OGC API - Common - Part 1

Core

Open Geospatial Consortium

Submission Date: <yyyy-mm-dd>

Approval Date: <yyyy-mm-dd>

Publication Date: 2020-11-25

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

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

Version: 0.0.9

Category: OGC® Implementation Standard

Editor: Charles Heazel

OGC API - Common - Part 1: Core

Document type:	OGC® Implementation Standard
Document stage:	Draft
Document language:	English

License Agreement

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

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

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

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

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

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

Table of Contents

Chapter 1. Introduction

i. Abstract

The OGC has extended their suite of standards to include Resource Oriented Architectures and Web APIs. In the course of developing these standards, some practices proved to be common across more than one of those standards. These common practices are documented in the OGC API - Common suite of standards. API-Common standards serve as reusable building-blocks. Developers of OGC standards will use these building-blocks in the construction of OGC Web API Standards. The result is a modular suite of coherent API standards which can be adapted by a system designer for the unique requirements of their system.

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

Table 1. Common Core Resources

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

ii. Keywords

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

ogcdoc, OGC document, geographic information, spatial data, API, json, html, OpenAPI, REST, Common

iii. Preface

OGC Declaration

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

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

iv. Submitting organizations

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

- Ecere Corporation
- Heazeltech LLC
- Hexagon
- 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

v. Submitters

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

Name	**Affiliation**
Charles Heazel *(editor)*	Heazeltech
David Blodgett	U.S. Geological Survey
Sylvester Hagler	U.S. National Geospatial-Intelligence Agency
Jeffrey Harrison	U.S. Army Geospatial Center
Frédéric Houbie	Hexagon
Jérôme Jacovella-St-Louis	Ecere Corporation
Chris Little	U.K. Met Office
Joan Masó	UAB-CREAF
Donald Sullivan	NASA

Chapter 2. Scope

This standard addresses Discovery operations directed against the API itself. 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.

This standard provides the first stop for clients seeking to understand and use a new Web API. Use of this standard is strongly recommended for and new implementation of OGC Web API stadards. However, in keeping with the OGC's principle of modular API standards, use of this standard is not required of OGC conformant Web API implementations.

Chapter 3. Conformance

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

This standard addresses one Standardization Target; Web APIs

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

This standard identifies four conformance classes. The conformance classes implemented by an OGC API are advertised through the `/conformance` resource on the landing page. Each conformance class is defined by one requirements class. The tests in Annex A are organized by Requirements Class. So an implementation of the *Core* conformance class must pass all tests specified in Annex A for the *Core* requirements class.

3.1. Core Requirements Class

The *Core Requirements Class* provides a minimal useful service interface for an OGC Web API. The requirements specified in this requirements class are recommended for all OGC Web APIs.

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

3.2. Encoding Requirements Classes

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

- HTML
- JSON

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

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

3.3. OpenAPI 3.0 Requirements Class

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

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

Chapter 4. References

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

- Rescorla, E.: **IETF RFC 2818, HTTP Over TLS**, http://tools.ietf.org/rfc/rfc2818.txt
- Klyne, G., Newman, C.: **IETF RFC 3339, Date and Time on the Internet: Timestamps**, http://tools.ietf.org/rfc/rfc3339.txt
- Berners-Lee, T., Fielding, R., Masinter, L: **IETF RFC 3986, Uniform Resource Identifier (URI): Generic Syntax**, http://tools.ietf.org/rfc/rfc3896.txt
- Greforio, J., Fielding, R., Hadley, M., Nottingham, M., Orchard, D.: **IETF RFC 6570, URI Template**, https://tools.ietf.org/html/rfc6570
- Fielding, R., Reschke, J.: **IETF RFC 7230, Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing**, https://tools.ietf.org/rfc/rfc7230.txt
- Fielding, R., Reschke, J.: **IETF RFC 7231, Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content**, https://tools.ietf.org/rfc/rfc7231.txt
- Fielding, R., Reschke, J.: **IETF RFC 7232, Hypertext Transfer Protocol (HTTP/1.1): Conditional Requests**, https://tools.ietf.org/rfc/rfc7232.txt
- Fielding, R., Reschke, J.: **IETF RFC 7235, Hypertext Transfer Protocol (HTTP/1.1): Authentication**, https://tools.ietf.org/rfc/rfc7235.txt
- Reschke, J.: **IETF RFC 7538, The Hypertext Transfer Protocol Status Code 308 (Permanent Redirect)**, https://tools.ietf.org/rfc/rfc7538.txt
- Bray, T.: **IETF RFC 8259, The JavaScript Object Notation (JSON) Data Interchange Format**, http://tools.ietf.org/rfc/rfc8259.txt
- Nottingham, M.: **IETF RFC 8288, Web Linking**, http://tools.ietf.org/rfc/rfc8288.txt
- json-schema-org: **JSON Schema,** September 2019, https://json-schema.org/specification.html
- Open API Initiative: **OpenAPI Specification 3.0.3**, https://github.com/OAI/OpenAPI-Specification/blob/master/versions/3.0.3.md
- Whiteside, A., Greenwood, J.: **OGC Web Services Common Standard**, version 2.0, OGC 06-121r9
- W3C Recommendation: **XML Schema Part 2: Datatypes Second Edition**, 28 October 2004, https://www.w3.org/TR/xmlschema-2/

Chapter 5. Terms and Definitions

This document uses the terms defined in Sub-clause 5.3 of OGC Web Services Common (OGC 06-121r9), which is based on the ISO/IEC Directives, Part 2, Rules for the structure and drafting of International Standards. In particular, the word “shall” (not “must”) is the verb form used to indicate a requirement to be strictly followed to conform to this standard.

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

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

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

- **Resource**

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

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

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

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

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

Chapter 6. Conventions

6.1. Web API Fundamentals

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

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

6.2. Identifiers

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

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

The normative provisions in this draft standard are denoted by the URI `http://www.opengis.net/spec/ogcapi-common-1/1.0`. All Requirements, Conformance Modules, and Conformance Classes that appear in this document are denoted by partial URIs that are relative to this base.

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

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

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

Summary for Developers:

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

Backus-Naur Definition of URI

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

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

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

        path-abempty  = *( "/" segment )

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

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

        path-empty    = 0<pchar>
```

The following rules should be used when interpreting the BNF 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 categorised according to RFC 3986 and OGC Web API standards.

Example URI and Components

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

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

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

6.3. Link relations

RFC 8288 (Web Linking) is used to express relationships between resources. Link relation types from the IANA Link Relations Registry are used wherever possible. Additional link relation types are registered with the OGC Link Relation Registry.

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

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

Permission 1	**/per/core/additional-link-relations**
Servers MAY link to resource types in addition to those identified in Table 2.	
A	Those links SHOULD use link relation types defined in the IANA or OGC registries.
B	Those links SHOULD NOT use any of the link relation types identified in Table 2.

Table 2. Link Relations

Link Relation	Purpose
`alternate`	Refers to a substitute for this context [IANA]. Refers to a representation of the current resource which is encoded using another media type (the media type is specified in the `type` link attribute).
`collection`	The target IRI points to a resource which represents the collection resource for the context IRI. [IANA]
`current`	Refers to a resource containing the most recent item(s) in a collection of resources. [IANA]
`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]
`http://www.opengis.net/def/rel/ogc/1.0/data`	Refers to the root resource of a dataset in an API. [OGC]
`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. [OGC]
`describedby`	Refers to a resource providing information about the link's context.[IANA] Links to external resources which further describe the subject resource

Link Relation	Purpose
`first`	An IRI that refers to the furthest preceding resource in a series of resources.[IANA]
`item`	The target IRI points to a resource that is a member of the collection represented by the context IRI.[IANA]
`http://www.opengis.net/def/rel/ogc/1.0/items`	Refers to a resource that is comprised of members of the collection represented by the link's context. [OGC]
`license`	Refers to a license associated with this context. [IANA]
`next`	Indicates that the link's context is a part of a series, and that the next in the series is the link target. [IANA]
`prev`	Indicates that the link's context is a part of a series, and that the previous in the series is the link target. [IANA]
`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]
`start`	Refers to the first resource in a collection of resources.[IANA]

OGC API hyperlinks are defined using the following Hyperlink Schema.

```
{
    "$schema": "http://json-schema.org/draft-07/schema#",
    "title": "Link Schema",
    "description": "Schema for external references",
    "type": "object",
    "required": [
        "href",
        "rel"
        ],
    "properties": {
        "href": {
            "type": "string",
            "example": "http://data.example.com/buildings/123"
            },
        "rel": {
            "type": "string",
            "example": "alternate"
            },
        "type": {
            "type": "string",
            "example": "application/geo+json"
            },
        "hreflang": {
            "type": "string",
            "example": "en"
            },
        "title": {
            "type": "string",
            "example": "Trierer Strasse 70, 53115 Bonn"
            },
        "length": {
            "type": "integer"
            }
        }
    }
```

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	**/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 the use of link relationships is provided in the OGC API - Common Users Guide.

6.4. Use of HTTPS

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

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

6.5. API definition

6.5.1. General remarks

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

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

6.5.2. Role of OpenAPI

This standard uses OpenAPI 3.0 fragments in its' 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 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.

6.5.3. References to OpenAPI components in normative statements

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

component in the OGC schema repository.

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

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

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

6.5.4. Reusable OpenAPI components

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

Chapter 7. Overview

The OGC API - Common - Part 1: Core Standard defines the resources and access mechanisms which are useful for a client seeking to understand the offerings and capabilities of an API. These resources and their access mechanisms are described in Table 3.

Table 3. Common Core Resources

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

7.1. Evolution from OGC Web Services

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

The OGC API - Common suite of standards specify common modules for defining OGC Web API standards that follow the current Web architecture. In particular, the recommendations as defined in the W3C/OGC best practices for sharing Spatial Data on the Web as well as the W3C best practices for sharing Data on the Web.

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 OGC Modular Web API approach has several facets:

- A common `core` which is recommended for all OGC Web API implementations. This OGC API - Common - Part 1: Core Standard provides the information needed by a client to understand and use an OGC Web API.
- Clear separation between common requirements and more resource specific capabilities. The OGC API - Common suite of standards specify the *common* requirements that may be relevant to almost anyone who wants to build an API for spatial resources. Resource-specific requirements are addressed in resource-specific OGC standards.

- Technologies that change more frequently are decoupled and specified in separate modules ("conformance classes" in OGC terminology). This enables, for example, the use/re-use of new encodings for spatial data or API descriptions.
- Modularization is not just about a single "service". OGC APIs provide building blocks that can be reused in APIs in general. In other words, a server supporting the OGC-Feature API should not be seen as a standalone service. Rather, this server should be viewed as a collection of API building blocks which together implement API-Feature capabilities. A corollary of this is that it should be possible to implement an API that simultaneously conforms to conformance classes from the Feature, Coverage, and other current or future OGC Web API standards.

A more detailed discussion of modular APIs can be found in the OGC API - Common Users Guide.

7.3. Using APIs

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

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

The other approach targets developers that are not familiar with the OGC API standards, but want to interact with spatial data provided by an API that happens to implement OGC API 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 the API. This assumes familiarity with the API definition language and the related tooling, but it should not be necessary to study the OGC API standards.

Chapter 8. Requirement Class "Core"

Requirements Class	
http://www.opengis.net/spec/ogcapi-common-1/1.0/req/core	
Target type	Web API

The Core Requirements Class of the API-Common Core Standard describes how `core` resources are accessed through an OGC conformant Web API. The requirements that make up this requirements class are grouped into two categories. General requirements are those requirements which are applicable regardless of the resource being accessed. Resource requirements are the requirements which define the `core` resources and their applicable constraints.

8.1. General Requirements

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

8.1.1. HTTP 1.1

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

Requirement 1	/req/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.

8.1.2. HTTP Status Codes

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

Table 4. Typical HTTP status codes

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

Status code	Description
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.

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

Permission 2	/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 4.

8.1.3. Query parameters

Validation of query parameters in the request URL is complicated. Ideally, the parameter names and valid values would be completely defined in the API definition. However, there are a number of reasons why this may not be possible.

OGC Web API standards do not mandate a language nor data model for the API defintion. While OpenAPI does support this level of detail, other API description languages may not. Therefore, an API standard cannot assume the presence of a capabilty which iteself is not required.

Provision of this level of detail may result in an API which is difficult to use or maintain. Consider an API which allows queries against resource properties. A full specification of the query parameters would list all possible combinations of Feature Type and Property as well as the valid

values for each. If the hosted content is subject to change, then each change would have to be reflected in the API definition. Implementors are unlikely to invest the resource needed to provide and maintain this level of detail. Likewise, client implementors would be reluctant to parse and extract this information. The increased complexity does not provide a comprable value to implementors or users.

A requirement for complete documentation of all query parameters would limit the abilty of a server to implement soft failures. For example, an API server may be able to identify and correct simple errors such as misspellings and capitalization. An OGC Web API standard should not preclude provision of this capabilty.

To accomodate this variety, the requirements for query parameter validation are defined around three casses:

- CASE A: The query parameter is fully described in the API definition
- CASE B: The query parameter is partially described in the API documentation
- CASE C: The server is able to detect and correct errors in the query parameter

CASE A:

Requirement 2	**/req/core/query-param-known**
IF the API definition of the query parameters specifies explicit parameter names AND a query parameter name matches one of those names THEN IF the request URI includes a query parameter value that is not valid for that parameter THEN The server SHALL return a response with the status code 400. ELSE The server SHALL meet the requirements specified for that parameter.	

This is the ideal case. The query parameter is fully defined in the API definition and the query parameter in the request URL matches that definition.

CASE B:

Requirement 3	**/req/core/query-param-implied**

IF the API definition of the query parameters does not specify explicit parameter names
AND
a request query parameter name falls within the valid range of one of the query parameter definitions
AND
that query parameter name is valid for the current state of the server
THEN
IF the request URI includes a query parameter value that is not valid for that parameter
THEN
The server SHALL return a response with the status code 400.
ELSE
The server SHALL meet the requirements specified for that parameter.

This is the case where the definition of a query parameter provides some lattitude. For example, if the API server allows querying on resource properties. In that case, the parameter definition may be a bounded range of possible values. However, all possible values may not be valid at any one time. If the query parameter name matches one of the properties currently hosted, then the request can proceed.

CASE C:

Requirement 4	**/req/core/query-param-corrected**
IF a query parameter does not meet the criteria for any of the query parameter names defined in the API definition AND the server is able to correct the error, THEN	
A	The server SHALL respond with a 307 or 308 status and the corrected URL,
B	The server SHALL validate the full URL prior to returning a 307 or 308 response,
C	The server SHALL only generate a 307 or 308 response if more serious errors are not detected.

An API implementation may choose to be forgiving of minor errors in the query parameters. The goal here is not to ignore the errors, but the inform the client of the error and instruct them on how to fix it.

Status codes 307 and 307 are specified for this purpose. Both refer the client to a new (corrected) URI for the resource.

Status code 308 indicates that the new URI is a permanent change. This is the most relevant response since the original URI will never be valid. However, Status Code 308 is defined in a separate RFC from the rest of the HTTP suite and may not be implemented by the client. Status Code 307 indicates that the new URI is a temporary change. While this staus is somewhat misleading, this

code is incorporated in the main body of HTTP standards. So it is more likely to be implemented by a client.

It is imperative that the redirect URI is correct. For that reason, subrequirements B and C assure that a redirect URI is not returned until ALL errors have been resolved.

Requirement 5	**/req/core/query-param-unknown**
IF a query parameter name does not meet the criteria for CASE A, B, or C THEN The server SHALL return a response with the status code `400`.	

If the query parameter cannot be validated or corrected, then tell the client.

8.1.4. Web Caching

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

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

8.1.5. Support for Cross-Origin Requests

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

Recommendation 3	**/rec/core/cross-origin**
A	If the server is intended to be accessed from 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)

8.1.6. Resource Encodings

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

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

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

HTML encoding recommendation:

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

JSON encoding recommndation:

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

8.1.7. Parameter Encoding

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

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

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

Capitalization

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

Requirement 6	**/req/core/query-param-capitalization**
A	Parameter names and values SHALL be case sensitive.
B	IF a parameter name or value includes a precent 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.

All parameter value strings should have the first word and any subsequent words in the name capitalized. All other letters should be lower case.

However, 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.

Recommendation 6	**/rec/core/query-param-capitalization**
A	Parameter names SHOULD be in Upper Camel Case.

Parameter Value Lists

Parameters may pass more than one value. The folowing requirements define how to encode a list of parameter values.

Requirement 7	**/req/core/query-param-list-delimiter**
A	Parameters values containing lists SHALL designate the list items using the comma (",") as a delimiter.

Requirement 8	**/req/core/query-param-list-escape**
A	Any list item values which include a space or comma SHALL escape the space or comma character using the URL encoding rules from IETF RFC 3986

Requirement 9	**/req/core/query-param-list-empty**
A	All empty entries SHALL be represented by the empty string ("").

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

Numeric and Boolean Values

Boolean values conform to the following requirement derived from OWS-Common.

Requirement 10	**/req/core/query-param-value-boolean**
A	Boolean values shall be represented by the uppercase strings "TRUE" and "FALSE", representing Boolean true and false respectively.

Integer values conform to the following requirement derived from XML Schema Part 2.

Requirement 11	**/req/core/query-param-value-integer**
A	Integer values SHALL be represented by a finite-length sequence of decimal digits with an optional leading sign. If the sign is omitted, "+" is assumed.

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 derived from XML Schema Part 2.

Requirement 12	**/req/core/query-param-value-decimal**
A	Decimal values SHALL be represented by a finite-length sequence of decimal digits separated by a period as a decimal indicator. • An optional leading sign is allowed. • If the sign is omitted, "+" 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 derived from XML Schema Part 2.

Requirement 13	**/req/core/query-param-value-double**
A	Double values SHALL be represented by a mantissa followed, optionally, by the character "E" or "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" or "e" and the following exponent are omitted, an exponent value of 0 SHALL be assumed.

Special values conform to the following requirement derived from XML Schema Part 2.

Requirement 14	/req/core/query-param-value-special
A	The special values positive and negative infinity and not-a-number SHALL be represented using the strings `INF`, `-INF` and `NaN`, respectively.

8.2. Resource Requirements

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

Table 5. Common Core Resources

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

8.2.1. API landing page

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

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

Operation

Requirement 15	/req/core/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/core/root-success.

Response

Requirement 16	/req/core/root-success
A	A successful execution of the operation SHALL be reported as a response with an HTTP status code `200`.

B	The content of that response SHALL be based upon the schema landingPage.json and include links to the following resources: • `/api` (relation type 'service-desc' or 'service-doc') • `/conformance` (relation type 'http://www.opengis.net/def/rel/ogc/1.0/conformance')

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

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

landingPage.json

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

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

Error Situations

See HTTP Status Codes for general guidance.

8.2.2. API Definition

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

Operation

Requirement 17	**/req/core/api-definition-op**
A	The server SHALL support the HTTP GET operation on all links from the landing page which have the relation type `service-desc`.
B	The server SHALL support the HTTP GET operation on all links from the landing page which have the relation type `service-doc`.
C	The responses to all HTTP GET requests issued in `A` and `B` SHALL satisfy requirement /req/core/api-definition-success.

Recommendation 7	**/rec/core/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/core/api-definition-success.

Response

Requirement 18	**/req/core/api-definition-success**
A	A successful execution of the operation SHALL be reported as a response with a HTTP status code `200`.
B	The content of that response SHALL be an API Definition document.
C	The API Definition document SHALL shall be consistent with the media type identified through HTTP content negotiation.
NOTE:	The `-f` parameter MAY be used to satisfy this requirement.

Recommendation 8	**/rec/core/api-definition-oas**

A	If the API definition document uses the OpenAPI Specification 3.0, THEN The document SHOULD conform to the OpenAPI Specification 3.0 requirements class.

Error Situations

See HTTP Status Codes for general guidance.

Service Metadata

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

Recommendation 9	**/rec/core/service-metadata**
In order to maintain commonality between OGC Web Services and OGC Web APIs, an OGC Web API should expose the Service Metadata defined in OWS-Common.	
A	The server SHOULD support the HTTP GET operation on all links from the landing page which have the relation type `service-meta`.
B	A successful execution of the operation SHOULD be reported as a response with an HTTP status code `200`.
C	The content of that response SHOULD include the Service Identification and Service Provider metadata defined in the OGC Web Services Common 2.0 Standard.
D	If the API definition document uses the OpenAPI Specification 3.0, THEN The response SHOULD follow the OpenAPI recommendations for Service Metadata.

8.2.3. Declaration of Conformance Classes

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

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

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

Operation

Requirement 19	**/req/core/conformance-op**
A	The server SHALL support the HTTP GET operation on the URI `{root}/conformance`.
B	The server SHALL support the HTTP GET operation on all links from the landing page which have the relation type `http://www.opengis.net/def/rel/ogc/1.0/conformance`.
C	The responses to all HTTP GET requests issued in `A` and `B` SHALL satisfy requirement /req/core/conformance-success.

Response

Requirement 20	**/req/core/conformance-success**
A	A successful execution of the operation SHALL be reported as a response with a HTTP status code `200`.
B	The content of that response SHALL be based upon the schema confClasses.json and list all OGC API conformance classes that the API conforms to.

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

Examples of OGC Conformance Declarations are provided in Conformance Examples.

Conformance Declaration Schema

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

Error situations

See HTTP Status Codes for general guidance.

Chapter 9. Encoding Requirements Classes

9.1. Overview

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

- HTML
- JSON

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

9.2. Requirement Class "HTML"

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

- The data is not discoverable using Web crawlers and search engines,
- The data can not be viewed directly in a browser - additional tools are required to view the data.

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

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

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

Requirement 21	**/req/html/definition**
A	Every `200`-response of an operation of the API SHALL support the media type `text/html`.

Requirement 22	**/req/html/content**

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

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

9.3. Requirement Class "JSON"

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

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

Requirement 23	**/req/json/definition**
A	`200`-responses of the server SHALL support the `application/json` media type.

Requirement 24	**/req/json/content**
A	Every `200`-response with the media type `application/json` SHALL include, or link to, a payload encoded according to the JSON Interchange Format

B	The schema of all responses with the media type `application/json` SHALL conform with the JSON Schema specified for that resource.

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

Chapter 10. OpenAPI 3.0 Requirements Class

10.1. Basic requirements

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

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

Requirement 25	**/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 26	**/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 27	**/req/oas30/oas-impl**
A	The API SHALL implement all capabilities specified in the OpenAPI definition.

10.2. Complete definition

Requirement 28	**/req/oas30/completeness**
A	The OpenAPI definition SHALL specify for each operation all HTTP Status Codes and Response Objects that the API uses in responses.
B	This includes the successful execution of an operation as well as all error situations that originate from the server.

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

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

10.3. Exceptions

Requirement 29	**/req/oas30/exceptions-codes**
A	For error situations that originate from an API server, the API definition SHALL cover all applicable HTTP Status Codes.

Example 1. An exception response object definition

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

10.4. Security

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

Security Requirements can be defined on following levels:

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

The OpenAPI specification currently supports the following security schemes:

- HTTP authentication,

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

Requirement 30	**/req/oas30/security**
A	For cases, where the operations of the API are access-controlled, the security scheme(s) and requirements SHALL be documented in the OpenAPI definition.

10.5. Service Metadata

OGC Web Services provide a set of metadata which identifies the service and provides information about the service provider. It would be useful if OGC Web APIs provide the same information. A `service-meta` link is provided on the Landing Page for this purpose.

Recommendation 11	**/rec/oas30/service-metadata**
In order to maintain commonality between OGC Web Services and OGC Web APIs, an OGC Web API should expose the Service Metadata defined in OWS-Common.	
A	A Service Metadata response SHOULD include the Service Identification and Service Provider metadata defined in the OGC Web Services Common 2.0 Standard.
B	Service Metadata SHOULD be encoded in the OpenAPI `Info` object.
C	To simplify access, the Service Metadata SHOULD be available as a separate resource from the Service Definition.
D	The JSON Schema at https://github.com/opengeospatial/oapi_common/blob/master/core/openapi/schemas/oasInfo.json MAY be used to encode the Service Metadata.

An example of an OpenAPI `Info` object populated with OWS metadata is provided in the Service Metadata Examples section.

10.6. Further Information

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

Chapter 11. 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`.

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

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

Chapter 12. Security Considerations

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

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

Annex A: Abstract Test Suite (Normative)

A.1. Introduction

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

A.2. Conformance Class Core

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

A.2.1. General Tests

HTTP

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

Query Parameters

Abstract Test 2	**/conf/core/query-param-known**
Test Purpose	Validate that a query parameter advertised through the API complies with its' advertised behavior.

Requirement	/req/core/query-param-known
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 exercise all of the advertised constraints on those values. (Example: minimum and maximum values) 2. Validate that the operation performed as expected. DONE DONE

Abstract Test 3	**/conf/core/query-param-unknown**
Test Purpose	Validate that an error is returned when a query parameter is used which has not been advertised through the API.
Requirement	/req/core/query-param-unknown
Test Method	DO FOR ALL operations advertised in the API definition 1. Execute that operation using a query parameter which is not advertised through the API definition. 2. Validate that the operation returns a reponse with the status code `400`. DONE

Abstract Test 4	**/conf/core/query-param-invalid**
Test Purpose	Validate that an error is returned when a query parameter does not comply with its' advertised behavior.
Requirement	/req/core/query-param-invalid
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 reponse with the status code `400`. DONE DONE

A.2.2. Landing Page {root}/

Abstract Test 5	**/conf/core/root-op**
Test Purpose	Validate that a landing page can be retrieved from the expected location.
Requirement	/req/core/root-op /req/core/root-success
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/core/root-success.

Abstract Test 6	**/conf/core/root-success**
Test Purpose	Validate that the landing page complies with the required structure and contents.
Requirement	/req/core/root-success
Test Method	Validate the landing page for all supported media types using the resources and tests identified in Table 6 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 6. Schema and Tests for Landing Pages

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

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

Abstract Test 7	**/conf/core/api-definition-op**
Test Purpose	Validate that the API Definition document can be retrieved from the expected location.
Requirement	/req/core/api-definition-op /req/core/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` 3. Validate the contents of the returned document using test /conf/core/api-definition-success. DONE

Abstract Test 8	**/conf/core/api-definition-success**
Test Purpose	Validate that the API Definition complies with the required structure and contents.
Requirement	/req/core/api-definition-success
Test Method	Validate the API Definition document against an appropriate schema document.

A.2.4. Conformance Path {root}/conformance

Abstract Test 9	**/conf/core/conformance-op**
Test Purpose	Validate that a Conformance Declaration can be retrieved from the expected locations.
Requirement	/req/core/conformance-op /req/core/conformance-success

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/core/conformance-success. DONE THEN 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/core/conformance-success. ENDIF

Abstract Test 10	**/conf/core/conformance-success**
Test Purpose	Validate that the Conformance Declaration response complies with the required structure and contents.
Requirement	/req/core/conformance-success
Test Method	1. Validate the response document against the schema confClasses.yaml 2. Validate that the document includes the conformance class "http://www.opengis.net/spec/ogcapi-common-1/1.0/conf/core" 3. Validate that the document list all OGC API conformance classes that the API implements.

A.3. Conformance Class JSON

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

A.3.1. JSON Definition

Abstract Test 11	**/conf/json/definition**
Test Purpose	Verify support for JSON
Requirement	/req/json/definition /req/json/content
Test Method	DO FOR EACH operation advertised for the API 1. Execute the operation specifing `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. DONE

A.3.2. JSON Content

Abstract Test 12	**/conf/json/content**
Test Purpose	Verify the content of a JSON document given an input document and schema.
Requirement	/req/json/content
Test Method	1. Validate that the document is a JSON (IETF RFC 8259) document. 2. Validate the document against the schema using a JSON Schema validator.

A.4. Conformance Class HTML

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

A.4.1. HTML Definition

Abstract Test 13	**/conf/html/definition**
Test Purpose	Verify support for HTML
Requirement	/req/html/definition /req/html/content
Test Method	DO FOR EACH operation advertised for the API 1. Execute the operation specifing `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.4.2. HTML Content

Abstract Test 14	**/conf/html/content**
Test Purpose	Verify the content of an HTML document given an input document and schema.
Requirement	/req/html/content
Test Method	1. Validate that the document is an HTML 5 document 2. Manually inspect the document against the schema.

A.5. Conformance Class OpenAPI 3.0

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

Abstract Test 15	**/conf/oas30/completeness**

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

Abstract Test 16	**/conf/oas30/exceptions-codes**
Test Purpose	Verify that the OpenAPI document fully describes potential exception codes.
Requirement	/req/oas30/exceptions-codes
Test Method	Verify that for each operation, the OpenAPI document describes all HTTP Status Codes that may be generated.

Abstract Test 17	**/conf/oas30/oas-definition-1**
Test Purpose	Verify that JSON and HTML versions of the OpenAPI document are available.
Requirement	/req/oas30/oas-definition-1
Test Method	1. Verify that an OpenAPI definition in JSON is available using the media type `application/vnd.oai.openapi+json;version=3.0` and link relation `service-desc` 2. Verify that an HTML version of the API definition is available using the media type `text/html` and link relation `service-doc`.

Abstract Test 18	**/conf/oas30/oas-definition-2**
Test Purpose	Verify that the OpenAPI document is valid JSON.
Requirement	/req/oas30/oas-definition-2
Test Method	Verify that the JSON representation conforms to the OpenAPI Specification, version 3.0.

Abstract Test 19	**/conf/oas30/oas-impl**

Test Purpose	Verify that all capabilities specified in the OpenAPI definition are implemented by the API.
Requirement	/req/oas30/oas-impl
Test Method	1. Construct 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 20	**/conf/oas30/security**
Test Purpose	Verify that any authentication protocols implemented by the API are documented in the OpenAPI document.
Requirement	/req/oas30/security
Test Method	1. Identify all authentication protocols supported by the API. 2. Validate that each authentication protocol is described in the OpenAPI document by a Security Schema Object and its' use specified by a Security Requirement Object.

Annex B: Examples (Informative)

B.1. Example Landing Pages

Example 2. JSON Landing Page

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

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

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


```
{
  "title": "Example API Landing Page",
  "description": "This is an example of an API landing page in JSON format",
  "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-doc",
      "type": "text/html",
      "title": "API definition for this endpoint as HTML",
      "href": "http://www.example.com/oapi-c/api?f=text/html"
    },
    {
      "rel": "http://www.opengis.net/def/rel/ogc/1.0/conformance",
      "type": "application/json",
      "title": "Conformance Declaration as JSON",
      "href": "http://www.example.com/oapi-c/conformance?f=application/json"
    },
    {
      "rel": "http://www.opengis.net/def/rel/ogc/1.0/conformance",
      "type": "text/html",
      "title": "Conformance Declaration as HTML",
      "href": "http://www.example.com/oapi-c/conformance?f=text/html"
    },
    {
      "rel": "http://www.opengis.net/def/rel/ogc/1.0/data",
      "type": "application/json",
      "title": "Collections Metadata as JSON",
      "href": "http://www.example.com/oapi-c/collections?f=application/json"
    },
    {
      "rel": "http://www.opengis.net/def/rel/ogc/1.0/data",
      "type": "text/html",
      "title": "Collections Metadata as HTML",
```

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

B.2. Conformance Examples

Example 3. Conformance Response

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

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

B.3. API Definition Examples

Example 4. JSON API Definition

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

This example also illustrates:

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

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

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

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

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

B.4. Service Metadata Examples

Example 5. Service Metadata

This is an example of the encoding of OWS-Common Service Metadata in an OpenAPI Info object. While not required, inclusion of this metadata provides common descriptive metadata for OGC Web Services and OGC Web APIs.

```
{
  "info" : {
    "title" : "My Web API",
    "version" : "1.0.0",
    "description" : "This example shows population of an OpenAPI Info element with
OWS-Common Service Metadata. The Service Identification and Service Provider
sections are addressed. .",
    "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"
  }
}
```

Annex C: Revision History

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

Annex D: Glossary

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

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

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

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

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

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

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

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

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

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

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

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

Annex E: Bibliography

- Fielding, Roy Thomas: **Architectural Styles and the Design of Network-based Software Architectures**. Doctoral dissertation, University of California, Irvine, 2000, https://www.ics.uci.edu/fielding/pubs/dissertation/fielding_dissertation.pdf
- IANA: **Link Relation Types**, https://www.iana.org/assignments/link-relations/link-relations.xml
- ISO/IEC 14977:1996(E) **Information technology – Syntactic metalanguage – Extended BNF**, available from ISO.
- Open Geospatial Consortium: **The Specification Model — A Standard for Modular specifications**, OGC 08-131
- **Schema.org**: http://schema.org/docs/schemas.html
- W3C: **Architecture of the World Wide Web, Volume One**, W3C Recommendation, 15 December 2004, https://www.w3.org/TR/webarch/
- W3C: **Data Catalog Vocabulary (DCAT) - Version 2**, W3C Recommendation, 04 February 2020, https://www.w3.org/TR/vocab-dcat-2/
- W3C: **Data on the Web Best Practices**, W3C Recommendation, 31 January 2017, https://www.w3.org/TR/dwbp/
- W3C/OGC: **Spatial Data on the Web Best Practices**, W3C Working Group Note, 28 September 2017, https://www.w3.org/TR/sdw-bp/
- W3C: **HTML5, W3C Recommendation**, http://www.w3.org/TR/html5/

Annex F: Backus-Naur Forms

F.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    = "!" / "$" / "&" / "'" / "(" / ")"
              / "*" / "+" / "," / ";" / "="
```

Annex G: OGC Web API Guidelines

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

#	Principle	Discussion
1	Don’t Reinvent	
2	Keep It Simple and Intuitive	
3	Use Well-Known Resource Types	
4	Construct consistent URIs	
5	Use HTTP Methods consistent with RFC 2616	
6	Put Selection Criteria behind the ‘?’	
7	Error Handling and use of HTTP Status Codes	
8	Use of HTTP Status Codes	
9	Use of HTTP Header	
10	Content Negotiation	
11	Pagination	
12	Processing Resources	
13	Support Metadata	
14	Consider your Security needs	
15	API Description	
16	Use IANA well-known identifiers	

17	Use explicit geospatial relations	
18	Support W3C Cross-Origin Resource Sharing	
19	Resource encodings	
20	Good APIs are testable from the beginning	