Requirement Class "Core"

1. Requirement Class "Core"

Requirements Class	
http://www.opengis.net/spec/OAPI_Common/1.0/req/core	
Target type	Web API
Dependency	RFC 2616 (HTTP/1.1)
Dependency	RFC 2818 (HTTP over TLS)
Dependency	RFC 3339 (Date and Time on the Internet: Timestamps)
Dependency	RFC 5988 (Web Linking)

1.1. Overview

Resources

An OGC API provides a lighweight interface to access one or more resources. The resources addressed by OGC APIs fall into three categories; Foundation Resources, Geospatial Resources, and Information Resources.

Foundation Resources are those resources which are common across all OGC APIs. Those resources are defined in this OGC API-Common standard. Other OGC API standards re-use these resources and, where necessary, extend them to address unique requirements.

Geospatial Resources are the resource which we usually think of as Geospatial Data. They include Features, Coverages, Maps, and Styles. This Standard defines basic patterns for accessing Geospatial Resources. Additional OGC API Standards have been developed to address specific API requirements for each Geospatial Resource type.

Information Resources are non-geospatial resources which support the operation of the API or the access and use of the Geospatial Resources.

Modular APIs

A goal of OGC API standards is to provide rapid and easy access to geospatial resources. To meet this goal, the needs of both the resource provider and the resource consumer must be considered. Our approach 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 likelyhood that they will integrate at run-time is greatly enhanced.

Navigation

OGC APIs are designed to support two access patterns; Hypermedia Access, and Direct Access. OGC APIs support both access patterns through the use of API Definition documents, standardized paths, and standardized hypermedia schemas.

Hypermedia Access

Hypermedia Access is the use of hypermedia links to navigate from one resource to another. This pattern is typical of the Web Browser environment. A resource consumer (typically a human) starts from a landing page, selects a link on that page, then moves on to the referenced resource.

Navigation of hyperlinks is facilitated if the hyperlink includes information about the resource type at the link destination. Therefore, OGC APIs will use a set of common link relationships.

Requirement 1	**/req/core/link-relations**
A	Links used in OGC APIs SHALL use the following link relations: • `alternate`: links to this resource in another media type (the media type is specified in the `type` link attribute) • `conformance`: links to conformance information • `data`: links to an information resource

	• `describedBy`: links to external resources which further describe the subject resource • `items`: links to each individual resource which is included in a collection resource • `self`: links to this resource, • `service-desc`: links to the API Definition • `service-doc`: an alternative to `service-desc`
B	This requirement does not constrain the use of other link relations for resource types not addressed by this requirement.

OGC API hyperlinks are defined using the following Hyperlink Schema[1].

Hyperlink Schema.

```
{
    "$schema": "http://json-schema.org/draft-07/schema#",
    "title": "Link Schema",
    "description": "Schema for external references",
    "type": "object",
    "required": [
        "href"
        ],
    "properties": {
        "href": {
            "type": "string"
            },
        "rel": {
            "type": "string"
            },
        "hreflang": {
            "type": "string"
            },
        "title": {
            "type": "string"
            }
        }
    }
```

[1] https://raw.githubusercontent.com/opengeospatial/OAPI_Common/master/openapi/schemas/link.json

Direct Access

Direct Access requires that the resource consumer possesses knowlege of the path to the resouce prior to attempting access. Typically this knowlege comes from the use of standard paths, receiving the path from another entity, or by processing an API definition resource. Direct access is particularly applicable to software analytics where there is no human in the loop.

Direct access is facilitated by the use of standard URL paths.

Foundation Resources

The standard paths defined in this Standard for Foundation Resources are:

1. "/" - the landing page
2. "/api" - the API Definition document for this API
3. "/conformance" - the conformance information for this API

Recommendation 1	**/rec/core/path-conventions**
A	Implementations of OGC API standards should comply with OGC conventions on URL paths as follows: • Landing Page = / • API Definition = /api • Conformance = /conformance • Collections = /collections

Geospatial Resources

Geospatial Resources can be exposed using the path template.

```
/collections/{collectionId}/items
      where
{collectionId} = a unique identifier for a resource collection (dataset)
```

Table 1. Path Behaviors

Path	Returns
"/collections"	Metadata describing the collections avalable from this API.
"/collections/{collectionId}	Metadata describing the collection with the unique identifier {collectionId}
"/collections/{collectionId}/items"	The collection identified by the {collectionId} parameter.

Information Resources

Information Resources can exposed using two path templates:

- /collections/{collectionId}/{resourceType}
- /{resourceType}

Where

{collectionId} = a unique identifier for a Geospatial Resource collection.

{resourceType} = a text string identifying the Information Resource type.

Information Resources associated with a specific collection should be accessed through the `/collections` path. Those which are not associated with a specific collection should use the `/{resourceType}` template.

The OGC API Information Resource types and their corresponding API standard are provided in Table 2

Table 2. Information Resource Types

Resource Type	API Standard
TBD	TBD

1.2. Foundation Resources

Foundation resources are those resources which are provided by every OGC API.

API landing page

Each OGC API has a single `LandingPage` (path `/`).

The purpose of the landing page is to provide users with the basic information they need to use this API as well as links to the resources exposed through the API.

Operation

Requirement 2	**/req/core/root-op**
A	The server SHALL support the HTTP GET operation at the path `/`.

Response

Requirement 3	**/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[2] and include links to the following resources: • the API definition (relation type 'service-desc' or 'service-doc') • `/conformance` (relation type 'conformance') • one or more information resources (relation type 'data')

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

[2] https://raw.githubusercontent.com/opengeospatial/oapi_common/master/OAPI-Common/openapi/schemas/landingPage.json

The landing page returned by this operation is based on the following Landing Page Schema[3]. Examples of OGC landing pages are provided in ???.

Landing Page Schema.

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

Error Situations

See the section called “HTTP Status Codes” for general guidance.

[3] https://raw.githubusercontent.com/opengeospatial/OAPI_Common/master/openapi/schemas/landingPage.json

API Definition

Every API is expected to provide a definition that describes capabilities provided by the API. This document 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 4	**/req/core/api-definition-op**
A	The URIs of all API definitions referenced from the landing page SHALL support the HTTP GET method.

Response

Requirement 5	**/req/core/api-definition-success**
A	A GET request to the URI of an API definition linked from the landing page (link relations `service-desc` or `service-doc`) with an `Accept` header with the value of the link property `type` SHALL return a document consistent with the requested media type.

Recommendation 2	**/rec/core/api-definition-oas**
A	If the API definition document uses the OpenAPI Specification 3.0, the document SHOULD conform to the OpenAPI Specification 3.0 requirements class.

If multiple API definition formats are supported, use content negotiation to select the desired representation.

Error Situations

See the section called “HTTP Status Codes” for general guidance.

Declaration of Conformance Classes

To support "generic" clients that want to accessing OGC APIs in general - and not "just" a specific API / server, the API has to declare the requirements classes it implements and conforms to.

Operation

Requirement 6	/req/core/conformance-op
A	The API SHALL support the HTTP GET operation at the path `/conformance`.

Response

Requirement 7	/req/core/conformance-success
A	A successful execution of the operation SHALL be reported as a response with a HTTP status code `200`.
B	The content of that response SHALL be based upon the OpenAPI 3.0 schema confClasses.yaml[4] and list all OGC API conformance classes that the API conforms to.

The conformance resource returned by this operation is based on the following Conformance Schema[5]. Examples of OGC conformance resources are provided in ???.

Conformance Schema.

```
{
"$schema": "http://json-schema.org/draft-07/schema#",
"title": "Conformance Classes Schema",
"description": "This schema defines the resource returned from the /
Conformance path",
```

[4] https://raw.githubusercontent.com/opengeospatial/oapi_common/master/OAPI-Common/openapi/schemas/confClasses.yaml

[5] https://raw.githubusercontent.com/opengeospatial/OAPI_Common/master/core/openapi/schemas/confClasses.json

```
"type": "object",
"required": [
    "conformsTo"
    ],
"properties": {
    "conformsTo": {
        "type": "array",
        "description": "ConformsTo is an array of URLs.  Each URL should
 correspond to a defined OGC Conformance class.  Unrecognized URLs
 should be ignored",
        "items": {
            "type": "string",
            "example": "http://www.opengis.net/spec/OAPI_Common/1.0/req/
core"
            }
        }
    }
}
```

Error situations

See the section called “HTTP Status Codes” for general guidance.

1.3. Geospatial Resources

Geospatial Resources are the resource types which we usually think of as Geospatial Data. They include Features, Coverages, Maps, and Styles.

Detailed requirements for each Geospatial Resource type are dealt with in the resource-specific API standards. However, this API Common standard has the responsiblity to see that they all OGC API standards work together by:

1. Providing specifications for the description of each collection (/collections/{collectionId}), and the list of collections (/collections)
2. Providing a consistent framework for serving geospatial data from the OGC API, regardless of the type. Consistent means that #1 works exactly the same (potentially with type-specific additional properties) and that the different types of data can all be collections on the same OGC API end-point.
3. Providing a tie point for other OGC API modules to connect to and reference (processes inputs & outputs, cataloging, searching and filtering collections, detailed metadata, tiles, styles, clipping and intersecting bounding boxes in

common) Just by virtue of understanding that /collections/{collectionId} points to a geospatial(--dataset--) **data layer**.

Collections Metadata

OGC APIs typically organize their Geospatial Resources into collections. These collections are accessed through the /collections path.

Operation

Requirement 8	**/req/core/rc-md-op**
A	The API SHALL support the HTTP GET operation at the path `/collections`.

Response

Requirement 9	**/req/core/rc-md-success**
A	A successful execution of the operation SHALL be reported as a response with a HTTP status code `200`.
B	The content of that response SHALL be based upon the OpenAPI 3.0 schema collections.yaml[6].

The collections metadata returned by this operation is based on the Collections Metadata Schema[7]. Examples of collections metadata are provided in ???.

Collections Metadata Schema.

```
{
    "$schema": "http://json-schema.org/draft-07/schema#",
    "title": "Collections Schema",
    "description": "This schema defines metadata resource returned
 from /collections.",
```

[6] https://raw.githubusercontent.com/opengeospatial/oapi_common/master/OAPI-Common/openapi/schemas/collections.yaml

[7] https://raw.githubusercontent.com/opengeospatial/OAPI_Common/master/core/openapi/schemas/collections.yaml

```
    "type": "object",
    "required": [
        "links",
        "collections"
        ],
    "properties": {
        "links": {
            "type": "array",
            "items": {"$href": "link.json"}
            },
        "collections": {
            "type": "array",
            "items": {"$href": "collectionInfo.json"}
            }
        }
    }
```

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

Requirement 10	**/req/core/rc-md-links**
A	A `200`-response SHALL include the following links in the `links` property of the response: • a link to this response document (relation: `self`), • a link to the response document in every other media type supported by the API (relation: `alternate`).
B	All links SHALL include the `rel` and `type` link parameters.

Recomendation 3	**/rec/core/rc-md-desciptions**
A	If external schemas or descriptions for the dataset exist that provide information about the structure or semantics of the data, a `200`-response SHOULD include links to each of those resources in the `links` property of the response (relation: `describedBy`).

B	The `type` link parameter SHOULD be provided for each link. This applies to resources that describe to the whole dataset.
C	For resources that describe the contents of a resource collection, the links SHOULD be set in the `links` property of the appropriate object in the `collections` resource.

Examples for descriptions are: XML Schema, Schematron, JSON Schema, RDF Schema, OWL, SHACL, a feature catalogue, etc.

The `collections` property of the Collections Metadata is based on the schema described in the section called “Collection Information”. The follwing requirements and recommendations govern the use of Collection Information in the Collections Metadata.

Requirement 11	**/req/core/rc-md-items**
A	For each resource collection provided through this API, metadata describing that collection SHALL be provided in the property `collections`.

Permission 1	**/per/core/rc-md-items**
A	To support servers with many collections, servers MAY limit the number of items included in the `collections` property.

Requirement 12	**/req/core/rc-md-items-links**
A	For each resource collection included in the response, the `links` property of the collection SHALL include an item for each supported encoding with a link to the collection resource (relation: `items`).
B	All links SHALL include the `rel` and `type` properties.

Error situations

See the section called “HTTP Status Codes” for general guidance.

Collection Information

Each resource collection is described by a set of metadata. That metadata is accessed directly using the `/collections/{collectionId}` path or as an entry in the `collections` property of the Collections Metadata resource.

Operation

Requirement 13	**/req/core/src-md-op**
A	If the API is exposing more than one collection of the same resouce type, then the API SHALL support the HTTP GET operation at the path `/collections/{collectionId}`.
B	The parameter `collectionId` is each `id` property in the resource collections response (JSONPath: `$.collections[*].id`).

Response

Requirement 14	**/req/core/src-md-success**
A	A successful execution of the operation SHALL be reported as a response with a HTTP status code `200`.
B	The content of that response SHALL be consistent with the content for this resource collection in the `/collections` response. That is, the values for `id`, `title`, `description` and `extent` SHALL be identical.

Collection Information is based on the Collection Information Schema[8] Schema.

Collection Information Schema.

```
{
    "$schema": "http://json-schema.org/draft-07/schema#",
    "title": "Collections Schema",
    "description": "This schema defines metadata resource returned
 from /collections/{collectionId}.",
    "type": "object",
    "required": [
        "id",
        "links"
        ],
    "properties": {
        "id": {
          "type": "string"
         },
        "title": {
            "type": "string"
            },
        "description": {
            "type": "string"
            },
        "links": {
            "type": "array",
            "items": {"$href": "link.json"}
            },
        "extent": {"$href": "bbox.yaml"},
        "crs": {
            "description": "the list of coordinate reference systems
 supported by the API; the first item is the default coordinate
 reference system",
            "type": "array",
            "items": {
                "type": "string"
                },
            "default": [
                "http://www.opengis.net/def/crs/OGC/1.3/CRS84"
                ],
            "example": [
                "http://www.opengis.net/def/crs/OGC/1.3/CRS84",
```

[8] https://raw.githubusercontent.com/opengeospatial/OAPI_Common/master/core/openapi/schemas/collectionInfo.yaml

```
                "http://www.opengis.net/def/crs/EPSG/0/4326"
                ]
            }
        }
    }
```

Additional requirements and recomendations apply to the `extent` propery of the Collection Information.

Requirement 15	**/req/core/rc-md-extent**
A	For each resource collection, the `extent` property, if provided, SHALL provide bounding boxes that include all spatial geometries and time intervals that include all temporal geometries in this collection. The temporal extent may use `null` values to indicate an open time interval.
B	If a resource has multiple properties with spatial or temporal information, it is the decision of the API implementation whether only a single spatial or temporal geometry property is used to determine the extent or all relevant geometries.

Recommendation 4	**/rec/core/rc-md-extent-single**
A	While the spatial and temporal extents support multiple bounding boxes (`bbox` array) and time intervals (`interval` array) for advanced use cases, implementations SHOULD provide only a single bounding box or time interval unless the use of multiple values is important for the use of the dataset and agents using the API are known to be support multiple bounding boxes or time intervals.

Permission 2	**/per/core/rc-md-extent-extensions**
A	The Core only specifies requirements for spatial and temporal extents. However, the `extent` object

	MAY be extended with additional members to represent other extents, for example, thermal or pressure ranges.
B	The Core only supports spatial extents in WGS84 longitude/latitude andtemporal extents in the calendar (these are the only enum values in extent.yaml[9]).
C	Extensions to the Core MAY add additional reference systems to the `extent` object.

Error situations

See the section called “HTTP Status Codes” for general guidance.

If the parameter `collectionId` does not exist on the server, the status code of the response will be `404` (see Table 3).

Collection Resource

A collection resource is the content of the collection as opposed to metadata about that collection. This standard defines the general behavior of this operation, but detailed requirements are the purvue of the API standard for that resource type.

Operation

Requirement 16	**/req/core/rc-op**
A	For every resource collection identified in the resource collections response (path `/collections`), the API SHALL support the HTTP GET operation at the path `/collections/{collectionId}/items`. • The parameter `collectionId` is each `id` property in the resource collections response (JSONPath: `$.collections[*].id`).

[9] https://raw.githubusercontent.com/opengeospatial/OAPI_Common/master/core/openapi/schemas/

Response

Requirement 17	/req/core/rc-response
A	A successful execution of the operation SHALL be reported as a response with a HTTP status code `200`.
B	The response SHALL only include resources selected by the request.

Error situations

See the section called “HTTP Status Codes” for general guidance.

1.4. Information Resources

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

1.5. Parameter Modules

Query parameters are used in URLs to limit the resources which are returned on a GET request. The API Common standard defines two standard parameters for use in OGC API standards.

Parameter bbox

Requirement 18	/req/core/rc-bbox-definition

A	The `bbox` parameter SHALL possess the following characteristics (using an OpenAPI Specification 3.0 fragment):
	
	`name: bbox`
	`in: query`

```
required: false
schema:
  type: array
  minItems: 4
  maxItems: 6
  items:
    type: number
style: form
explode: false
```

Requirement 19	/req/core/rc-bbox-response
A	If the `bbox` parameter is provided, only resources that have a spatial geometry that intersects the bounding box SHALL be part of the result set.
B	If a resource has multiple spatial geometry properties, it is the decision of the server whether only a single spatial geometry property is used to determine the extent or all relevant geometries.
C	The `bbox` parameter SHALL also match all resources in the collection that are not associated with a spatial geometry.
D	The bounding box is provided as four or six numbers, depending on whether the coordinate reference system includes a vertical axis (height or depth): • Lower left corner, coordinate axis 1 • Lower left corner, coordinate axis 2 • Lower left corner, coordinate axis 3 (optional) • Upper right corner, coordinate axis 1 • Upper right corner, coordinate axis 2 • Upper right corner, coordinate axis 3 (optional)
C	The coordinate reference system of the values on axis 1 and 2 SHALL be interpreted as WGS84 longitude/latitude (http://www.opengis.net/def/crs/

	OGC/1.3/CRS84) unless a different coordinate reference system is specified in a parameter `bbox-crs`.
D	The coordinate values SHALL be within the extent specified for the coordinate reference system.

"Intersects" means that the rectangular area specified in the parameter `bbox` includes a coordinate that is part of the (spatial) geometry of the resource. This includes the boundaries of the geometries (e.g. for curves the start and end position and for surfaces the outer and inner rings).

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

For WGS84 longitude/latitude the bounding box is in most cases the sequence of minimum longitude, minimum latitude, maximum longitude and maximum latitude. However, in cases where the box spans the anti-meridian the first value (west-most box edge) is larger than the third value (east-most box edge).

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

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

A template for the definition of the parameter in YAML according to OpenAPI 3.0 is available at bbox.yaml[10].

Parameter datetime

Requirement 20	/req/core/rc-time-definition
A	The `datetime` parameter SHALL have the following characteristics (using an OpenAPI Specification 3.0 fragment):

[10] https://raw.githubusercontent.com/opengeospatial/OAPI_Common/master/core/openapi/parameters/bbox.yaml

```
name: datetime
in: query
required: false
schema:
  type: string
style: form
explode: false
```

Requirement 21	/req/core/rc-time-response
A	If the `datetime` parameter is provided, only resources that have a temporal geometry that intersects the temporal information in the `datetime` parameter SHALL be part of the result set.
B	If a resourcee has multiple temporal properties, it is the decision of the API whether only a single temporal property is used to determine the extent or all relevant temporal properties.
C	The `datetime` parameter SHALL match all resources in the collection that are not associated with a temporal geometry.

D	The temporal information is either a date-time or a time interval. The parameter value SHALL conform to the following syntax (using ABNF [11]):
	`interval-closed = date-time "/" date-time`
	`interval-open-start = "../" date-time`
	`interval-open-end = date-time "/.."`
	`interval = interval-closed / interval-open-start / interval-open-end`
	`datetime = date-time / interval`

E	The syntax of `date-time` is specified by RFC 3339, 5.6 [12] .

[11] https://tools.ietf.org/html/rfc2234
[12] https://tools.ietf.org/html/rfc3339#section-5.6

F	Open ranges in time intervals at the start or end SHALL be supported using a double-dot (`..`).

"Intersects" means that the time (instant or period) specified in the parameter `datetime` includes a timestamp that is part of the temporal geometry of the resource (again, a time instant or period). For time periods this includes the start and end time.

Example 2. A date-time

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

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

For resources with a temporal property that is a timestamp (like `lastUpdate` in the building features), a date-time value would match all resources where the temporal property is identical.

For resources with a temporal property that is a date or a time interval, a date-time value would match all resources where the timestamp is on that day or within the time interval.

Example 3. Intervals

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

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

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

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

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

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

A template for the definition of the parameter in YAML according to OpenAPI 3.0 is available at datetime.yaml [13].

[13] https://raw.githubusercontent.com/opengeospatial/OAPI_Common/master/core/openapi/parameters/datetime.yaml

1.6. General Requirements

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

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 22	/req/core/http
A	The API SHALL conform to HTTP 1.1.
B	If the API supports HTTPS, then the API SHALL also conform to HTTP over TLS.

HTTP Status Codes

Table 3 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 3. Typical HTTP status codes

Status code	Description
`200`	A successful request.
`304`	An entity tag was provided in the request and the resource has not been changed since the previous request.
`400`	The server cannot or will not process the request due to an apparent client error. For example, a query parameter had an incorrect value.
`401`	The request requires user authentication. The response includes a `WWW-Authenticate` header field containing a challenge applicable to the requested resource.
`403`	The server understood the request, but is refusing to fulfill it. While status code `401` indicates missing or bad authentication,

Status code	Description
	status code `403` indicates that authentication is not the issue, but the client is not authorised 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`	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 3	/per/core/additional-status-codes
A	Servers MAY support other capabilities of the HTTP protocol and, therefore, MAY return other status codes than those listed in Table 3, too.

Web Caching

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

Recommendation 5	/rec/core/etag
A	The service SHOULD support entity tags and the associated headers as specified by HTTP/1.1.

Support for Cross-Origin Requests

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

Recommendation 6	**/rec/core/cross-origin**
A	If the server is intended to be accessed from the browser, cross-origin requests SHOULD be supported. Note that support can also be added in a proxy layer on top of the server.

Two common mechanisms to support cross-origin requests are:

- Cross-origin resource sharing (CORS)[14]
- JSONP (JSON with padding)[15]

Encodings

While the OAPI Common standard does not specify any mandatory encoding, the following encodings are recommended. See Clause 6 (Overview) for a discussion.

Recommendation 7	**/rec/core/html**
A	To support browsing a API with a web browser and to enable search engines to crawl and index the dataset, implementations SHOULD consider to support an HTML encoding.

Recommendation 8	**/rec/core/geojson**

[14] https://en.wikipedia.org/wiki/Cross-origin_resource_sharing
[15] https://en.wikipedia.org/wiki/JSONP

A	If the resource can be represented for the intended use in GeoJSON, implementations SHOULD consider to support GeoJSON as an encoding.

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

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

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

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

Two common approaches are:

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

Link Headers

Recommendation 9	**/rec/core/link-header**
A	Links included in payload of responses SHOULD also be included as `Link` headers in the HTTP response according to RFC 8288, Clause 3.

B	This recommendation does not apply, if 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.

Coordinate Reference Systems

As discussed in Chapter 9 of the W3C/OGC Spatial Data on the Web Best Practices document, how to express and share the location of resources in a consistent way is one of the most fundamental aspects of publishing geographic data and it is important to be clear about the coordinate reference system that coordinates are in.

For the reasons discussed in the Best Practices, OGC APIs use WGS84 longitude and latitude as the default coordinate reference system.

Requirement 23	**/req/core/crs84**
A	Unless the client explicitly requests a different coordinate reference system, all spatial geometries SHALL be in the coordinate reference system http://www.opengis.net/def/crs/OGC/1.3/CRS84 (WGS 84 longitude/latitude) for geometries without height information and http://www.opengis.net/def/crs/OGC/0/CRS84h (WGS 84 longitude/latitude plus ellipsoidal height) for geometries with height information.

The implementations compliant with the Core are not required to support publishing geometries in coordinate reference systems other than http://www.opengis.net/def/crs/OGC/1.3/CRS84. The Core also does not specify a capability to request geometries in a different reference system than the native one of the published resource. Such a capability will be specified in other OGC API standards.