OGC® DOCUMENT: 19-072

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

Open Geospatial Consortium

OGC API - COMMON - PART 1: CORE

STANDARD
Implementation

DRAFT

Version: 1.0.0
Submission Date: 2021-08-23
Approval Date: 2022-02-02
Publication Date: 2022-11-09
Editor: Charles Heazel

CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF RECOMMENDATIONS

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. These common practices are documented in the OGC API — Common standard. The OGC API - Common standard is a multi-part standard that specifies reusable building-blocks that can be used in the construction of OGC API Standards. This document presents Part 1, the Core, of the OGC API – Common standard. Standards developers will use these building-blocks in the construction of other OGC Standards that relate to Web APIs. The result is a modular suite of coherent API standards which can be adapted by a system designer for the unique requirements of their system.

The purpose of the OGC API — Common — Part 1: Core Standard (API-Core) is to define those fundamental building blocks and requirements which are applicable to all OGC Web API Standards.

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

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

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

IV SECURITY CONSIDERATIONS

The OGC API — Common — Part 1: 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 11.4 for a discussion of OpenAPI support for security controls.

V SUBMITTING ORGANIZATIONS

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

- Ecere Corporation
- Heazeltech LLC
- Hexagon
- Interactive Instruments GmbH
- U.K. Met Office
- Universitat Autònoma de Barcelona (CREAF)
- U.S. Army Geospatial Center
- U.S. Geological Survey
- U.S. National Aeronautics and Space Administration (NASA)
- U.S. National Geospatial-Intelligence Agency (NGA)

VI SUBMITTERS

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

Table — Submitters

NAME	AFFILIATION
Charles Heazel *(editor)*	Heazeltech
David Blodgett	U.S. Geological Survey
Clemens Portele	interactive instruments GmbH
Sylvester Hagler	U.S. National Geospatial-Intelligence Agency
Jeffrey Harrison	U.S. Army Geospatial Center
Frédéric Houbie	Hexagon

NAME	AFFILIATION
Jérôme Jacovella-St-Louis	Ecere Corporation
Chris Little	U.K. Met Office
Joan Masó	UAB-CREAF
Donald Sullivan	NASA
Panagiotis (Peter) A. Vretanos	CubeWerx Inc., part of the MariaDB Corporation

1

SCOPE

1 SCOPE

The OGC API — Common — Part 1: Core Standard provides the fundamental rules for implementing a Web API that conforms to OGC API Standards. It seeks to establish a solid foundation which can be extended by other resource-specific Web API Standards.

First, this OGC Standard establishes rules for the use of HTTP protocols and Uniform Resource Identifiers (URIs). These requirements seek to provide both API servers and clients a predictable environment for the exchange of HTTP requests and responses. These rules are applicable regardless of the resources being accessed.

This OGC Standard then enables discovery operations directed against a Web API implementation. It identifies the hosted resources, defines conformance classes, and provides both human and machine-readable documentation of the API design. The requirements specified in this standard SHOULD be applicable to any Web API implementation.

2

CONFORMANCE

2 CONFORMANCE

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

OGC API — Common — Part 1: Core provides a common foundation for OGC Web API standards. The assumption is that this standard will only be implemented through inclusion in other standards. Therefore, all the relevant abstract tests in Annex A should be included in or referenced by the Abstract Test Suite (ATS) in each standard that implements conformance classes defined in this standard.

This standard identifies five conformance classes. The conformance classes implemented by an OGC API are advertised through the `/conformance` resource on the implementation instance of the API's 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.

2.1. Core Requirements Class

The requirements specified in the *Core Requirements Class* are applicable to all OGC API standards. They assure consistent use of the HTTP protocols, provide rules for the construction of URIs, and define requirements governing the use and processing of URI query parameters. Through these requirements, OGC API — Common seeks to assure that implementations of OGC API standards will provide a predictable interface for their peers.

The Core requirements class is specified in Clause 8 **Core Requirements Class**.

2.2. Landing Page Requirements Class

The requirements specified in the *Landing Page Requirements Class* provide a minimal useful service interface for an OGC Web API. These resources convey a basic understanding of the API and provide a starting point for further discovery. The requirements specified in this requirements class are recommended for all OGC Web APIs.

The Landing Page requirements class is specified in Clause 9 **Landing Page Requirements Class**.

2.3. Encoding Requirements Classes

The OGC API — Common Standard 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 implementer of the *API-Common* standard may decide to implement other encodings instead of, or in addition to, these two.

The Encoding Requirements Classes are specified in Clause 10 **Encoding Requirements Classes**.

2.4. OpenAPI 3.0 Requirements Class

The OGC API — Common Standard does not mandate any encoding or format for the formal definition of the API. The preferred 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 Clause 11 **OpenAPI 3.0 Requirements Class**.

2.5. OGC Building Blocks Registry

To facilitate the discovery and management of the building blocks specified in this and other OGC API Standards, a registry has been established at https://blocks.ogc.org .

3

NORMATIVE REFERENCES

3 NORMATIVE REFERENCES

The following documents are referred to in the text in such a way that some or all of their content constitutes requirements of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

E. Rescorla: IETF RFC 2818, *HTTP Over TLS*. RFC Publisher (2000). https://www.rfc-editor.org/info/rfc2818.

T. Berners-Lee, R. Fielding, L. Masinter: IETF RFC 3986, *Uniform Resource Identifier (URI): Generic Syntax*. RFC Publisher (2005). https://www.rfc-editor.org/info/rfc3986.

J. Gregorio, R. Fielding, M. Hadley, M. Nottingham, D. Orchard: IETF RFC 6570, *URI Template*. RFC Publisher (2012). https://www.rfc-editor.org/info/rfc6570.

R. Fielding, J. Reschke (eds.): IETF RFC 7231, *Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content*. RFC Publisher (2014). https://www.rfc-editor.org/info/rfc7231.

R. Fielding, J. Reschke (eds.): IETF RFC 7232, *Hypertext Transfer Protocol (HTTP/1.1): Conditional Requests*. RFC Publisher (2014). https://www.rfc-editor.org/info/rfc7232.

M. Nottingham, E. Wilde: IETF RFC 7807, *Problem Details for HTTP APIs*. RFC Publisher (2016). https://www.rfc-editor.org/info/rfc7807.

M. Nottingham: IETF RFC 8288, *Web Linking*. RFC Publisher (2017). https://www.rfc-editor.org/info/rfc8288.

T. Bray (ed.): IETF RFC 8259, *The JavaScript Object Notation (JSON) Data Interchange Format*. RFC Publisher (2017). https://www.rfc-editor.org/info/rfc8259.

Open API Initiative: OpenAPI Specification, Version 3.0. The latest patch version at the time of publication of this standard was 3.0.3, available from http://spec.openapis.org/oas/v3.0.3.

4

TERMS AND DEFINITIONS

4 TERMS AND DEFINITIONS

This document uses the terms defined in OGC Policy Directive 49, which is based on the ISO/IEC Directives, Part 2, Rules for the structure and drafting of International Standards. In particular, the word “shall” (not “must”) is the verb form used to indicate a requirement to be strictly followed to conform to this document and OGC documents do not use the equivalent phrases in the ISO/IEC Directives, Part 2.

This document also uses terms defined in the OGC Standard for Modular specifications (OGC 08-131r3), also known as the ‘ModSpec’. The definitions of terms such as standard, specification, requirement, and conformance test are provided in the ModSpec.

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

4.1. Kebab case

A case style where punctuation is removed and spaces are replaced by single hyphens. All letters are in lower case when used in OGC documentation. (Wikipedia) Example: `kebab-case-style`

4.2. Landing Page

Any page whose primary purpose is to contain a description of something else. (W3C, URLs in Data Primer)

Note 1 to entry: Note 1 — Landing pages often provide summaries or additional information about the thing that they describe. Examples are landing pages for images on Flickr or videos on YouTube, which are HTML pages that embed the media that they describe and provide access to comments and other metadata about it. Landing pages for documents are often tables of contents or abstracts.

Note 2 to entry: Note 2 — A landing page for a Web API serves as the root node of the API Resource tree and provides the information needed to navigate all the resources exposed through the API.

4.3. OGC Web API

A Web API that implements one or more Conformance Classes from an OGC API Standard.

4.4. Representation

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

4.5. Resource

entity that might be identified (Dublin Core Metadata Initiative — DCMI Metadata Terms)

Note 1 to entry: The term “resource”, when used in the context of an OGC Web API standard, should be understood to mean a Web Resource unless otherwise indicated.

4.6. Resource Type

a type of resource.

Note 1 to entry: Resource types are re-usable components that are independent of where the resource resides in the API.

4.7. Uniform Resource Identifier (URI)

an identifier consisting of a sequence of characters matching the syntax rule named “<URI>”. (IETF RFC 3986)

4.8. 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)

4.9. Web API

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

4.10. Web Resource

a resource that is identified by a URI.

5

ABBREVIATED TERMS

5 ABBREVIATED TERMS

API	Application Programming Interface
CORS	Cross-Origin Resource Sharing
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IANA	Internet Assigned Numbers Authority
OGC	Open Geospatial Consortium
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
YAML	YAML Ain't Markup Language

6

CONVENTIONS

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 standard are denoted by the URI `http://www.opengis.net/spec/ogcapi-common-1/1.0`. All Requirements, Requirements Modules and Conformance Modules 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. In this document the tag `{root}` designates the root node of a URI.

The partial URIs used to identify Resources in this document are referred to as the resource `path`. The purpose of a resource `path` is to identify the referenced resource within the context of this standard. Implementers are encouraged to use these partial URIs in their implementations, thereby providing a common look and feel to OGC APIs.

The OGC API — Common — Part 1 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:

```
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>
```

Figure 1 — Backus-Naur Definition of URI

The following rules should be used when interpreting the BNF for use with this standard:

- `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 categorized according to RFC 3986 and OGC Web API standards.

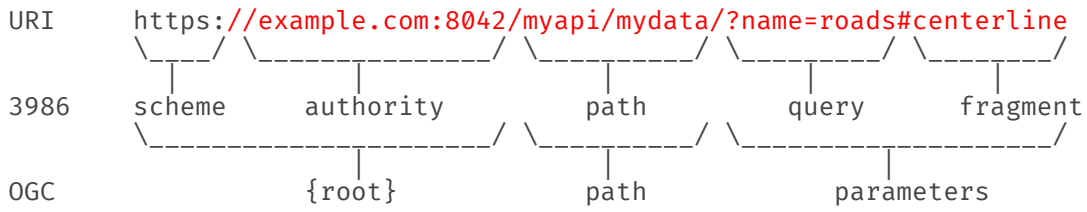

Figure 2 — Example URI and Components

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.

NOTEOGC Web API standards may include a community-defined identifier as part of a URI (for example: image id or feature id). Definition of the format of those identifiers is out of scope for these standards. Implementers should take care that these identifiers are properly encoded (see RFC 3986) in the URIs for all hosted resources.

Additional information on this topic is provided in the OGC API — Common — Users Guide.

6.3. Links

OGC Web API Standards use RFC 8288 (Web Linking) to express relationships between resources. Resource representations defined in these standards commonly include a “links” element. A “links” element is an array of individual hyperlink elements. These “links” elements provide a convention for associating related resources.

The individual hyperlink elements that make up a “links” element are defined using the following Hyperlink Schema.

```
type: object
required:
  - href
  - rel
properties:
  href:
    type: string
    description: Supplies the URI to a remote resource (or resource fragment).
    example: http://data.example.com/buildings/123
  rel:
    type: string
    description: The type or semantics of the relation.
    example: alternate
  type:
    type: string
    description: A hint indicating what the media type of the result of
dereferencing the link should be.
    example: application/geo+json
  hreflang:
    type: string
    description: A hint indicating what the language of the result of
dereferencing the link should be.
    example: en
  title:
    type: string
    description: Used to label the destination of a link such that it can be
used as a human-readable identifier.
    example: Trierer Strasse 70, 53115 Bonn
  length:
```

```
    type: integer
```

Figure 3 — Link Relation Schema

NOTE The `href` value is not restricted to absolute links. Relative links are also allowed.

In addition, links should be passed in the response using HTTP link headers. These links are accessible to the client without a need to process the resource.

RECOMMENDATION 1	
IDENTIFIER	`/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.

Additional information on this topic is provided in the OGC API — Common — Users Guide.

6.4. Link relations

Link relation types identify the semantics of a link. For example, a link with the relation type "service-meta" indicates that the current link context has service metadata at the link target.

Link relation types are expressed using the "rel" property from the Hyperlink Schema.

The "rel" property is populated using values from the IANA Link Relations Registry wherever possible. Additional values are registered with the OGC Link Relation Registry. Additional relation type values can be used if neither of these registers suffice.

The link relationships used in the OGC API — Common — Part 1: Core standard are described in Table 1. Additional relation types may be used if the implementation warrants it.

Table 1 — Link Relations

LINK RELATION	PURPOSE
`alternate`	Refers to a substitute for this context [IANA]. Refers to a representation of the current resource that is encoded using another media type (the media type is specified in the `type` link attribute).
`http://www.opengis.net/def/rel/ogc/1.0/data-meta`	Identifies general metadata for the context (dataset or collection) that is primarily intended for consumption by machines.

LINK RELATION	PURPOSE
http://www.opengis.net/def/rel/ogc/1.0/conformance	Refers to a resource that identifies the specifications that the link's context conforms to. [OGC]
describedby	Refers to a resource providing information about the link's context.[IANA] Links to external resources that further describe the subject resource
license	Refers to a license associated with this context. [IANA]
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]
service-meta	Identifies general metadata for the context that is primarily intended for consumption by machines. [IANA]

Additional information on the use of link relationships is provided in the OGC API — Common — Users Guide.

6.5. Use of HTTPS

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

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

6.6. API definition

6.6.1. General remarks

This OGC standard specifies requirements and recommendations for the development of APIs that share spatial resources while using 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.

So that client developers can more easily learn how to use the API, good documentation is essential. 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. The use of OpenAPI is one way to provide that machine-readable documentation.

6.6.2. Role of OpenAPI

This OGC API standard uses OpenAPI 3.0 fragments in 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 standard includes a conformance class for OGC API definitions that follow the OpenAPI specification 3.0. Alternative API definition languages are also allowed. Conformance classes for additional API definition languages will be added as the OGC API landscape continues to evolve.

6.6.3. 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.

6.6.4. 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.

Additional information on the use of OpenAPI as an API definition is provided in the OGC API — Common — Users Guide.

7

OVERVIEW

7 OVERVIEW

The OGC API — Common — Part 1: Core Standard defines common requirements and recommendations which are applicable to all OGC Web API Standards.

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 and early 2000s. However, technology has evolved. New Resource-Oriented APIs provide an alternative to the Service-Oriented Approach. New OGC Web API standards are under development to provide API alternatives to the OWS standards.

The OGC API — Common Standard specifies 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.

A detailed discussion of OGC Web Services and Web APIs can be found in the OGC API — Common — Users Guide.

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.

The modular Web API approach adopted by OGC API Standards has several facets:

- A common `core` that is recommended for all implementations of OGC API Standards. This OGC API — Common — Part 1: Core Standard defines this `core` in the Core Requirements Class.
- Common descriptive resources which allow clients to learn the purpose and capabilities of an API as well as how they should be used. These resources are defined in the Landing Page Requirements Class.

- Clear separation between common requirements and more resource specific capabilities. The OGC API — Common Standard specifies 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 that implements the OGC API – Features Standard should not be seen as a standalone service. Rather, this server should be viewed as a collection of API building blocks that together implement the capabilities that are specified in OGC API — Features. A corollary of this is that it should be possible to implement an API that simultaneously conforms to conformance classes from multiple current or future OGC API Standards.

7.3. Using APIs

OGC API Standards are expected to support two different approaches that clients may use when accessing an OGC conformant Web API.

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

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

8

CORE REQUIREMENTS CLASS

8 CORE REQUIREMENTS CLASS

The following requirements and recommendations define protocol-level conventions that should be applicable to all OGC Web APIs.

REQUIREMENTS CLASS 1	
IDENTIFIER	`http://www.opengis.net/spec/ogcapi-common-1/1.0/req/core`
TARGET TYPE	Web API
PREREQUISITES	RFC 7231 (HTTP/1.1) RFC 2818 (HTTP over TLS) RFC 8288 (Web Linking)

8.1. HTTP 1.1

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

REQUIREMENT 1	
IDENTIFIER	`/req/core/http`
A	OGC Web APIs SHALL conform to HTTP 1.1.
B	If the API supports HTTPS, then the API SHALL also conform to HTTP over TLS.

8.2. HTTP Status Codes

Table 2 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 2 — Typical HTTP status codes

STATUS CODE	DESCRIPTION
`200`	A successful request.
`302`	The target resource was found but resides temporarily under a different URI. A 302 response is not evidence that the operation has been successfully completed.
`303`	The server is redirecting the user agent to a different resource. A 303 response is not evidence that the operation has been successfully completed.
`304`	An entity tag was provided in the request and the resource has not changed since the previous request.
`307`	The target resource resides temporarily under a different URI and the user agent MUST NOT change the request method if it performs an automatic redirection to that URI.
`308`	Indicates that the target resource has been assigned a new permanent URI and any future references to this resource ought to use one of the enclosed URIs.
`400`	The server cannot or will not process the request due to an apparent client error. For example, a query parameter had an incorrect value.
`401`	The request requires user authentication. The response includes a `WWW-Authenticate` header field containing a challenge applicable to the requested resource.
`403`	The server understood the request, but is refusing to fulfill it. While status code `401` indicates missing or bad authentication, status code `403` indicates that authentication is not the issue, but the client is not authorized to perform the requested operation on the resource.
`404`	The requested resource does not exist on the server. For example, a path parameter had an incorrect value.
`405`	The request method is not supported. For example, a POST request was submitted, but the resource only supports GET requests.
`406`	Content negotiation failed. For example, the `Accept` header submitted in the request did not support any of the media types supported by the server for the requested resource.
`500`	An internal error occurred in the server.

The return status codes described in Table 2 do not cover all possible conditions. See IETF RFC 7231 for a complete list of HTTP status codes.

RECOMMENDATION 2	
IDENTIFIER	`/per/core/additional-status-codes`
A	Servers MAY implement additional capabilities provided by the HTTP protocol. Therefore, they MAY return status codes in addition to those listed in Table 2.

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

RECOMMENDATION 3	
IDENTIFIER	`/rec/core/problem-details`
A	An OGC Web API should include a "Problem Details" report in any error response in accordance with IETF RFC 7807.

8.3. Query parameters

8.3.1. Parameter Names

REQUIREMENT 2	
IDENTIFIER	`/req/core/query-param-name-unknown`
A	The server SHALL return a response with the status code `400` IF 1. the request URI includes a query parameter that is not specified in the API definition 2. /per/core/query-param-name-specified does not apply, and 3. /per/core/query-param-name-tolerance does not apply.

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

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

RECOMMENDATION 4	
IDENTIFIER	`/per/core/query-param-name-specified`

RECOMMENDATION 4	
A	The specification of a query parameter in the API definition MAY encompass a range of parameter names. Any query parameter that falls within the specified range can be considered “specified” in the API definition.
STATEMENT	Examples of a parameter range include: • A regular expression that defines the valid parameter names, • A URI Template segment that defines the valid parameter names, • An indication that all parameter names are accepted (no parameter validation).

RECOMMENDATION 5	
IDENTIFIER	`/per/core/query-param-name-tolerance`
A	Servers MAY display tolerance for requests with incorrect query parameter names. These acts of tolerance include: • Accept alternate capitalizations, spellings, and/or aliases of parameters, • Ignore unknown/unrecognized parameters, • Return a response with a status code of 30x redirecting the client to a more correct version of the request.
B	Servers should not be excessively tolerant. The response a client receives from the server should be a reasonable response for the request submitted.

8.3.2. Parameter Values

REQUIREMENT 3	
IDENTIFIER	`/req/core/query-param-value-invalid`
A	The server SHALL respond with a response with the status code `400` IF 1. the request URI includes a query parameter that has an invalid value and 2. /per/core/query-param-value-specified does not apply and 3. /per/core/query-param-value-tolerance does not apply.

The criteria for a parameter value to be considered “invalid” varies with the expressiveness of the API definition language used and the ability of the API server to tolerate errors.

A service implementer should endeavor to provide as much detail in the server’s API definition as the API definition language allows. However, there is no requirement to list every possible value that each query parameter might accept. Rather, the API implementation should include a reasonable degree of error recovery.

RECOMMENDATION 6	
IDENTIFIER	`/per/core/query-param-value-specified`
A	The specification of a query parameter in the API definition should include a definition of the valid values for that parameter. Any value that meets that criteria can be considered “specified” in the API definition.
B	The API definition language chosen may not be capable of expressing the desired range of values. In that case the server SHOULD provide: 1. A definition of the parameter values that best expresses the intended use of that parameter, 2. Additional human readable text documenting the actual range of validity.

RECOMMENDATION 7	
IDENTIFIER	`/per/core/query-param-value-tolerance`
A	Servers MAY display tolerance for requests where a parameter has an incorrect value. These acts of tolerance include: • substituting default values for invalid ones, • correcting formatting errors (e.g. convert integers to float).
B	Servers should not be excessively tolerant. The response a client receives from the server should be a reasonable response for the request submitted.

8.4. Web Caching

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

RECOMMENDATION 8	
IDENTIFIER	`/rec/core/etag`
A	The service SHOULD support entity tags and the associated headers as specified by HTTP 1.1.

8.5. 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 9	
IDENTIFIER	`/rec/core/cross-origin`
A	If the server is intended to be accessed from a browser, cross-origin requests SHOULD be supported. Note that support can also be added in a proxy layer on top of the server.

Two common mechanisms to support cross-origin requests are:

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

8.6. String Internationalization

If the server supports representing resources in multiple languages, the usual HTTP content negotiation mechanisms apply. The client states its language preferences in the `Accept-Language` header of a request and the server responds with responses that have linguistic text in the language that best matches the requested languages and the capabilities of the server.

RECOMMENDATION 10	
IDENTIFIER	`/rec/core/string-i18n`
A	For encodings that support string internationalization, the server SHOULD include information about the language for each string value that includes linguistic text.

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

The link object based on RFC 8288 (Web Linking) includes a `hreflang` attribute that can be used to state the language of the referenced resource. This can be used to include links to the same data in, for example, English or French. Just like with multiple encodings, a server that wants to use language-specific links will have to support a mechanism to mint language-specific URIs for resources in order to express links to, for example, the same resource in another

language. Again, this document does not mandate any particular approach how such a capability is supported by the server.

8.7. Resource Encodings

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

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

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

HTML encoding recommendation:

RECOMMENDATION 11	
IDENTIFIER	`/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 recommendation:

RECOMMENDATION 12	
IDENTIFIER	`/rec/core/json`
A	To support processing of an API with a web applet, implementations SHOULD consider supporting a JSON encoding.

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

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

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

As clients simply need to dereference the URI of the link, the implementation details and the mechanism of how the encoding is included in the URI of the link are not important. Developers interested in the approach of a particular implementation, such as manipulating ("hacking") URIs in the browser address bar, can study the API Definition document for that server.

Two common approaches are to use:

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

8.8. Parameter Encoding

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

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

The Backus-Naur Form (BNF) definition of a URI is provided in Annex Annex D.1.

8.8.1. Capitalization

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

REQUIREMENT 4	
IDENTIFIER	`/req/core/query-param-capitalization`
A	Parameter names and values SHALL be case sensitive.
B	IF a parameter name or value includes a percent encoded (escaped) character, THEN the upper case hexadecimal digits ("A" through "F") of that percent encoded character SHALL be equivalent to the lower case digits "a" through "f" respectively.

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

RECOMMENDATION 13	
IDENTIFIER	`/rec/core/query-param-capitalization`
A	Query parameter names SHOULD be in kebab case.
B	Query parameter values are usually reflective of the internal structure of the target resource. Unless otherwise specified, these values SHOULD be in kebab case.

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

8.8.2. Parameter Value Lists

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

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

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

REQUIREMENT 5	
IDENTIFIER	`/req/core/query-param-list-delimiter`
A	Parameters values containing lists SHOULD specify the delimiter to be used in the API definition.
B	The default list item delimiter SHALL be the comma (“,”).

REQUIREMENT 6	
IDENTIFIER	`/req/core/query-param-list-escape`
A	Any list item values that include a space or comma SHALL escape the space or comma character using the URI encoding rules from IETF RFC 3986

REQUIREMENT 7	
IDENTIFIER	`/req/core/query-param-list-empty`
A	All empty entries SHALL be represented by the empty string.

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

8.8.3. Numeric and Boolean Values

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

Boolean values conform to the following requirement.

REQUIREMENT 8	
IDENTIFIER	`/req/core/query-param-value-boolean`
A	Boolean values shall be represented by the lowercase strings “true” and “false”, representing Boolean true and false respectively.

Integer values conform to the following requirement.

REQUIREMENT 9	
IDENTIFIER	`/req/core/query-param-value-integer`
A	Integer values SHALL be represented by a finite-length sequence of decimal digits with an optional leading negative “-” sign. Positive values are assumed if the leading sign is omitted.

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

Decimal values conform to the following requirement.

REQUIREMENT 10	
IDENTIFIER	`/req/core/query-param-value-decimal`

REQUIREMENT 10	
A	Decimal values SHALL be represented by a finite-length sequence of decimal digits separated by a period as a decimal indicator. • An optional leading negative sign ("-") is allowed. • If the sign is omitted, positive ("+") is assumed. • Leading and trailing zeroes are optional. • If the fractional part is zero, the period and following zero(es) can be omitted.

Double values conform to the following requirement.

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

Special values, if supported, should conform to the following recommendation.

RECOMMENDATION 14	
IDENTIFIER	`/rec/core/query-param-value-special`
A	The special values positive and negative infinity and not-a-number SHOULD be represented using the strings `inf`, `-inf` and `nan`, respectively.

9

LANDING PAGE REQUIREMENTS CLASS

9 LANDING PAGE REQUIREMENTS CLASS

REQUIREMENTS CLASS 2	
IDENTIFIER	`http://www.opengis.net/spec/ogcapi-common-1/1.0/req/landing-page`
TARGET TYPE	Web API

The Landing Page Requirements Class of the OGC API — Common — Part 1: Core Standard describes how the `Landing Page` and its associated resources are accessed through an OGC conformant Web API.

The Landing Page resources are introduced in Table 3. The requirements and recommendations applicable to these resources are provided in the sections below.

Table 3 — Landing Page Resources

RESOURCE	HTTP METHOD	DESCRIPTION
Landing Page	GET	the landing page
API Definition	GET	the API Definition document for this API
Conformance Declaration	GET	the conformance information for this API

9.1. API landing page

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

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

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

While the three metadata elements are defined as text strings, the attribution element is special. Specifically, the attribution element can contain markup text. Markup allows a text string to

import images and format text. The capabilities are only limited by the markup language used. See the example landing page for an example of the use of markup in the attribution element.

9.1.1. Operation

REQUIREMENT 12	
IDENTIFIER	`/req/landing-page/root-op`
A	The server SHALL support the HTTP GET operation on the URI `{root}/`.
B	The response to the HTTP GET request issued in A SHALL satisfy requirement /req/landing-page/root-success.

9.1.2. Response

REQUIREMENT 13	
IDENTIFIER	`/req/landing-page/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.yaml and include links to the following resources: • API Definition (relation type 'service-desc' or 'service-doc') • Conformance Declaration (relation type 'http://www.opengis.net/def/rel/ogc/1.0/conformance')

The Landing Page returned by this operation is based on the following OpenAPI schema.

```
type: object
required:
  - links
properties:
  title:
    type: string
    title: The title of the API.
    description: While a title is not required, implementers are strongly
advised to include one.
    example: Buildings in Bonn
  description:
    type: string
    example: Access to data about buildings in the city of Bonn via a Web API
that conforms to the OGC API Common specification.
  attribution:
    type: string
```

```
    title: attribution for the API
    description: The `attribution` should be short and intended for
presentation to a user, for example, in a corner of a map. Parts of the text
can be links to other resources if additional information is needed. The
string can include HTML markup.
  links:
    type: array
    items:
      $ref: link.yaml
```

Figure 4 — Landing Page Schema

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

Examples of OGC API landing pages are provided in Annex B.1.

A JSON schema for the Landing Page resource can be found on the OGC Schema Repository.

9.1.3. Error Situations

See Clause 8.2 for general guidance.

9.2. API Definition

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

9.2.1. Operation

REQUIREMENT 14	
IDENTIFIER	`/req/landing-page/api-definition-op`
A	The server SHALL support the HTTP GET operation on all links from the landing page that have the relation type `service-desc`.
B	The server SHALL support the HTTP GET operation on all links from the landing page that have the relation type `service-doc`.
C	The responses to all HTTP GET requests issued in A and B SHALL satisfy requirement /req/landing-page/api-definition-success.

RECOMMENDATION 15	
IDENTIFIER	`/rec/landing-page/api-definition-op`
A	The server SHOULD support the HTTP GET operation on the URI `{root}/api`.
B	The response to the HTTP GET request issued in A SHOULD satisfy requirement /req/landing-page/api-definition-success.

9.2.2. Response

REQUIREMENT 15	
IDENTIFIER	`/req/landing-page/api-definition-success`
A	A successful execution of the operation SHALL be reported as a response with a HTTP status code `200`.
B	The content of that response SHALL be an API Definition document.
C	The API Definition document SHALL be consistent with the media type identified through HTTP content negotiation.

NOTE The `-f` parameter MAY be used to satisfy this requirement.

RECOMMENDATION 16	
IDENTIFIER	`/rec/landing-page/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.

9.2.3. Error Situations

See Clause 8.2 for general guidance.

9.3. Declaration of Conformance Classes

The OGC Web API Standards define a collection of modules that 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 implementers have a choice on the modules to use, there is no simple answer. The best that can be done is to provide a list of the modules implemented, a declaration of the Conformance Classes.

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

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

9.3.1. Operation

REQUIREMENT 16	
IDENTIFIER	`/req/landing-page/conformance-op`
A	The server SHALL support the HTTP GET operation on the URI `{root}/conformance`.
B	The server SHALL support the HTTP GET operation on all links from the landing page that have the relation type `http://www.opengis.net/def/rel/ogc/1.0/conformance`.
C	The responses to all HTTP GET requests issued in `A` and `B` SHALL satisfy requirement /req/landing-page/conformance-success.

9.3.2. Response

REQUIREMENT 17	
IDENTIFIER	`/req/landing-page/conformance-success`
A	A successful execution of the operation SHALL be reported as a response with a HTTP status code `200`.
B	The content of that response SHALL be based upon the schema confClasses.yaml and list all OGC API conformance classes that the API conforms to.

The Conformance Declaration returned by this operation is based on the following OpenAPI schema.

```
type: object
required:
  - conformsTo
properties:
  conformsTo:
    type: array
    items:
      type: string
      example: "http://www.opengis.net/spec/ogcapi-common-1/1.0/conf/core"
```

Figure 5 — Conformance Declaration Schema

A JSON schema for the Landing Page resource can be found on the OGC Schema Repository.

Examples of OGC Conformance Declarations are provided in Annex B.2.

9.3.3. Error situations

See Clause 8.2 for general guidance.

10

ENCODING REQUIREMENTS CLASSES

10 ENCODING REQUIREMENTS CLASSES

10.1. Overview

This clause specifies two requirements classes for encodings that may be used by an OGC Web API implementation. These encodings are commonly used for spatial data on the web applications:

- HTML
- JSON

Neither of these encodings are mandatory. An implementation of the OGC API — Common Standard may implement one, both, or none of them. Other encodings are possible.

10.2. Requirement Class “HTML”

Geographic information that is only accessible in formats such as GeoJSON or GML have two issues when web application principles are considered:

- The data are not discoverable using Web crawlers and search engines,
- The data cannot 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 3	
IDENTIFIER	`http://www.opengis.net/spec/ogcapi-common-1/1.0/req/html`
TARGET TYPE	Web API

REQUIREMENTS CLASS 3	
PREREQUISITES	Landing Page Requirements Class HTML5 Schema.org

REQUIREMENT 18	
IDENTIFIER	`/req/html/definition`
A	`200`-responses of the server SHALL support the `text/html` media type for the Landing Page and Conformance resources.

REQUIREMENT 19	
IDENTIFIER	`/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 17	
IDENTIFIER	`/rec/html/schema-org`
A	A `200`-response with the media type `text/html`, SHOULD include Schema.org annotations.

10.3. Requirement Class "JSON"

JSON is a lightweight data-interchange format designed to facilitate structured data interchange between applications. JSON is commonly used for Web-based software-to-software interchanges. Most Web developers are comfortable with using a JSON-based format. Therefore, support for JSON is recommended for machine-to-machine interactions.

REQUIREMENTS CLASS 4

IDENTIFIER	`http://www.opengis.net/spec/ogcapi-common-1/1.0/req/json`
TARGET TYPE	Web API
PREREQUISITES	Landing Page Requirements Class IETF RFC 8259: The JavaScript Object Notation (JSON) Data Interchange Format JSON Schema

REQUIREMENT 20

IDENTIFIER	`/req/json/definition`
A	`200`-responses of the server SHALL support the `application/json` media type for the Landing Page and Conformance resources.

REQUIREMENT 21

IDENTIFIER	`/req/json/content`
A	Every request to a Landing Page or Conformance resource which: 1. Receives a `200`-response 2. with the `Content-Type` header set to `application/json` SHALL include, or link to, a payload encoded according to the JSON Interchange Format
B	The payload for these responses SHALL conform with the JSON Schema specified for the resource in the Landing Page Requirements Class.

RECOMMENDATION 18

IDENTIFIER	`/rec/json/problem-details`
A	Any OGC Web API implementation instance returning an RFC 7807 “Problem Details” report in JSON should set the `Content-Type` header to “application/problem+json” and structure the report using the JSON Schema here.

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

An example JSON Problem Details report is available at ExceptionExample.yaml.

11

OPENAPI 3.0 REQUIREMENTS CLASS

11 OPENAPI 3.0 REQUIREMENTS CLASS

11.1. Basic requirements

APIs conforming to this requirements class are self-documenting using an OpenAPI Document.

REQUIREMENTS CLASS 5	
IDENTIFIER	`http://www.opengis.net/spec/ogcapi-common-1/1.0/req/oas30`
TARGET TYPE	Web API
PREREQUISITE	OpenAPI Specification 3.0

REQUIREMENT 22	
IDENTIFIER	`/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 23	
IDENTIFIER	`/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 24	
IDENTIFIER	`/req/oas30/oas-impl`
A	The API SHALL implement all capabilities specified in the OpenAPI definition.

11.2. Complete definition

REQUIREMENT 25	
IDENTIFIER	`/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 Clause 8.2.

Clients should 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 26	
IDENTIFIER	`/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 — 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
```

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 at the following levels:

- Root — applicable to the whole API except when overridden by Security Requirements defined at a lower level of the API.
- 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 27	
IDENTIFIER	`/req/oas30/security`
A	If the operations of the API are access-controlled, the security scheme(s) and requirements SHALL be documented in the OpenAPI definition.

11.5. Query Parameter Definition

The OpenAPI specification defines query parameters using the `Parameter` object with the `in` property set to "query". The parameter name is a literal value provided by the `name` property. Since the parameter names are literals, each parameter must be described separately.

OGC API — Common requires that all query parameters are specified in the API definition. In the case of a Feature server, this could mean that every property of every feature type must be described in the API definition, a requirement that few implementers would accept.

The OpenAPI specification provides a capability that allows additional parameters to be specified without explicitly declaring them. That is, parameters that have not been explicitly specified in the API definition for the operation will still be considered “specified” for purposes of validation (see /per/core/query-param-name-specified and /per/core/query-param-name-tolerance.

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

Figure 6 — OpenAPI schema for additional "free-form" query parameters

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 `freeFormParameters` is this object:

```
{
  "my_first_parameter": "some value",
  "my_other_parameter": 42
}
```

Figure 7 — Example "free-form" query parameter

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

11.6. Further Information

Additional guidance on using OpenAPI in OGC Web API implementations can be found in the OGC API — Common — Users Guide.

12

MEDIA TYPES

12 MEDIA TYPES

12.1. Normal Response Media Types

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

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

12.2. OpenAPI Media Types

The media types for an OpenAPI definition are `application/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.

12.3. Problem Details Media Types

OGC API — Common recommends that implementers use IETF RFC 7807 when constructing the response body for an error condition. The media types for an RFC 7807 Problem Details response body are:

- application/problem+json — for responses in JSON
- application/problem+xml — for responses in XML

A

ANNEX A (INFORMATIVE) ABSTRACT TEST SUITE (NORMATIVE)

A

ANNEX A (INFORMATIVE) ABSTRACT TEST SUITE (NORMATIVE)

A.1. Introduction

OGC Web APIs are not 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 that are not to be tested.

A.2. Conformance Class Core

CONFORMANCE CLASS A.1	
IDENTIFIER	`http://www.opengis.net/spec/ogcapi-common-1/1.0/conf/core`
SUBJECT	http://www.opengis.net/spec/ogcapi-common/1.0/req/core
TARGET TYPE	Web API

A.2.1. HTTP

ABSTRACT TEST A.1	
IDENTIFIER	`/conf/core/http`
SUBJECT	/req/core/http

ABSTRACT TEST A.1	
TEST PURPOSE	Validate that the resources advertised through the API can be accessed using the HTTP 1.1 protocol and, where appropriate, TLS.
TEST METHOD	1. All compliance tests shall be configured to use the HTTP 1.1 protocol exclusively. 2. For APIs that support HTTPS, all compliance tests shall be configured to use HTTP over TLS (RFC 2818) with their HTTP 1.1 protocol.

A.2.2. Query Parameters

ABSTRACT TEST A.2	
IDENTIFIER	`/conf/core/query-param-name-unknown`
SUBJECT	/req/core/query-param-name-unknown
TEST PURPOSE	Validate that an error is returned when a query parameter is used that has not been specified in the API definition.
TEST METHOD	DO FOR ALL operations advertised in the API definition 1. Execute that operation using a query parameter that is not specified in the API definition. 2. Validate that the operation returns a response with the status code `400`. DONE

NOTE 1 The parameter name chosen should not fall within the range allowed by /per/core/query-param-name-specified or /per/core/query-param-name-tolerance.

ABSTRACT TEST A.3	
IDENTIFIER	`/conf/core/query-param-value-invalid`
SUBJECT	/req/core/query-param-value-invalid
TEST PURPOSE	Validate that an error is returned when a query parameter contains a value that is not valid for that parameter.
TEST METHOD	DO FOR ALL query parameters advertised in the API definition DO FOR ALL operations for which that parameter is valid 1. Execute that operation using the query parameter with values that do not comply with the advertised constraints on those values. (Example: exceeding minimum or maximum values) 2. Validate that the operation returns a response with the status code `400`. DONE DONE

NOTE 2 The parameter values chosen should not fall within the range allowed by /per/core/query-param-value-specified or /per/core/query-param-value-tolerance.

ABSTRACT TEST A.4	
IDENTIFIER	`/conf/core/query-param-capitalization`
SUBJECT	/req/core/query-param-capitalization
TEST PURPOSE	Validate that a parameter name is correctly capitalized
TEST METHOD	DO FOR ALL query parameters advertised in the API definition DO FOR ALL operations for which that parameter is valid 1. Execute that operation using that query parameter capitalized as specified in the API definition 2. Validate that the operation returns a response with the status code `200`. 3. Execute that operation using that query parameter with capitalization inverse of that specified in the API definition (Example: “Range” vs. “rANGE”) 4. Validate that the operation returns a response with the status code `400`. DONE DONE

ABSTRACT TEST A.5	
IDENTIFIER	`/conf/core/query-param-list-delimiter`
SUBJECT	/req/core/query-param-list-delimiter
TEST PURPOSE	For every query parameter where the value may be a list, validate that the server uses the designated delimiter to parse the items from the list.
TEST METHOD	DO FOR ALL query parameters advertised in the API definition where the parameter value can be a list DO FOR ALL operations for which that parameter is valid 1. Generate a request using that parameter and a value that is a delimited list of items. Use a delimiter that is: a. As advertised in the API definition OR b. A comma (“,”) if no delimiter is advertised. 2. Validate that the server properly interprets that parameter value 3. Generate a request using that parameter and a value where: a. the value is a delimited list AND b. the delimiter used to construct the list is not valid for this server 4. Validate that the server returns a status code `400` DONE

ABSTRACT TEST A.5

DONE

ABSTRACT TEST A.6

IDENTIFIER	`/conf/core/query-param-list-escape`
SUBJECT	/req/core/query-param-list-escape
TEST PURPOSE	For every query parameter where the value may be a list, validate that all escaped space and comma characters that appear in the values are properly processed.
TEST METHOD	DO FOR ALL query parameters advertised in the API definition where the parameter value can be a list DO FOR ALL operations for which that parameter is valid 1. Generate a request using that parameter and a value that includes an escaped space or comma 2. Validate that the server properly interprets that parameter value 3. Generate a request using that parameter and a value that includes a non-escaped space or comma 4. Validate that the server returns a status code `400` DONE DONE

ABSTRACT TEST A.7

IDENTIFIER	`/conf/core/query-param-list-empty`
SUBJECT	/req/core/query-param-list-empty
TEST PURPOSE	For every query parameter where the value may be a list, validate that the server interprets an empty string as an empty list.
TEST METHOD	DO FOR ALL query parameters advertised in the API definition where the parameter value can be a list DO FOR ALL operations for which that parameter is valid 1. Generate a request using that parameter and an empty string for the parameter value. 2. Validate that the server properly interprets that parameter value as an empty list DONE DONE

ABSTRACT TEST A.8

IDENTIFIER	`/conf/core/query-param-value-boolean`
SUBJECT	/req/core/query-param-value-boolean
TEST PURPOSE	For every query parameter where the value is a boolean, validate that the server correctly interprets boolean values of “true” and “false”.
TEST METHOD	DO FOR ALL query parameters advertised in the API definition where the parameter value is a boolean DO FOR ALL operations for which that parameter is valid 1. Generate a request using that parameter and the string “true” for the parameter value 2. Validate that the server properly interprets that parameter value as true boolean value 3. Generate a request using that parameter and the string “false” for the parameter value 4. Validate that the server properly interprets that parameter value as false boolean value DONE DONE

ABSTRACT TEST A.9

IDENTIFIER	`/conf/core/query-param-value-integer`
SUBJECT	/req/core/query-param-value-integer
TEST PURPOSE	For every query parameter where the value is an integer, validate that the server correctly interprets properly encoded integer values.
TEST METHOD	DO FOR ALL query parameters advertised in the API definition where the parameter value is an integer DO FOR ALL operations for which that parameter is valid 1. Generate a request using that parameter and an unsigned numeric string for the parameter value 2. Validate that the server properly interprets that parameter value as an unsigned integer 3. Generate a request using that parameter and a negative signed numeric string for the parameter value 4. Validate that the server properly interprets that parameter value as a signed negative integer value DONE DONE

ABSTRACT TEST A.10

IDENTIFIER	`/conf/core/query-param-value-decimal`

ABSTRACT TEST A.10

SUBJECT	/req/core/query-param-value-decimal
TEST PURPOSE	For every query parameter where the value is a decimal, validate that the server correctly interprets properly encoded decimal values.
TEST METHOD	DO FOR ALL query parameters advertised in the API definition where the parameter value is a decimal number DO FOR ALL operations for which that parameter is valid 1. Generate a request using that parameter and a parameter value that is: a) an unsigned numeric string preceded by one or more zeros b) contains at least two factional digits followed by one or more zeros 2. Validate that the server properly interprets that parameter value as an unsigned decimal value 3. Generate a request using that parameter and a parameter value that is: a) a signed negative numeric string b) contains at least two factional digits followed by one or more zeros 4. Validate that the server properly interprets that parameter value as a signed negative decimal value 5. Generate a request using that parameter and a parameter value that: a) is an unsigned numeric string preceded by one or more zeros b) does not contain non-numeric characters (“.”) 6. Validate that the server properly interprets that parameter value as an unsigned decimal value with a fractional value of zero DONE DONE

ABSTRACT TEST A.11

IDENTIFIER	`/conf/core/query-param-value-double`
SUBJECT	/req/core/query-param-value-double
TEST PURPOSE	For every query parameter where the value is a double, validate that the server correctly interprets properly encoded double values.
TEST METHOD	DO FOR ALL query parameters advertised in the API definition where the parameter value is a double value DO FOR ALL operations for which that parameter is valid 1. Generate a request using that parameter and a parameter value that is: a) greater than 4,294,967,296 (“e” and exponent required) and b) contains at least two factional digits followed by one or more zeros 2. Validate that the server properly interprets that parameter value 3. Generate a request using that parameter and a parameter value that is: a) less than -4,294,967,296 (without “e” and exponent) and b) contains at least two factional digits followed by one or more zeros 4. Validate that the server properly interprets that parameter value 5. Generate a request using that parameter and a parameter value that is:

ABSTRACT TEST A.11
a) Less than 4,294,967,296 b) Greater than -4,294,967,295 c) Represented as a single decimal number 6. Validate that the server properly interprets that parameter value DONE DONE

A.3. Conformance Class Landing Page

CONFORMANCE CLASS A.2	
IDENTIFIER	`http://www.opengis.net/spec/ogcapi-common-1/1.0/conf/landing-page`
SUBJECT	http://www.opengis.net/spec/ogcapi-common/1.0/req/landing-page
PREREQUISITE	Core Conformance Class
TARGET TYPE	Web API

A.3.1. Landing Page

ABSTRACT TEST A.12	
IDENTIFIER	`/conf/landing-page/root-op`
SUBJECT	/req/landing-page/root-op /req/landing-page/root-success
TEST PURPOSE	Validate that a landing page can be retrieved from the expected location.
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 /conf/landing-page/root-success.

ABSTRACT TEST A.13	
IDENTIFIER	`/conf/landing-page/root-success`
SUBJECT	/req/landing-page/root-success
TEST PURPOSE	Validate that the landing page complies with the required structure and contents.
TEST METHOD	Validate the landing page for all supported media types using the resources and tests identified in Table A.1 For formats that require manual inspection, perform the following: 1. Validate that the landing page includes a “service-desc” and/or “service-doc” link to an API Definition 2. Validate that the landing page includes a “http://www.opengis.net/def/rel/ogc/1.0/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 A.1 — Schema and Tests for Landing Pages

FORMAT	SCHEMA DOCUMENT	TEST ID
HTML	landingPage.yaml	/conf/html/content
JSON	landingPage.yaml	/conf/json/content

A.3.2. API Definition

ABSTRACT TEST A.14	
IDENTIFIER	`/conf/landing-page/api-definition-op`
SUBJECT	/req/landing-page/api-definition-op /req/landing-page/api-definition-success
TEST PURPOSE	Validate that the API Definition document can be retrieved from the expected location.
TEST METHOD	DO FOR EACH `service-desc` and `service-doc` link on the landing page: 1. Issue an HTTP GET request for the link 2. Validate that a document was returned with a status code `200`

ABSTRACT TEST A.14	
	3. Validate the contents of the returned document using test /conf/landing-page/api-definition-success. DONE

ABSTRACT TEST A.15	
IDENTIFIER	`/conf/landing-page/api-definition-success`
SUBJECT	/req/landing-page/api-definition-success
TEST PURPOSE	Validate that the API Definition complies with the required structure and contents.
TEST METHOD	Validate the API Definition document against an appropriate schema document.

A.3.3. Conformance Declaration

ABSTRACT TEST A.16	
IDENTIFIER	`/conf/landing-page/conformance-op`
SUBJECT	/req/landing-page/conformance-op /req/landing-page/conformance-success
TEST PURPOSE	Validate that a Conformance Declaration can be retrieved from the expected locations.
TEST METHOD	DO FOR EACH `http://www.opengis.net/def/rel/ogc/1.0/conformance` link on the landing page: 1. Issue an HTTP GET request for the link 2. Validate that a document was returned with a status code `200` 3. Validate the contents of the returned document using test /conf/landing-page/conformance-success. DONE 1. Issue an HTTP GET request for the `{root}/conformance` path 2. Validate that a document was returned with a status code `200` 3. Validate the contents of the returned document using test /conf/landing-page/conformance-success.

ABSTRACT TEST A.17	
IDENTIFIER	`/conf/landing-page/conformance-success`
SUBJECT	/req/landing-page/conformance-success
TEST PURPOSE	Validate that the Conformance Declaration response complies with the required structure and contents.
TEST METHOD	1. Validate the response document against the schema confClasses.yaml 2. Validate that the document lists all OGC API conformance classes that the API implements.

A.4. Conformance Class JSON

CONFORMANCE CLASS A.3	
IDENTIFIER	`http://www.opengis.net/spec/ogcapi-common-1/1.0/conf/json`
SUBJECT	http://www.opengis.net/spec/ogcapi-common-1/1.0/req/json
PREREQUISITES	Core Conformance Class Landing Page Conformance Class
TARGET TYPE	Web API

A.4.1. JSON Definition

ABSTRACT TEST A.18	
IDENTIFIER	`/conf/json/definition`
SUBJECT	/req/json/definition /req/json/content
TEST PURPOSE	Verify support for JSON
TEST METHOD	DO FOR EACH resource and operation defined in the Landing Page Conformance Class: 1. Execute the operation specifying `application/json` as the media type 2. Validate that a document was returned with a status code `200` 3. Validate the contents of the returned document using test /conf/json/content.

ABSTRACT TEST A.18	
	DONE

A.4.2. JSON Content

ABSTRACT TEST A.19	
IDENTIFIER	`/conf/json/content`
SUBJECT	/req/json/content
TEST PURPOSE	Verify the content of a JSON document given an input document and schema.
TEST METHOD	1. Validate that the document is a JSON (IETF RFC 8259) document. 2. Validate the document against the schema using a JSON Schema validator.

A.5. Conformance Class HTML

CONFORMANCE CLASS A.4	
IDENTIFIER	`http://www.opengis.net/spec/ogcapi-common-1/1.0/conf/html`
SUBJECT	http://www.opengis.net/spec/ogcapi-common/1.0/req/html
PREREQUISITES	Core Conformance Class Landing Page Conformance Class
TARGET TYPE	Web API

A.5.1. HTML Definition

ABSTRACT TEST A.20	
IDENTIFIER	`/conf/html/definition`
SUBJECT	/req/html/definition

ABSTRACT TEST A.20	
	/req/html/content
TEST PURPOSE	Verify support for HTML
TEST METHOD	DO FOR EACH resource and operation defined in the Landing Page Conformance Class: 1. Execute the operation specifying `text/html` as the media type 2. Validate that a document was returned with a status code `200` 3. Validate the contents of the returned document using test /conf/html/content. DONE

A.5.2. HTML Content

ABSTRACT TEST A.21	
IDENTIFIER	`/conf/html/content`
SUBJECT	/req/html/content
TEST PURPOSE	Verify the content of an HTML document given an input document and schema.
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 A.5	
IDENTIFIER	`http://www.opengis.net/spec/ogcapi-common-1/1.0/conf/oas30`
SUBJECT	http://www.opengis.net/spec/ogcapi-common/1.0/req/oas30
PREREQUISITES	Core Conformance Class Landing Page Conformance Class
TARGET TYPE	Web API

ABSTRACT TEST A.22

IDENTIFIER	`/conf/oas30/completeness`
SUBJECT	/req/oas30/completeness
TEST PURPOSE	Verify the completeness of an OpenAPI document.
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 A.23

IDENTIFIER	`/conf/oas30/exceptions-codes`
SUBJECT	/req/oas30/exceptions-codes
TEST PURPOSE	Verify that the OpenAPI document fully describes potential exception codes.
TEST METHOD	Verify that for each operation, the OpenAPI document describes all HTTP Status Codes that may be generated.

ABSTRACT TEST A.24

IDENTIFIER	`/conf/oas30/oas-definition-1`
SUBJECT	/req/oas30/oas-definition-1
TEST PURPOSE	Verify that JSON and HTML versions of the OpenAPI document are available.
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 A.25

IDENTIFIER	`/conf/oas30/oas-definition-2`
SUBJECT	/req/oas30/oas-definition-2

ABSTRACT TEST A.25

TEST PURPOSE	Verify that the OpenAPI document is valid JSON.
TEST METHOD	Verify that the JSON representation conforms to the OpenAPI Specification, version 3.0.

ABSTRACT TEST A.26

IDENTIFIER	`/conf/oas30/oas-impl`
SUBJECT	/req/oas30/oas-impl
TEST PURPOSE	Verify that all capabilities specified in the OpenAPI definition are implemented by the API.
TEST METHOD	1. Construct an operation for each OpenAPI `Path` object including all server URL options, HTTP operations and enumerated path parameters. 2. Validate that each operation performs in accordance with the API definition.

ABSTRACT TEST A.27

IDENTIFIER	`/conf/oas30/security`
SUBJECT	/req/oas30/security
TEST PURPOSE	Verify that any authentication protocols implemented by the API are documented in the OpenAPI document.
TEST METHOD	1. Identify all authentication protocols supported by the API. 2. Validate that each authentication protocol is described in the OpenAPI document by a Security Schema Object and its use specified by a Security Requirement Object.

B

ANNEX B (INFORMATIVE) EXAMPLES (INFORMATIVE)

B

ANNEX B (INFORMATIVE) EXAMPLES (INFORMATIVE)

B.1. Example Landing Pages

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

- HTML
- JSON
- OGC API — Common — Part 2: Geospatial Data

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


```
{
    "title": "Example API Landing Page",
    "description": "This is an example of an API landing page in JSON format",
    "attribution": "<a href='www.ign.es' rel=' '>IGN</a> <a href='www.govdata.
de/dl-de/by-2-0'>(c)</a>",
    "links": [
        {
            "rel": "service-desc",
            "type": "application/json",
            "title": "API definition for this endpoint as JSON",
            "href": "http://www.example.com/oapi-c/api?f=application/json"
        },
        {
            "rel": "service-desc",
            "type": "text/html",
            "title": "API definition for this endpoint as HTML",
            "href": "http://www.example.com/oapi-c/api?f=text/html"
        },
        {
            "rel": "http://www.opengis.net/def/rel/ogc/1.0/data-meta",
            "type": "application/xml",
            "title": "ISO 19115 Metadata as XML",
            "href": "http://www.example.com/oapi-c/data-meta?f=application/xml"
        },
        {
            "rel": "http://www.opengis.net/def/rel/ogc/1.0/conformance",
            "type": "application/json",
            "title": "Conformance Declaration as JSON",
```

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


The example is presented below in YAML.


```
title: Example API Landing Page
description: This is an example of an API landing page in JSON format
```

```
attribution: "<a href='www.ign.es' rel=' '>IGN</a> <a href='www.govdata.de/dl-de/by-2-0'>(c)</a>"
links:
- rel: service-desc
  type: application/json
  title: API definition for this endpoint as JSON
  href: http://www.example.com/oapi-c/api?f=application/json
- rel: service-desc
  type: text/html
  title: API definition for this endpoint as HTML
  href: http://www.example.com/oapi-c/api?f=text/html
- rel: http://www.opengis.net/def/rel/ogc/1.0/data-meta
  type: application/xml
  title: ISO 19115 Metadata as XML
  href: http://www.example.com/oapi-c/data-meta?f=application/xml
- rel: http://www.opengis.net/def/rel/ogc/1.0/conformance
  type: application/json
  title: Conformance Declaration as JSON
  href: http://www.example.com/oapi-c/conformance?f=application/json
- rel: http://www.opengis.net/def/rel/ogc/1.0/conformance
  type: application/xml
  title: Conformance Declaration as XML
  href: http://www.example.com/oapi-c/conformance?f=application/xml
- rel: http://www.opengis.net/def/rel/ogc/1.0/conformance
  type: text/html
  title: Conformance Declaration as HTML
  href: http://www.example.com/oapi-c/conformance?f=text/html
- rel: http://www.opengis.net/def/rel/ogc/1.0/data
  type: application/json
  title: Collections Metadata as JSON
  href: http://www.example.com/oapi-c/collections?f=application/json
- rel: http://www.opengis.net/def/rel/ogc/1.0/data
  type: application/xml
  title: Collections Metadata as XML
  href: http://www.example.com/oapi-c/collections?f=application/xml
- rel: http://www.opengis.net/def/rel/ogc/1.0/data
  type: text/html
  title: Collections Metadata as HTML
  href: http://www.example.com/oapi-c/collections?f=text/html
- rel: self
  type: application/json
  title: This Document
  href: http://www.example.com/oapi-c?f=application/json
- rel: alternate
  type: application/xml
  title: This Document as XML
  href: http://www.example.com/oapi-c?f=application/xml
- rel: alternate
  type: text/html
  title: This Document as HTML
  href: http://www.example.com/oapi-c?f=text/html
```

B.2. Conformance Examples

Example — 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, as well as 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/landing-page",
    "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"
  ]
}
```

The example is presented below in YAML.

```
conformsTo:
- http://www.opengis.net/spec/ogcapi-common-1/1.0/req/core
- http://www.opengis.net/spec/ogcapi-common-1/1.0/req/landing-page
- http://www.opengis.net/spec/ogcapi-common-1/1.0/req/oas30
- http://www.opengis.net/spec/ogcapi-common-1/1.0/req/html
- http://www.opengis.net/spec/ogcapi-common-1/1.0/req/json
```

B.3. API Definition Examples

Example — JSON API Definition: This is an example of an API Definition response in JSON (YAML). It describes an implementation of the OGC API — Common — Part 1: Core Standard. This example also illustrates:

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

```
openapi: 3.0.2
info:
  title: >-
    A sample API conforming to the draft standard OGC API - Common - Part 1:
    Core
  version: 1.0.0
  description: >-
    'This is a sample OpenAPI definition of the draft standard OGC API -
Common - Part 1: Core.'
  contact:
    name: Acme Corporation
    email: info@example.org
    url: http://example.org/
  license:
    name: CC-BY 4.0 license
    url: https://creativecommons.org/licenses/by/4.0/
  x-keywords:
    - geospatial
    - data
    - api
servers:
```

```
  - url: https://data.example.org/
    description: Production server
  - url: https://dev.example.org/
    description: Development server
tags:
  - name: Capabilities
    description: essential characteristics of this API
  - name: server
    description: Information about the server hosting this API
    externalDocs:
      description: information
      url: https://example.com/sample_api/documentation
paths:
  /:
    get:
      description: >-
        The landing page provides links to the API definition and the
        conformance statements for this API.
      parameters:
        - $ref: '#/components/parameters/f'
      operationId: getLandingPage
      responses:
        '200':
          $ref: 'https://raw.githubusercontent.com/opengeospatial/ogcapi-
common/master/core/openapi/3.0/API-Common-Part-1_1_0.yaml#/components/responses
/LandingPage'
        '400':
          $ref: '#/components/responses/400'
        '500':
          $ref: '#/components/responses/500'
      summary: Landing page
      tags:
        - server
  /api:
    get:
      description: This document
      parameters:
        - $ref: '#/components/parameters/f'
      responses:
        '200':
          $ref: '#/components/responses/200'
        '400':
          $ref: '#/components/responses/400'
        default:
          $ref: '#/components/responses/400'
      summary: This document
      tags:
        - server
  /conformance:
    get:
      description: >-
        A list of all conformance classes specified in a standard that the
        server conforms to.
      parameters:
        - $ref: '#/components/parameters/f'
      operationId: getConformanceDeclaration
      responses:
        '200':
          $ref: 'https://raw.githubusercontent.com/opengeospatial/ogcapi-
common/master/core/openapi/3.0/API-Common-Part-1_1_0.yaml#/components/responses
/ConfClasses'
        '400':
          $ref: '#/components/responses/400'
```

```
        '500':
          $ref: '#/components/responses/500'
      summary: API conformance definition
      tags:
        - server
components:
  parameters:
    f:
      description: >-
        The optional f parameter indicates the output format that the server
        shall provide as part of the response document.  The default format is
JSON.
      explode: false
      in: query
      name: f
      required: false
      schema:
        default: json
        enum:
          - json
          - html
        type: string
      style: form
  responses:
    '200':
      description: successful operation
    '400':
      $ref: 'https://raw.githubusercontent.com/opengeospatial/ogcapi-common/
master/core/openapi/3.0/API-Common-Part-1_1_0.yaml#/components/responses/400'
    '500':
      $ref: 'https://raw.githubusercontent.com/opengeospatial/ogcapi-common/
master/core/openapi/3.0/API-Common-Part-1_1_0.yaml#/components/responses/500'
```

B.4. Service Metadata Examples

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

```
info:
  title: My Web API
  version: 1.0.0
  description: This example shows population of an OpenAPI Info element with
identifying
    metadata for both the service and the service provider.
  contact:
    name: Acme Corporation
    email: info@example.org
    url: http://example.org/
    x-OGC-serviceContact:
      individualName: John Smith
      positionName: System Administrator
      role: pointOfcontact
      hoursOfService: 24 Hours
      contractInstructions: None
      onlineResource: http://example.org/contact
      address:
        deliveryPoint: 123 Any Street
        city: Boston
```

```
      administrativeArea: MA
      postalCode: '12345'
      country: USA
      electronicMailAddress: smith.j@example.org
    telephone:
      voice: "+1.123.456.7890"
      facsimile: "+1.123.456.7890 "
license:
  name: CC-BY 4.0 license
  url: https://creativecommons.org/licenses/by/4.0/
x-serviceType: http://www.opengis.net/doc/IS/ogcapi-common-1/1.0
x-serviceTypeVersion: '1.0'
x-profile: DGIWG
x-keywords:
- geospatial
- data
- api
x-fees: None
x-accessConstraints: None
```

C

ANNEX C (INFORMATIVE) GLOSSARY

C ANNEX C (INFORMATIVE) GLOSSARY

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

NOTE 1 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 08-131)

NOTE 2 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 08-131)

NOTE 3"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 08-131)

- **Requirements Class**

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

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

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

NOTE 4 The standardization target is the entity that may receive a certificate of conformance for a requirements class.

D

ANNEX D (INFORMATIVE) BACKUS-NAUR FORMS

ANNEX D (INFORMATIVE) BACKUS-NAUR FORMS

D.1. BNF for URI

The following Augmented Backus-Naur Form (ABNF) is from Appendix A of IETF RFC 3986.

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

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

URI-reference = URI / relative-ref

absolute-URI  = scheme ":" hier-part [ "?" query ]

relative-ref  = relative-part [ "?" query ] [ "#" fragment ]

relative-part = "//" authority path-abempty
                / path-absolute
                / path-noscheme
                / path-empty

scheme        = ALPHA *( ALPHA / DIGIT / "+" / "-" / "." )

authority     = [ userinfo "@" ] host [ ":" port ]
userinfo      = *( unreserved / pct-encoded / sub-delims / ":" )
host          = IP-literal / IPv4address / reg-name
port          = *DIGIT

IP-literal    = "[" ( IPv6address / IPvFuture  ) "]"

IPvFuture     = "v" 1*HEXDIG "." 1*( unreserved / sub-delims / ":" )

IPv6address   =                            6( h16 ":" ) ls32
              /                       "::" 5( h16 ":" ) ls32
              / [               h16 ] "::" 4( h16 ":" ) ls32
              / [ *1( h16 ":" ) h16 ] "::" 3( h16 ":" ) ls32
              / [ *2( h16 ":" ) h16 ] "::" 2( h16 ":" ) ls32
              / [ *3( h16 ":" ) h16 ] "::"    h16 ":"   ls32
              / [ *4( h16 ":" ) h16 ] "::"              ls32
              / [ *5( h16 ":" ) h16 ] "::"              h16
              / [ *6( h16 ":" ) h16 ] "::"
```

```
h16           = 1*4HEXDIG
ls32          = ( h16 ":" h16 ) / IPv4address
IPv4address   = dec-octet "." dec-octet "." dec-octet "."

dec-octet     = DIGIT                 ; 0-9
              / %x31-39 DIGIT         ; 10-99
              / "1" 2DIGIT            ; 100-199
              / "2" %x30-34 DIGIT     ; 200-249
              / "25" %x30-35          ; 250-255

reg-name      = *( unreserved / pct-encoded / sub-delims )

path          = path-abempty    ; begins with "/" or is empty
              / path-absolute   ; begins with "/" but not "//"
              / path-noscheme   ; begins with a non-colon segment
              / path-rootless   ; begins with a segment
              / path-empty      ; zero characters

path-abempty  = *( "/" segment )
path-absolute = "/" [ segment-nz *( "/" segment ) ]
path-noscheme = segment-nz-nc *( "/" segment )
path-rootless = segment-nz *( "/" segment )
path-empty    = 0<pchar>

segment       = *pchar
segment-nz    = 1*pchar
segment-nz-nc = 1*( unreserved / pct-encoded / sub-delims / "@" )
              ; non-zero-length segment without any colon ":"

pchar         = unreserved / pct-encoded / sub-delims / ":" / "@"

query         = *( pchar / "/" / "?" )

fragment      = *( pchar / "/" / "?" )

pct-encoded   = "%" HEXDIG HEXDIG

unreserved    = ALPHA / DIGIT / "-" / "." / "_" / "~"
reserved      = gen-delims / sub-delims
gen-delims    = ":" / "/" / "?" / "#" / "[" / "]" / "@"
sub-delims    = "!" / "$" / "&" / "'" / "(" / ")"
              / "*" / "+" / "," / ";" / "="
```

Figure D.1 — Backus-Naur Form for URI

E

ANNEX E (INFORMATIVE) OGC WEB API GUIDELINES

ANNEX E (INFORMATIVE) OGC WEB API GUIDELINES

The following table discusses how this standard addresses the design principles documented in the OGC Web API Guidelines.

Table E.1 — OGC Web API Guidelines

#	PRINCIPLE	DISCUSSION
1	**Don't reinvent**	Great care was taken in the development of this standard to only address capabilities that were not already standardized and to define how the needed capabilities integrate into a single API.
2	**Keep it simple and intuitive**	OGC Web APIs are developed using a building block approach. Conformance Classes are defined that encompass requirements sufficient to create a usable software module and no more. Complex APIs are constructed by assembling the applicable Conformance Classes.
3	**Use well-known resource types**	Except where unique to a specific Conformance Class, all resource types are IANA or OGC registered types. OGC Web API standards do not mandate an encoding. The encodings supported by an API are specified by the corresponding encoding Conformance Classes. All encodings used to-date are IANA registered media types.
4	**Construct consistent URIs**	OGC Web APIs are built from standardized modules using standardized patterns. This modular approach assures that the URIs are consistent across OGC Web APIs. OGC API — Common defines stylistic conventions for query parameters, query values, identifiers, and path elements used to create OGC Web API URIs.
5	**Use HTTP methods consistent with RFC 7231**	OGC web APIs are restricted to the HTTP methods defined in IETF RFC 7231.
6	**Put selection criteria behind the '?'**	Section 6.1 of this Standard defines the conventions to be used when creating URIs for OGC Web API standards. This includes the use of the "?" to separate query parameters from the rest of the URI. Note that this does not preclude the use of resource identifiers (ex. collection identifiers) as part of the path. However those can be considered identifying criteria rather than selection criteria.
7	**Error handling and use of HTTP status codes**	This standard identifies the applicable HTTP status codes and under what conditions they should be returned. Status codes and supporting information are returned in the HTTP response using a reporting structure based on RFC 7807.

#	PRINCIPLE	DISCUSSION
8	**Use explicit list of HTTP status codes**	Clause 8.2 provides a list of the HTTP status codes that implementers of this standard should be prepared to generate and accept. This list is not exhaustive (see guideline #1).
9	**Use of HTTP header**	OGC API — Common does not preclude use of HTTP headers where it is appropriate to do so. Only standard HTTP headers are used. Due to the common use of the HATEOAS pattern in OGC Web APIs, HTTP headers are not always accessible. The use of query parameter overrides is allowed.
10	**Allow flexible content negotiation**	IETF RFC 7231 content negotiation is available on all transactions. Since the HTTP headers are not always accessible, content negotiation may be performed through a query parameter (see #9).
11	**Pagination**	Of the resources defined in OGC API — Common — Part 1: Core only the conformance resource is "listable". We do not anticipate the conformance resource to grow to any size, so support for pagenation would add complexity will little to no value (violating #2) If an OpenAPI document is used as the API definition, then pagination could become an issue for this resource. The question of how to handle large Open API documents is still an open issue being worked across the Standards Working Groups.
12	**Processing resources**	Processing resources are not addressed by this Standard.
13	**Support metadata**	Support for metadata is provided through metadata resource links. Examples include links with the relation type `service-desc`,`service-doc`, `service-meta`, or `data-meta`.
14	**Consider your security needs**	While not mandated, use of HTTPS vs. HTTP is encouraged throughout this standard. Authenitcation is not precluded by this standard, but in keeping with guideline #1, this standard does not presume to dictate what authentication methods can be used. OGC API — Common — Part 1: Core only defines GET requests. The security issues associated with CRUD are not applicable to this standard.
15	**API description**	The API definition is available using the `service-desc` (machine readable) and `service-doc` (human readable) associations from the landing page. OpenAPI is the only API definition type currently supported.
16	**Use well-known identifiers**	IANA identifiers are used where they are available. Where no IANA identifiers are appropriate, OGC registered identifiers are used. OGC identifiers are only used after they have been reviewed and approved by the OGC Naming Authority.
17	**Use explicit relations**	All relations in this standard are typed using relation types registered in the IANA or the OGC relation type registers.

#	PRINCIPLE	DISCUSSION
18	**Support W3C cross-origin resource sharing**	This guideline is addressed in Clause 8.5.
19	**Resource encodings**	Conformance classes for both HTML and JSON have been defined. Implementation of both the HTML and JSON Conformance Classes is recommended.
20	**Good APIs are testable from the beginning**	The Abstract Test Suite (ATS) for this standard is provided in Annex A. The ATS is defined to sufficient level of detail to validate that it is implementable and comprehensive
21	**Specify whether operations are safe and/or idempotent**	According to IETF RFC 7231 “the GET, HEAD, OPTIONS, and TRACE methods are defined to be safe.” and the “PUT, DELETE, and safe request methods are idempotent”. All request methods in this standard (GET) are both safe and idempotent.
22	**Make resources discoverable**	All resources defined in this standard can be navigated to through resource links and optional standard paths. All resource links are typed using registered relation types. These links are encoded using a standard link structure that includes the media type, language, and title of the resource.
23	**Make default behavior explicit**	This Standard defines the proper and allowed responses for any valid or invalid request.

F

ANNEX F (INFORMATIVE) REVISION HISTORY

ANNEX F (INFORMATIVE) REVISION HISTORY

Table F.1 — 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.
2021-04-12	Public Comments	C. Heazel	all	Second public comment period.
2021-08-01	Final Review	C. Heazel	all	Ready for SWG review and TC vote
2022-11-04	1.0.0	G. Hobona	all	Final OGC Staff review

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] Whiteside, A., Greenwood, J.: **OGC Web Services Common Standard**, version 2.0, OGC 06-121r9

[2] Open Geospatial Consortium: **The Specification Model — A Standard for Modular specifications**, OGC 08-131

[3] Schema.org: http://schema.org/docs/schemas.html

[4] W3C: **Architecture of the World Wide Web, Volume One**, W3C Recommendation, 15 December 2004, https://www.w3.org/TR/webarch/

[5] W3C: **Data Catalog Vocabulary (DCAT) — Version 2**, W3C Recommendation, 04 February 2020, https://www.w3.org/TR/vocab-dcat-2/

[6] W3C: **Data on the Web Best Practices**, W3C Recommendation, 31 January 2017, https://www.w3.org/TR/dwbp/

[7] W3C/OGC: **Spatial Data on the Web Best Practices**, W3C Working Group Note, 28 September 2017, https://www.w3.org/TR/sdw-bp/

[8] W3C: **HTML5, W3C Recommendation**, http://www.w3.org/TR/html5/

[9] W3C Recommendation: **XML Schema Part 2: Datatypes Second Edition**, 28 October 2004, https://www.w3.org/TR/xmlschema-2/

[10] 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

[11] Kernighan, B., Richie, D.: **The C Programming Language**, Bell Laboratories, 1978

[12] IANA: **Link Relation Types**, https://www.iana.org/assignments/link-relations/link-relations.xml

[13] Fielding, R., Reschke, J.: **IETF RFC 7230, Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing**, https://tools.ietf.org/rfc/rfc7230.txt

[14] Fielding, R., Reschke, J.: **IETF RFC 7235, Hypertext Transfer Protocol (HTTP/1.1): Authentication**, https://tools.ietf.org/rfc/rfc7235.txt

[15] Reschke, J.: **IETF RFC 7538, The Hypertext Transfer Protocol Status Code 308 (Permanent Redirect)**, https://tools.ietf.org/rfc/rfc7538.txt

[16] ISO 15836-2:2019, **Information and documentation — The Dublin Core metadata element set — Part 2: DCMI Properties and classes**

[17] json-schema-org: **JSON Schema,** September 2019, https://json-schema.org/specification.html

[18] ISO/IEC 14977:1996(E) **Information technology – Syntactic metalanguage – Extended BNF**, available from ISO.