of `application/json` 2. All `200` responses SHALL be encoded in the JSON (IETF RFC 8259) media type.

A.3.2. JSON Content

Abstract Test 9	**/ats/json/content**
Test Purpose	Verify the content of a JSON document given an input document and schema.
Requirement	/req/json/content

Test Method	1. Validate that the document is a JSON document. 2. Validate the document against the schema using a JSON Schema validator.

A.4. 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.4.1. HTML Definition

Abstract Test 10	**/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.4.2. HTML Content

Abstract Test 11	**/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.5. 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 12	**/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 13	**/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 14	**/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 15	**/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 16	**/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 17	**/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 2. 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" }
  ]
}
```

B.2. API Description Examples

NOTE include::examples/tbd.adoc[]

B.3. Conformance Examples

Example 3. Conformance Response

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

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

Annex C: Revision History

Date	Release	Editor	Primary clauses modified	Description
2019-10-31	October 2019 snapshot	C. Heazel	all	Baseline update
2020-04-21	Public Comments	C. Heazel	all	Separation of Collections from Core plus additional comment adjudications.
2020-07-31	Cleanup	C. Heazel	all	General clean-up and update.

Annex D: Glossary

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

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

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

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

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

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

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

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

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

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

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

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

Annex E: Bibliography

- Fielding, Roy Thomas: **Architectural Styles and the Design of Network-based Software Architectures**. Doctoral dissertation, University of California, Irvine, 2000, https://www.ics.uci.edu/fielding/pubs/dissertation/fielding_dissertation.pdf
- IANA: **Link Relation Types**, https://www.iana.org/assignments/link-relations/link-relations.xml
- Open Geospatial Consortium: **The Specification Model — A Standard for Modular specifications**, OGC 08-131
- **Schema.org**: http://schema.org/docs/schemas.html
- W3C: **Architecture of the World Wide Web, Volume One**, W3C Recommendation, 15 December 2004, https://www.w3.org/TR/webarch/
- W3C: **Data Catalog Vocabulary (DCAT) - Version 2**, W3C Recommendation, 04 February 2020, https://www.w3.org/TR/vocab-dcat-2/
- W3C: **Data on the Web Best Practices**, W3C Recommendation, 31 January 2017, https://www.w3.org/TR/dwbp/
- W3C/OGC: **Spatial Data on the Web Best Practices**, W3C Working Group Note, 28 September 2017, https://www.w3.org/TR/sdw-bp/
- W3C: **HTML5, W3C Recommendation**, http://www.w3.org/TR/html5/