**i. Abstract**

The OGC has extended their suite of standards to include Resource Oriented Architectures (ROA) 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 specifies reusable building-blocks. Developers of OGC standards will use these building-blocks in the construction of OGC Web API Standards. The goal is a modular suite of coherent OGC API standards that can be adapted by a system designer to meet 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 OGC API. These resources and their access mechanisms are described in [Table 1](http://docs.opengeospatial.org/DRAFTS/19-072.html#cr-table).

<SNIP, SNIP>

**2. Scope**

The OGC API - Common standard addresses Discovery operations directed against the API itself. The standard 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 OGC 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 standards. 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.

**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](http://docs.opengeospatial.org/DRAFTS/19-072.html#webapi-definition)

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 or referenced in the Abstract Test Suite (ATS) 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 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.

**3.1. Core Requirements Class**

The requirements specifies in the Common *Core Requirements Class* provide 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)](http://docs.opengeospatial.org/DRAFTS/19-072.html#rc_core-section) **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](http://docs.opengeospatial.org/DRAFTS/19-072.html#rc_html-section)
* [JSON](http://docs.opengeospatial.org/DRAFTS/19-072.html#rc_json-section)

Neither of these encodings is mandatory. An implementer 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)](http://docs.opengeospatial.org/DRAFTS/19-072.html#rc_encoding-section) **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 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 [(Chapter 10)](http://docs.opengeospatial.org/DRAFTS/19-072.html#rc_oas30-section) **OpenAPI 3.0 Requirements Class**.

<<SNIP, SNIP>>

**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](http://docs.opengeospatial.org/DRAFTS/19-072.html#resource-definition).
2. [Resources](http://docs.opengeospatial.org/DRAFTS/19-072.html#resource-definition) are uniquely identified using [Uniform Resource Identifiers](http://docs.opengeospatial.org/DRAFTS/19-072.html#uri-definition) (URI).
3. A user manipulates a [resource](http://docs.opengeospatial.org/DRAFTS/19-072.html#resource-definition) through [representations](http://docs.opengeospatial.org/DRAFTS/19-072.html#representation-definition) of that [resource](http://docs.opengeospatial.org/DRAFTS/19-072.html#resource-definition).
4. A [representation](http://docs.opengeospatial.org/DRAFTS/19-072.html#representation-definition) is the current or intended state of a [resource](http://docs.opengeospatial.org/DRAFTS/19-072.html#resource-definition) encoded for exchange between components.
5. The format used to encode a [representation](http://docs.opengeospatial.org/DRAFTS/19-072.html#representation-definition) is negotiated between the components participating in the exchange.
6. [Representations](http://docs.opengeospatial.org/DRAFTS/19-072.html#representation-definition) 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](http://docs.opengeospatial.org/DRAFTS/19-072.html#WEBARCH) establishes the URI as the single global identification system for the Web. Therefore, URIs or [URI Templates](http://docs.opengeospatial.org/DRAFTS/19-072.html#rfc6570) 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)](http://docs.opengeospatial.org/DRAFTS/19-072.html#url-definition) 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](http://docs.opengeospatial.org/DRAFTS/19-072.html#requirement-definition), [Conformance Modules](http://docs.opengeospatial.org/DRAFTS/19-072.html#ctm-definition), and [Conformance Classes](http://docs.opengeospatial.org/DRAFTS/19-072.html#ctc-definition) that appear in this document are denoted by partial URIs that are relative to this base.

[Resources](http://docs.opengeospatial.org/DRAFTS/19-072.html#resource-definition) 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](https://github.com/OAI/OpenAPI-Specification/blob/master/versions/3.0.2.md#server-object). In this document the tag {root} designates the root node of a URI.

The partial URIs used to identify [Resources](http://docs.opengeospatial.org/DRAFTS/19-072.html#resource-definition) 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 standard defines [Resources](http://docs.opengeospatial.org/DRAFTS/19-072.html#resource-definition) which may appear in more than one place in the API. These [Resource Types](http://docs.opengeospatial.org/DRAFTS/19-072.html#resource-type-definition) are identified by name rather than by URI.

**Summary for Developers:**

[RFC 3986](http://docs.opengeospatial.org/DRAFTS/19-072.html#rfc3986) defines a URI in Backus-Naur Form [(BNF)](http://docs.opengeospatial.org/DRAFTS/19-072.html#bnf-citation) 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 categorized 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](http://docs.opengeospatial.org/DRAFTS/19-072.html#rc2616) and [URI Syntax](http://docs.opengeospatial.org/DRAFTS/19-072.html#rc3986) IETF RFCs. If URIs includ reserved characters that are delimiters in the URI subcomponent, these have to be percent-encoded. See Clause 2 of [RFC 3986](http://docs.opengeospatial.org/DRAFTS/19-072.html#rfc3986) for details.

|  |  |
| --- | --- |
| Note | OGC Web API standards may include a community-defined identifier as part of a URI (ex. 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](http://docs.opengeospatial.org/DRAFTS/19-072.html#rfc3986)) in the URIs for all hosted resources. |

Additional information on this topic is provided in the [OGC API - Common Users Guide](http://docs.opengeospatial.org/DRAFTS/20-071.html#identifiers-section).

**6.3. Links**

OGC Web API Standards use [RFC 8288 (Web Linking)](http://docs.opengeospatial.org/DRAFTS/19-072.html#rfc8288) 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](https://github.com/opengeospatial/oapi_common/blob/master/core/openapi/schemas/link.json).

{

"$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",

"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"

}

}

}

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](http://docs.opengeospatial.org/DRAFTS/19-072.html#rfc8288). |
| 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. |

**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](https://github.com/opengeospatial/oapi_common/blob/master/core/openapi/schemas/link.json).

The "rel" property is populated using values from the [IANA Link Relations Registry](https://www.iana.org/assignments/link-relations/link-relations.xhtml) wherever possible. Additional values are registered with the [OGC Link Relation Registry](https://github.com/opengeospatial/NamingAuthority/blob/master/registers/linkrelations.csv). Additional relation type values can be used if neither of these registers suffice.

The link relationships used in API-Common Core are described in [Table 2](http://docs.opengeospatial.org/DRAFTS/19-072.html#link-relations-table). Additional relation types may be used if the implementation warrants it.

| 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). |
| <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. |
| <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 which 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](http://docs.opengeospatial.org/DRAFTS/20-071.html#link-relations-section).

**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](http://docs.opengeospatial.org/DRAFTS/19-072.html#rfc2818) and not [HTTP](http://docs.opengeospatial.org/DRAFTS/19-072.html#rfc2616).

OGC Web API standards do not prohibit the use of any valid HTTP option. However, implementers should be aware that optional capabilities which 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 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 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. 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](http://docs.opengeospatial.org/DRAFTS/19-072.html#rc_oas30-section) for OGC API definitions that follow the [OpenAPI specification 3.0](http://docs.opengeospatial.org/DRAFTS/19-072.html#openapi). 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](http://docs.opengeospatial.org/DRAFTS/19-072.html#openapi), 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>.

**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 implementing one or more OGC API modules (conformance classes). These resources and their access mechanisms are described in [Table 3](http://docs.opengeospatial.org/DRAFTS/19-072.html#cr-table-2).

| Table 3. Common Core Resources | | | |
| --- | --- | --- | --- |
| **Resource** | **URI** | **HTTP Method** | **Document Reference** |
| Landing page | / | GET | [API Landing Page](http://docs.opengeospatial.org/DRAFTS/19-072.html#landing-page) |
| API definition | /api | GET | [API Definition](http://docs.opengeospatial.org/DRAFTS/19-072.html#api-definition) |
| Conformance declaration | /conformance | GET | [Declaration of Conformance Classes](http://docs.opengeospatial.org/DRAFTS/19-072.html#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 and early 2000s. However, technology has evolved. New Resource-Oriented APIs provide an alternative to Service-Oriented Web Services. 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](http://docs.opengeospatial.org/DRAFTS/19-072.html#SDWBP) as well as the [W3C best practices for sharing Data on the Web](http://docs.opengeospatial.org/DRAFTS/19-072.html#DWBP).

**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 using OGC API modules. 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](http://docs.opengeospatial.org/DRAFTS/20-071.html#modular-api).

**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 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.

**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](http://docs.opengeospatial.org/DRAFTS/19-072.html#general-requirements-section) are those requirements which are applicable regardless of the resource being accessed. [Resource requirements](http://docs.opengeospatial.org/DRAFTS/19-072.html#resource-requirements-section) are the requirements which define the core resources and their applicable constraints.

**8.1. General Requirements**

The following requirements and recommendations 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](http://docs.opengeospatial.org/DRAFTS/19-072.html#rfc2616). |
| B | If the API supports HTTPS, then the API SHALL also conform to [HTTP over TLS](http://docs.opengeospatial.org/DRAFTS/19-072.html#rfc2818). |

**8.1.2. HTTP Status Codes**

[Table 4](http://docs.opengeospatial.org/DRAFTS/19-072.html#status-codes) 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](http://docs.opengeospatial.org/DRAFTS/19-072.html#web-caching) 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 4](http://docs.opengeospatial.org/DRAFTS/19-072.html#status-codes) do not cover all possible conditions.

|  |  |
| --- | --- |
| **Permission 1** | **/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](http://docs.opengeospatial.org/DRAFTS/19-072.html#status-codes). |

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](http://docs.opengeospatial.org/DRAFTS/19-072.html#rfc7807) addresses this need by providing "Problem Details" response schemas for both JSON and XML.

|  |  |
| --- | --- |
| **Recommendation 2** | **/rec/core/problem-details** |
| An OGC Web API should include a "Problem Details" report in any error response in accordance with [IETF RFC 7807](http://docs.opengeospatial.org/DRAFTS/19-072.html#rfc7807). | |

**8.1.3. Query parameters**

|  |  |
| --- | --- |
| **Requirement 2** | **/req/core/query-param-unknown** |
| A | The server SHALL respond with 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-specified does not apply, and 3. /per/core/query-param-tolerance does not apply. |

|  |  |
| --- | --- |
| **Requirement 3** | **/req/core/query-param-invalid** |
| A | The server SHALL respond with a response with the status code 400, if the request URI includes a query parameter that has an invalid value. |

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 it 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.

|  |  |
| --- | --- |
| **Permission 2** | **/per/core/query-param-specified** |
| A | The specification of a query parameter in the API definition MAY encompass a range of parameter names. Any query parameter which falls within the specified range can be considered "specified" in the API definition.  Examples of a parameter range include:   * A regular expression which defines the valid parameter names, * A URL Template segment which defines the valid parameter names, * An indication that all parameter names are accepted (no parameter validation). |
| B | The API definition language chosen may not be capable of expressing the desired range of values. In that case the server SHOULD provide:   * A definition of the parameter range which best expresses the intended use of that parameter, * Additional human readable text documenting the actual range of validity. |

|  |  |
| --- | --- |
| **Permission 3** | **/per/core/query-param-tolerance** |
| A | Servers MAY display tolerance for requests with incorrect query parameters. 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. |

**8.1.4. 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)](http://docs.opengeospatial.org/DRAFTS/19-072.html" \l "rfc7232).

|  |  |
| --- | --- |
| **Recommendation 3** | **/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 4** | **/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)](https://en.wikipedia.org/wiki/Cross-origin_resource_sharing)
* [JSONP (JSON with padding)](https://en.wikipedia.org/wiki/JSONP)

**8.1.6. 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 5** | **/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](https://www.w3.org/TR/json-ld/#string-internationalization) should be used.

The [link object](http://schemas.opengis.net/ogcapi/common/part1/1.0/openapi/schemas/link.yaml) based on [RFC 8288 (Web Linking)](http://docs.opengeospatial.org/DRAFTS/19-072.html#rfc8288) 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](http://docs.opengeospatial.org/DRAFTS/19-072.html#resource-encoding-section), 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.

**8.1.7. Resource Encodings**

A Web API provides access to [resources](http://docs.opengeospatial.org/DRAFTS/19-072.html#resource-definition) through [representations](http://docs.opengeospatial.org/DRAFTS/19-072.html#representation-definition) 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](http://docs.opengeospatial.org/DRAFTS/19-072.html#rfc7231).

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 6** | **/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 7** | **/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](http://docs.opengeospatial.org/DRAFTS/19-072.html#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](http://docs.opengeospatial.org/DRAFTS/19-072.html#mediatypes-section) includes guidance on media types for [encodings](http://docs.opengeospatial.org/DRAFTS/19-072.html#rc_encoding-section) 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 how this is supported by the server.

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

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.

**8.1.8. 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 Uniform Resource Identifier (URI). The URI syntax is defined in [IETF RFC 3986](http://docs.opengeospatial.org/DRAFTS/19-072.html#rfc3986). Rules for building URI Templates can be found in [IETF RFC 6570](http://docs.opengeospatial.org/DRAFTS/19-072.html#rfc6570).

The Backus-Naur Form (BNF) definition of a URI is provided in [Annex F](http://docs.opengeospatial.org/DRAFTS/19-072.html#uri-bnf-annex).

**Capitalization**

[IETF RFC 3986](http://docs.opengeospatial.org/DRAFTS/19-072.html#rfc3986) sections 6.2.2.1 and 2.1 provide the requirements for capitalization in URIs.

|  |  |
| --- | --- |
| **Requirement 4** | **/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 8** | **/rec/core/query-param-capitalization** |
| A | Query parameter names SHOULD be in Kebab case. (lower case with dash "-" delimiters) |
| 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.

**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 uses.

|  |  |
| --- | --- |
| **Requirement 5** | **/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** | **/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](http://docs.opengeospatial.org/DRAFTS/19-072.html#rfc3986) |

|  |  |
| --- | --- |
| **Requirement 7** | **/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**

Geospatial technology is a mathematical discipline. The 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 URL. These rules are based on the computer science basic data types identified by [Kernighan and Ritchie](http://docs.opengeospatial.org/DRAFTS/19-072.html#k-and-r-1978).

Boolean values conform to the following requirement.

|  |  |
| --- | --- |
| **Requirement 8** | **/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** | **/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 is 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** | **/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 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** | **/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 conform to the following requirement.

|  |  |
| --- | --- |
| **Requirement 12** | **/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](http://docs.opengeospatial.org/DRAFTS/19-072.html#core-resources-table). The requirements and recommendations applicable to these resources are provided in the sections below.

| Table 5. Common Core Resources | |
| --- | --- |
| **URI Path** | **Description** |
| "/" | the [landing page](http://docs.opengeospatial.org/DRAFTS/19-072.html#landing-page) |
| "/api" | the [API Definition](http://docs.opengeospatial.org/DRAFTS/19-072.html#api-definition) document for this API |
| "/conformance" | the [conformance](http://docs.opengeospatial.org/DRAFTS/19-072.html#conformance-classes) 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.

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 which 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](http://docs.opengeospatial.org/DRAFTS/19-072.html#json-landing-page) for an example of the use of markup in the attribution element.

**Operation**

|  |  |
| --- | --- |
| **Requirement 13** | **/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](http://docs.opengeospatial.org/DRAFTS/19-072.html#req_core_root_success). |

**Response**

|  |  |
| --- | --- |
| **Requirement 14** | **/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](https://github.com/opengeospatial/oapi_common/blob/master/core/openapi/schemas/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') |

The landing page returned by this operation is based on the following [JSON schema](https://github.com/opengeospatial/oapi_common/blob/master/core/openapi/schemas/landingPage.json).

landingPage.json

{

"$schema": "http://json-schema.org/draft-07/schema#",

"title": "Landing Page Schema",

"description": "JSON schema for the OGC API - Common landing page",

"type": "object",

"required": [

"links"

],

"properties": {

"title": {

"title": "The title of the API.",

"description": "While a title is not required, implementors are strongly advised to include one.",

"type": "string"

},

"description": {

"description": "A textual description of the API",

"type": "string"

},

"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": {

"description": "Links to the resources exposed through this API.",

"type": "array",

"items": {"$href": "link.json"}

}

},

"additionalProperties": true

}

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

|  |  |
| --- | --- |
| **Recommendation 9** | **/rec/core/service-metadata** |
| A Web API SHOULD provide service metadata. | |
| A | A Web API service SHOULD provide one or more service metadata resources accessible by an HTTP GET operation. |
| B | The landing page for a Web API service SHOULD provide links to the service metadata resources using the relation type service-meta. |
| C | A successful execution of the operation SHOULD be reported as a response with an HTTP status code 200. |

Additional information about Service Metadata can be found in the [OAPI-Common Users Guide](http://docs.opengeospatial.org/DRAFTS/20-071.html#service-metadata-section).

Examples of OGC landing pages are provided in [Example Landing Pages](http://docs.opengeospatial.org/DRAFTS/19-072.html#landing-page-examples).

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

**Error Situations**

See [HTTP Status Codes](http://docs.opengeospatial.org/DRAFTS/19-072.html#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 15** | **/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](http://docs.opengeospatial.org/DRAFTS/19-072.html#req_core_api_definition_success). |

|  |  |
| --- | --- |
| **Recommendation 10** | **/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](http://docs.opengeospatial.org/DRAFTS/19-072.html#req_core_api-definition-success). |

**Response**

|  |  |
| --- | --- |
| **Requirement 16** | **/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 be consistent with the media type identified through HTTP content negotiation. |
| NOTE: | The -f parameter MAY be used to satisfy this requirement. |

|  |  |
| --- | --- |
| **Recommendation 11** | **/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](http://docs.opengeospatial.org/DRAFTS/19-072.html#rc_oas30-section). |

**Error Situations**

See [HTTP Status Codes](http://docs.opengeospatial.org/DRAFTS/19-072.html#http-status-codes) for general guidance.

**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 implementers 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 which 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.

**Operation**

|  |  |
| --- | --- |
| **Requirement 17** | **/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](http://docs.opengeospatial.org/DRAFTS/19-072.html#req_core_conformance_success). |

**Response**

|  |  |
| --- | --- |
| **Requirement 18** | **/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](https://github.com/opengeospatial/oapi_common/blob/master/core/openapi/schemas/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](https://github.com/opengeospatial/oapi_common/blob/master/core/openapi/schemas/confClasses.json).

Examples of OGC Conformance Declarations are provided in [Conformance Examples](http://docs.opengeospatial.org/DRAFTS/19-072.html#conformance-response-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/ogcapi-common-1/1.0/conf/core"

}

}

}

}

**Error situations**

See [HTTP Status Codes](http://docs.opengeospatial.org/DRAFTS/19-072.html#http-status-codes) for general guidance.

**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 for spatial data on the web applications:

* [HTML](http://docs.opengeospatial.org/DRAFTS/19-072.html#rc_html-section)
* [JSON](http://docs.opengeospatial.org/DRAFTS/19-072.html#rc_json-section)

Neither of these encodings are mandatory. An implementation of the [Core](http://docs.opengeospatial.org/DRAFTS/19-072.html#rc_core-section) requirements class may implement one, both, or none of them. Other encodings are possible.

**9.2. 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](http://docs.opengeospatial.org/DRAFTS/19-072.html#SDWBP). Therefore, the OGC API - Common Standard [recommends](http://docs.opengeospatial.org/DRAFTS/19-072.html#rec_html) 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"](http://docs.opengeospatial.org/DRAFTS/19-072.html#rc_core) |
| Dependency | [HTML5](http://docs.opengeospatial.org/DRAFTS/19-072.html#html5) |
| Dependency | [Schema.org](http://docs.opengeospatial.org/DRAFTS/19-072.html#schema_org) |

|  |  |
| --- | --- |
| **Requirement 19** | **/req/html/definition** |
| A | Every 200-response of an operation of the API SHALL support the media type text/html. |

|  |  |
| --- | --- |
| **Requirement 20** | **/req/html/content** |
| A | Every 200-response of the API with the media type "text/html" SHALL be a [HTML 5 document](https://www.w3.org/TR/html5/) that includes the following information in the HTML body:   * All information identified in the schemas of the [Response Object](https://github.com/OAI/OpenAPI-Specification/blob/master/versions/3.0.2.md#responseObject) in the HTML <body/>, and * All links in HTML <a/> elements in the HTML <body/>. |

|  |  |
| --- | --- |
| **Recommendation 12** | **/rec/html/schema-org** |
| A | A 200-response with the media type text/html, SHOULD include [Schema.org](http://docs.opengeospatial.org/DRAFTS/19-072.html#schema_org) annotations. |

**9.3. Requirement Class "JSON"**

JSON is a lightweight data-interchange format designed to facilitate structured data interchange between applications. JSON commonly used for Web-based software-to-software interchanges. Most Web developers are comfortable with using a JSON-based format. Therefore 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"](http://docs.opengeospatial.org/DRAFTS/19-072.html#rc_core) |
| Dependency | [IETF RFC 8259: The JavaScript Object Notation (JSON) Data Interchange Format](http://docs.opengeospatial.org/DRAFTS/19-072.html#rfc8259) |
| Dependency | [JSON Schema](http://docs.opengeospatial.org/DRAFTS/19-072.html#jschema) |

|  |  |
| --- | --- |
| **Requirement 21** | **/req/json/definition** |
| A | 200-responses of the server SHALL support the application/json media type. |

|  |  |
| --- | --- |
| **Requirement 22** | **/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](https://tools.ietf.org/html/rfc8259) |
| B | The schema of all responses with the media type application/json SHALL conform with the JSON Schema specified for that resource. |

|  |  |
| --- | --- |
| **Recommendation 13** | **/rec/json/problem-details** |
| 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](https://github.com/opengeospatial/ogcapi-common/blob/master/core/openapi/schemas/exception.json). | |

An example JSON Schema for the landing page is available at [landingPage.json](https://github.com/opengeospatial/oapi_common/blob/master/core/openapi/schemas/landingPage.json).

An example JSON Problem Details report is available at [ExceptionExample.json](https://github.com/opengeospatial/oapi_common/blob/master/core/openapi/examples/ExceptionExample.json).

**10. OpenAPI 3.0 Requirements Class**

**10.1. Basic requirements**

APIs conforming to this requirements class are self-documenting using an [OpenAPI Document](https://github.com/OAI/OpenAPI-Specification/blob/master/versions/3.0.0.md#oasDocument).

|  |  |
| --- | --- |
| **Requirements Class** | |
| <http://www.opengis.net/spec/ogcapi-common-1/1.0/req/oas30> | |
| Target type | Web API |
| Dependency | [Requirements Class "OAPI Core"](http://docs.opengeospatial.org/DRAFTS/19-072.html#rc_core) |
| Dependency | [OpenAPI Specification 3.0.2](http://docs.opengeospatial.org/DRAFTS/19-072.html#openapi) |

|  |  |
| --- | --- |
| **Requirement 23** | **/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 24** | **/req/oas30/oas-definition-2** |
| A | The JSON representation SHALL conform to the [OpenAPI Specification, version 3.0](http://docs.opengeospatial.org/DRAFTS/19-072.html#openapi). |

Two example OpenAPI documents are included in [Annex B](http://docs.opengeospatial.org/DRAFTS/19-072.html#oas-examples).

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

**10.2. Complete definition**

|  |  |
| --- | --- |
| **Requirement 26** | **/req/oas30/completeness** |
| A | The OpenAPI definition SHALL specify for each operation all [HTTP Status Codes](https://github.com/OAI/OpenAPI-Specification/blob/master/versions/3.0.0.md#httpCodes) and [Response Objects](https://github.com/OAI/OpenAPI-Specification/blob/master/versions/3.0.0.md#responseObject) 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](http://docs.opengeospatial.org/DRAFTS/19-072.html#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](http://docs.opengeospatial.org/DRAFTS/19-072.html#http-status-codes).

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.

**10.3. Exceptions**

|  |  |
| --- | --- |
| **Requirement 27** | **/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 overridden.
* Operation - only applicable to this operation. Overrides any requirements defined at the Root level.

The OpenAPI specification currently supports the following [security schemes](https://github.com/OAI/OpenAPI-Specification/blob/master/versions/3.0.0.md#security-scheme-object):

* 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 28** | **/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. |

**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 14** | **/rec/oas30/service-metadata** |
| An OGC Web API SHOULD expose service metadata. | |
| A | That Service Metadata SHOULD provide identifying metadata about both the service and the provider of that service. |
| 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. |

An example of a populated OpenAPI Info object is provided in the [Service Metadata Examples](http://docs.opengeospatial.org/DRAFTS/19-072.html#service-metadata-examples) section.

**10.6. Query Parameter Definition**

OpenAPI 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.

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.

OpenAPI 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-specified](http://docs.opengeospatial.org/DRAFTS/19-072.html#per_core-query-param-specified) and [/per/core/query-param-tolerance](http://docs.opengeospatial.org/DRAFTS/19-072.html#per_core-query-param-tolerance).

OpenAPI schema for additional "free-form" query parameters

in: query

name: freeFormParameters

schema:

type: object

additionalProperties: true

style: form

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

{

"my\_first\_parameter": "some value",

"my\_other\_parameter": 42

}

In the request URI this would be expressed as &my\_first\_parameter=some%20value&my\_other\_parameter=42.

**10.7. Further Information**

Additional guidance on using OpenAPI in OGC Web API implementations can be found in the [OAPI-Common Users Guide](http://docs.opengeospatial.org/DRAFTS/20-071.html#oas30-usage-section).

**11. Media Types**

**11.1. 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.

**11.2. OpenAPI Media Types**

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. |

**11.3. Problem Details Media Types**

OGC API-Common recommends that implementers use [IETF RFC 7807](http://docs.opengeospatial.org/DRAFTS/19-072.html#rfc7807) 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