INTERNATIONAL STANDARD

ISO 19168-1

First edition
2020-09

Geographic information — Geospatial API for features —

Part 1: Core

Information géographique — API géospatiale pour les entités —

Partie 1: Profil minimal

Reference number
ISO 19168-1:2020(E)

ISO copyright office
CP 401 • Ch. de Blandonnet 8
CH-1214 Vernier, Geneva
Phone: +41 22 749 01 11
Email: copyright@iso.org
Website: www.iso.org

Published in Switzerland

Contents

Page

Foreword

ISO (the International Organization for Standardization) is a worldwide federation of national standards bodies (ISO member bodies). The work of preparing International Standards is normally carried out through ISO technical committees. Each member body interested in a subject for which a technical committee has been established has the right to be represented on that committee. International organizations, governmental and non-governmental, in liaison with ISO, also take part in the work. ISO collaborates closely with the International Electrotechnical Commission (IEC) on all matters of electrotechnical standardization.

The procedures used to develop this document and those intended for its further maintenance are described in the ISO/IEC Directives, Part 1. In particular, the different approval criteria needed for the different types of ISO documents should be noted. This document was drafted in accordance with the editorial rules of the ISO/IEC Directives, Part 2 (see www.iso.org/directives).

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO shall not be held responsible for identifying any or all such patent rights. Details of any patent rights identified during the development of the document will be in the Introduction and/or on the ISO list of patent declarations received (see www.iso.org/patents).

Any trade name used in this document is information given for the convenience of users and does not constitute an endorsement.

For an explanation of the voluntary nature of standards, the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the World Trade Organization (WTO) principles in the Technical Barriers to Trade (TBT), see www.iso.org/iso/foreword.html.

This document was prepared by Technical Committee ISO/TC 211, *Geographic information/Geomatics*.

A list of all parts in the ISO 19168 series can be found on the ISO website.

Any feedback or questions on this document should be directed to the user's national standards body. A complete listing of these bodies can be found at www.iso.org/members.html.

Introduction

OGC API standards[9] define modular API building blocks to spatially enable Web APIs in a consistent manner. The OpenAPI specification is used to define the API building blocks.

The OGC API family of standards is organized by resource type. This document specifies the fundamental API building blocks for interacting with features. The spatial data community uses the term 'feature' for things in the real world that are of interest.

For those not familiar with the term 'feature,' the explanations on Spatial Things, Features and Geometry in the W3C/OGC Spatial Data on the Web Best Practice document provide more detail.

OGC API Features provides API building blocks to create, modify and query features on the Web. OGC API Features is comprised of multiple parts, each of them a separate standard. This document, the "Core", specifies the core capabilities and is restricted to fetching features where geometries are represented in the coordinate reference system, WGS 84, with axis order longitude/latitude. Additional capabilities that address more advanced needs will be specified in additional parts. Examples include support for creating and modifying features, more complex data models, richer queries, additional coordinate reference systems, multiple datasets and collection hierarchies.

By default, every API implementing this document will provide access to a single dataset. Rather than sharing the data as a complete dataset, the OGC API Features standards offer direct, fine-grained access to the data at the feature (object) level.

The API building blocks specified in this document are consistent with the architecture of the Web. In particular, the API design is guided by the IETF HTTP/HTTPS RFCs, the W3C Data on the Web Best Practices, the W3C/OGC Spatial Data on the Web Best Practices and the emerging OGC Web API Guidelines. A particular example is the use of the concepts of datasets and dataset distributions as defined in DCAT and used in schema.org.

This document specifies discovery and query operations that are implemented using the HTTP GET method. Support for additional methods (in particular POST, PUT, DELETE, PATCH) will be specified in additional parts.

Discovery operations enable clients to interrogate the API, including the API definition and metadata about the feature collections provided by the API, to determine the capabilities of the API and retrieve information about available distributions of the dataset.

Query operations enable clients to retrieve features from the underlying data store based upon simple selection criteria, defined by the client.

A subset of the OGC API family of standards is expected to be published by ISO. For example, this document is published by ISO as ISO 19168-1. To reflect that only a subset of the OGC API standards will be published by ISO and to avoid using organization names in the titles of ISO standards, standards from the "OGC API" series are published by ISO as "Geospatial API," i.e. the title of this document in OGC is "OGC API — Features — Part 1: Core" and the title in ISO is "Geographic Information — Geospatial API for features — Part 1: Core."

For simplicity, this document consistently uses:

— "OGC API" to refer to the family of standards for geospatial Web APIs that in ISO is published as "Geospatial API;"

— "OGC API - Features" to refer to the multipart standard for features that in ISO is published as ISO 19168 / "Geographic Information - Geospatial API for features;"

— "OGC API - Features — Part 1: Core" to refer to this document that in ISO is published as ISO 19168-1 / "Geographic Information - Geospatial API for features — Part 1: Core."

This document defines the resources listed in Table 1. For an overview of the resources, see 7.1.

Table 1 — Overview of resources, applicable HTTP methods and links to the document sections

Resource	Path	HTTP method	Document reference
Landing page	/	GET	7.2 API landing page
Conformance declaration	/conformance	GET	7.4 Declaration of conformance classes
Feature collections	/collections	GET	7.13 Feature collections
Feature collection	/collections/{collectionId}	GET	7.14 Feature collection
Features	/collections/{collectionId}/items	GET	7.15 Features
Feature	/collections/{collectionId}/items/{featureId}	GET	7.16 Feature

Implementations of OGC API Features are intended to support two different approaches for how clients can use the API. For further information, see 6.1.

Geographic information — Geospatial API for features —

Part 1: Core

1 Scope

This document specifies the behaviour of Web APIs that provide access to features in a dataset in a manner independent of the underlying data store. This document defines discovery and query operations.

Discovery operations enable clients to interrogate the API, including the API definition and metadata about the feature collections provided by the API, to determine the capabilities of the API and retrieve information about available distributions of the dataset.

Query operations enable clients to retrieve features from the underlying data store based upon simple selection criteria, defined by the client.

2 Normative references

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

Internet Engineering Task Force (IETF), RFC 2818: **HTTP Over TLS** [online]. Edited by E. Rescorla. 2000 [viewed 2020-03-16]. Available at https://tools.ietf.org/rfc/rfc2818.txt

Internet Engineering Task Force (IETF), RFC 3339:2002: **Date and Time on the Internet: Timestamps** [online]. Edited by G. Klyne, C. Newman. 2002 [viewed 2020-03-16]. Available at https://tools.ietf.org/rfc/rfc3339.txt

Internet Engineering Task Force (IETF), RFC 7230 to RFC 7235: **HTTP/1.1** [online]. Edited by R. Fielding, J. Reschke, Y. Lafon, M. Nottingham. 2014 [viewed 2020-04-28]. Available at https://tools.ietf.org/rfc/rfc7230.txt, https://tools.ietf.org/rfc/rfc7231.txt, https://tools.ietf.org/rfc/rfc7232.txt, https://tools.ietf.org/rfc/rfc7233.txt, https://tools.ietf.org/rfc/rfc7234.txt, and https://tools.ietf.org/rfc/rfc7235.txt

Internet Engineering Task Force (IETF), RFC 8288:2017: **Web Linking** [online]. Edited by M. Nottingham. 2017 [viewed 2020-03-16]. Available at https://tools.ietf.org/rfc/rfc8288.txt

OpenAPI Initiative (OAI), **OpenAPI Specification 3.0** [online]. 2020 [viewed 2020-03-16]. The latest patch version at the time of publication of this standard was 3.0.3, available at http://spec.openapis.org/oas/v3.0.3

3 Terms, definitions and abbreviated terms

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

ISO and IEC maintain terminological databases for use in standardization at the following addresses:

— ISO Online browsing platform: available at https://www.iso.org/obp

— IEC Electropedia: available at http://www.electropedia.org/

3.1.1
dataset
collection of data

Note 1 to entry: Published or curated by a single agent, and available for access or download in one or more formats.

Note 2 to entry: The use of 'collection' in the definition from DCAT is broader than the use of the term collection in this document. See the definition of *feature collection* (3.1.4).

[SOURCE: DCAT[8], 6.6, modified — Definition split into definition and Note 1 to entry; Note 2 to entry has been added]

3.1.2
distribution
specific representation of a *dataset* (3.1.1)

EXAMPLE A downloadable file, an RSS feed or an API.

[SOURCE: DCAT[8], 6.7, modified — Definition has been shortened]

3.1.3
feature
abstraction of real-world phenomena

Note 1 to entry: The explanations on Spatial Things, Features and Geometry in the W3C/OGC Spatial Data on the Web Best Practice document[6] provide more detail.

[SOURCE: ISO 19101-1:2014, 4.1.11, modified — Note 1 to entry has been added]

3.1.4
feature collection
collection
set of *features* (3.1.3) from a *dataset* (3.1.1)

3.1.5
Web API
API using an architectural style that is founded on the technologies of the Web

Note 1 to entry: Best Practice 24: Use Web Standards as the foundation of APIs in the W3C Data on the Web Best Practices[7] provides more detail.

[SOURCE: DWBP[7], 8.10.1, modified — Rephrased for clarity]

3.1 Abbreviated terms

API Application Programming Interface

CORS Cross-Origin Resource Sharing

CRS Coordinate Reference System

HTTP Hypertext Transfer Protocol

HTTPS Hypertext Transfer Protocol Secure

IANA Internet Assigned Numbers Authority

OGC Open Geospatial Consortium

TRS Temporal Coordinate Reference System

URI Uniform Resource Identifier

YAML YAML Ain't Markup Language

4 Conformance

This document defines six requirements/conformance classes.

The standardization targets of all conformance classes are "Web APIs."

The main requirements class is:

— Core.

The Core specifies requirements that all Web APIs have to implement.

The Core does not mandate a specific encoding or format for representing features or feature collections. Four requirements classes depend on the Core and specify representations for these resources in commonly used encodings for spatial data on the web:

— HTML,

— GeoJSON,

— Geography Markup Language (GML), Simple Features Profile, Level 0, and

— Geography Markup Language (GML), Simple Features Profile, Level 2.

None of these encodings are mandatory and an implementation of the Core may also decide to implement none of them, but to implement another encoding instead.

That said, the Core requirements class includes recommendations to support, where practical, HTML and GeoJSON as encodings. Clause 6 (Overview) includes a discussion about the recommended encodings.

The Core does not mandate any encoding or format for the formal definition of the API either. One option is the OpenAPI 3.0 specification and a requirements class has been specified for OpenAPI 3.0, which depends on the Core:

— OpenAPI specification 3.0.

As with the feature encodings, an implementation of the Core requirements class may also decide to use other API definition representations in addition or instead of an OpenAPI 3.0 definition. Examples for alternative API definitions: OpenAPI 2.0 (Swagger), future versions of the OpenAPI specification, an OWS Common 2.0 capabilities document or WSDL.

The Core is intended to be a minimal useful API for fine-grained read-access to a spatial dataset where geometries are represented in the coordinate reference system WGS 84 with axis order longitude/latitude.

Additional capabilities, e.g. support for transactions, complex data structures, rich queries, other coordinate reference systems, subscription/notification, and returning aggregated results, may be specified in future parts of the OGC API Features series or as vendor-specific extensions.

Conformance with this document shall be checked using all the relevant tests specified in Annex A (normative) of this document. The framework, concepts, and methodology for testing, and the criteria to be achieved to claim conformance are specified in the OGC Compliance Testing Policies and Procedures and the OGC Compliance Testing web site. Table 2 provides conformance class URIs

Table 2 — Conformance class URIs

Conformance class	URI
Core	http://www.opengis.net/spec/ogcapi-features-1/1.0/conf/core
HTML	http://www.opengis.net/spec/ogcapi-features-1/1.0/conf/html
GeoJSON	http://www.opengis.net/spec/ogcapi-features-1/1.0/conf/geojson
GML, Simple Features Profile, Level 0	http://www.opengis.net/spec/ogcapi-features-1/1.0/conf/gmlsf0
GML, Simple Features Profile, Level 2	http://www.opengis.net/spec/ogcapi-features-1/1.0/conf/gmlsf2
OpenAPI Specification 3.0	http://www.opengis.net/spec/ogcapi-features-1/1.0/conf/oas30

5 Conventions

5.1 Identifiers

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

All requirements and conformance tests that appear in this document are denoted by partial URIs which are relative to this base.

5.2 Link relations

To express relationships between resources, RFC 8288 (Web Linking) is used.

The following registered link relation types[3] are used in this document.

— **alternate**: Refers to a substitute for this context.

— **collection**: The target IRI points to a resource which represents the collection resource for the context IRI.

— **describedby**: Refers to a resource providing information about the link's context.

— **item**: The target IRI points to a resource that is a member of the collection represented by the context IRI.

— **next**: Indicates that the link's context is a part of a series, and that the next in the series is the link target.

— **license**: Refers to a license associated with this context.

— **prev**: Indicates that the link's context is a part of a series, and that the previous in the series is the link target.

  — This relation is only used in examples.

— **self**: Conveys an identifier for the link's context.

— **service-desc**: Identifies service description for the context that is primarily intended for consumption by machines.

  — API definitions are considered service descriptions.

— **service-doc**: Identifies service documentation for the context that is primarily intended for human consumption.

In addition, the following link relation types are used for which no applicable registered link relation type could be identified.

— **items**: Refers to a resource that is comprised of members of the collection represented by the link's context.

— **conformance**: Refers to a resource that identifies the specifications that the link's context conforms to.

— **data**: Refers to the root resource of a dataset in an API.

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

5.3 Use of HTTPS

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

5.4 HTTP URIs

This document does not restrict the lexical space of URIs used in the API beyond the requirements of the HTTP and URI Syntax IETF RFCs. If URIs include reserved characters that are delimiters in the URI subcomponent, these have to be percent-encoded. See RFC 3986:2005, Clause 2[2] for details.

5.5 API definition

5.5.1 General remarks

Good documentation is essential for every API so that developers can more easily learn how to use the API. In the best case, documentation will be available in HTML and in a format that can be processed by software to connect to the API.

This document specifies requirements and recommendations for APIs that share feature data and that want to follow a standard way of doing so. In general, APIs will go beyond the requirements and recommendations stated in this document, or other parts of the OGC API series of standards, and will support additional operations, parameters, etc. that are specific to the API or the software tool used to implement the API.

5.5.2 Role of OpenAPI

This document uses OpenAPI 3.0 fragments as examples and to formally state requirements. However, using OpenAPI 3.0 is not required for implementing a server.

Therefore, the Core requirements class only requires that an API definition be provided and linked from the landing page.

A separate requirements class is specified for API definitions that follow the OpenAPI specification 3.0. This does not preclude that in the future or in parallel, other versions of OpenAPI or other API descriptions are provided by a server.

NOTE This approach is used to avoid lock-in to a specific approach to defining an API as it is expected that the API landscape will continue to evolve.

In this document, fragments of OpenAPI definitions are shown in YAML (YAML Ain't Markup Language) [1] since YAML is easier to read than JSON and is typically used in OpenAPI editors. YAML is described by its authors as a human friendly data serialization standard for all programming languages.

5.5.3 References to OpenAPI components in normative statements

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

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

— If the server supports an XML encoding, `xml` properties may be added to the relevant OpenAPI schema components.

— The range of values of a parameter or property may be extended (additional values) or constrained (if a subset of all possible values is applicable to the server). An example for a constrained range of values is to explicitly specify the supported values of a string parameter or property using an enum.

— The default value of a parameter may be changed or added unless a requirement explicitly prohibits this.

— Additional properties may be added to the schema definition of a Response Object.

— Informative text may be changed or added, e.g. comments or description properties.

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

5.5.4 Paths in OpenAPI definitions

All paths in an OpenAPI definition are relative to a base URL of the server.

EXAMPLE 1 URL of the OpenAPI definition.

If the OpenAPI Server Object looks like this:

```
servers:
  - url: https://dev.example.org/
    description: Development server
  - url: https://data.example.org/
    description: Production server
```

The path "/mypath" in the OpenAPI definition of a Web API would be the URL `https://data.example.org/mypath` for the production server.

5.5.5 Reusable OpenAPI components

Reusable components for OpenAPI definitions for implementations of OGC API Features are referenced from this document.

6 Overview

6.1 Design considerations

While this is the first version of the OGC API Features series, the fine-grained access to features over the Web has been supported by the OGC Web Feature Service (WFS) standard (in ISO: ISO 19142) and many implementations of that standard for many years. WFS uses a Remote-Procedure-Call-over-HTTP architectural style using XML for any payloads. When the WFS standard was originally designed in the late 1990s and early 2000s this was the state-of-the-art.

OGC API Features supports similar capabilities, but uses a modernized approach that follows the current Web architecture and in particular the W3C/OGC best practices for sharing Spatial Data on the Web as well as the W3C best practices for sharing Data on the Web.

Beside the general alignment with the architecture of the Web (e.g., consistency with HTTP/HTTPS, hypermedia controls), another goal for OGC API Features is modularization. This goal has several facets, as described below.

— Clear separation between core requirements and more advanced capabilities. This document specifies the core requirements that are relevant for almost everyone who wants to share or use feature data on a fine-grained level. Additional capabilities that several communities are using today will be specified as extensions in additional parts of the OGC API Features series.

— Technologies that change more frequently are decoupled and specified in separate modules ("requirements 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 features or resources, but about providing building blocks for fine-grained access to spatial data that can be used in Web APIs in general. In other words, a server supporting OGC API Features is not intended to implement just a standalone Features API. A corollary of this is that the same Web API may also implement other standards of the OGC API family that support additional resource types; for example, tile resources could provide access to the same features, but organized in a spatial partitioning system; or map resources could process the features and render them as map images.

Implementations of OGC API Features are intended to support two different approaches for how clients can use the API.

In the first approach, clients are implemented with knowledge about this document 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 Features.

The other approach targets developers that are not familiar with the OGC API standards, but want to interact with spatial data provided by an API that happens to implement OGC API Features. In this case the developer studies and uses 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.

6.2 Encodings

This document does not mandate any encoding or format for representing features or feature collections. In addition to rules for HTML, the standard encoding for Web content, rules for commonly used encodings for spatial data on the Web are provided (GeoJSON, GML).

None of these encodings is mandatory and an implementation of the Core requirements class may implement none of them but implement another encoding instead.

Support for HTML is recommended as HTML is the core language of the World Wide Web. A server that supports HTML will support browsing the data with a web browser and will enable search engines to crawl and index the dataset.

GeoJSON is a commonly used format that is simple to understand and well supported by tools and software libraries. Since most Web developers are comfortable with using a JSON-based format, this version of OGC API Features recommends supporting GeoJSON for encoding feature data, if the feature data can be represented in GeoJSON for the intended use.

Some examples for cases that are out-of-scope of GeoJSON are:

— when solids are used for geometries (e.g., in a 3D city model),

— geometries that include non-linear curve interpolations that cannot be simplified (e.g., use of arcs in authoritative geometries),

— geometries that have to be represented in a coordinate reference system (CRS) that is not based on WGS 84 longitude/latitude (e.g., an authoritative national CRS),

— features that have more than one geometric property.

In addition to HTML and GeoJSON, a significant volume of feature data is available in XML-based formats, notably GML. GML supports more complex requirements than GeoJSON and does not have any of the limitations mentioned in the above bullets, but as a result, GML is more complex to handle for both servers and clients. Requirements classes for GML are, therefore, included in this document. We expect that these will typically be supported by servers where users are known to expect feature data in XML/GML.

The recommendations for using HTML and GeoJSON reflect the importance of HTML and the current popularity of JSON-based data formats. As the practices in the Web community evolve, the recommendations will likely be updated in future versions of this document to provide guidance on using other encodings.

This document does not provide any guidance on other encodings. The supported encodings, or more precisely the media types of the supported encodings, can be determined from the API definition. The desired encoding is selected using HTTP content negotiation.

For example, if the server supports GeoJSON Text Sequences, an encoding that is based on JSON text sequences and GeoJSON to support streaming by making the data incrementally parseable, the media type `application/geo+json-seq` would be used.

In addition, HTTP supports compression and therefore the standard HTTP mechanisms can be used to reduce the size of the messages between the server and the client.

6.3 Examples

This document uses a simple example throughout the document: The dataset contains buildings and the server provides access to them through a single feature collection ("buildings") and two encodings, GeoJSON and HTML.

The buildings have a few (optional) properties: the polygon geometry of the building footprint, a name, the function of the building (residential, commercial or public use), the floor count and the timestamp of the last update of the building feature in the dataset.

In addition to the examples included in the document, additional and more comprehensive examples are available at http://schemas.opengis.net/ogcapi/features/part1/1.0/examples.

7 Requirements class "Core"

7.1 Overview

Requirements class	
http://www.opengis.net/spec/ogcapi-features-1/1.0/req/core	
Target type	Web API
Dependency	RFC 7230 to RFC 7235 (HTTP/1.1)
Dependency	RFC 2818 (HTTP over TLS)
Dependency	RFC 3339 (Date and Time on the Internet: Timestamps)
Dependency	RFC 8288 (Web Linking)

A server that implements this requirements class provides access to the features in a dataset.

NOTE 1 Other parts of this document can define API extensions that support multiple datasets. The statement that the features are from "a dataset" is not meant to preclude such extensions. It reflects that this document does not specify how the API publishes features or other spatial data from multiple datasets.

The entry point is a `Landing page` (path /).

NOTE 2 All paths (e.g., /) are relative. If the API covers other resources beyond those specified in this document, the landing page can also be, for example, a sub-resource of the base URL of the API.

The `Landing page` provides links to:

— the `API definition` (link relations `service-desc` and `service-doc`),

— the `Conformance declaration` (path /`conformance`, link relation `conformance`), and

— the `Collections` (path /`collections`, link relation `data`).

The `API definition` describes the capabilities of the server that can be used by clients to connect to the server or by development tools to support the implementation of servers and clients. Accessing the `API definition` using HTTP GET returns a description of the API. The `API definition` can be hosted on the API server(s) or a separate server.

The `Conformance declaration` states the conformance classes from standards or community specifications, identified by a URI, to which the API conforms. Clients can, but are not required to, use this information. Accessing the Conformance declaration using HTTP GET returns the list of URIs of conformance classes implemented by the server.

The data is organized into one or more collections, `Collections` provides information about and access to the collections.

This document specifies requirements only for collections consisting of features, i.e., each collection considered by this document is a feature collection. Other OGC API standards may add requirements for other types of collections.

NOTE 3 To support the future use of datasets with items that are not features, the term "feature" has not been added in the names of the resource types or their paths.

This document does not include any requirements about how the features in the dataset have to be aggregated into collections. A typical approach is to aggregate by feature type but any other approach that fits the dataset or the applications using this distribution may also be used.

Accessing `Collections` using HTTP GET returns a response that contains at least the list of collections. For each `Collection`, a link to the items in the collection (`Features`, path /`collections`/`{collectionId}`/`items`, link relation `items`) as well as key information about the collection. This information includes:

— a local identifier for the collection that is unique for the dataset;

— a list of coordinate reference systems (CRS) in which geometries may be returned by the server. The first CRS is the default coordinate reference system (in the *Core*, the default is always WGS 84 with axis order longitude/latitude);

— an optional title and description for the collection;

— an optional extent that can be used to provide an indication of the spatial and temporal extent of the collection, typically derived from the data;

— an optional indicator about the type of the items in the collection (the default value, if the indicator is not provided, is 'feature').

The `Collection` resource is also available at path `/collections/{collectionId}`, often with more details than included in the `Collections` response.

Each `Collection` that is a feature collection consists of features. This document only discusses the behaviour of feature collections.

Each feature in a dataset is part of exactly one collection.

Accessing the `Features` using HTTP GET returns a document consisting of features in the collection. The features included in the response are determined by the server based on the query parameters of the request. To support access to larger collections without overloading the client, the API supports paged access with links to the next page, if more features are selected than the page size.

A `bbox` or `datetime` parameter may be used to select only a subset of the features in the collection (the features that are in the bounding box or time interval). The `bbox` parameter also matches all features in the collection that are not associated with a location. The `datetime` parameter also matches all features in the collection that are not associated with a time stamp or interval.

The `limit` parameter may be used to control the subset of the selected features that should be returned in the response, the page size.

Each page may include information about the number of selected and returned features (`numberMatched` and `numberReturned`) as well as links to support paging (link relation next).

Each `Feature` (`path /collections/{collectionId}/items/{featureId}`) is also a separate resource and may be requested individually using HTTP GET.

In addition to the simple path structures described above, where all features are organized in a one-level collection hierarchy, additional parts of the OGC API Feature series are expected to provide alternate access to the features served by the API via additional, deeper collection hierarchies.

7.2 API landing page

7.2.1 Operation

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

7.2.2 Response

Requirement 2	/req/core/root-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 landingPage.yaml and include at least links to the following resources: the API definition (relation type 'service-desc' or 'service-doc') `/conformance` (relation type 'conformance') `/collections` (relation type 'data')

Schema for the landing page

```
type: object
required:
  - links
properties:
  title:
    type: string
  description:
    type: string
  links:
    type: array
    items:
      $ref: http://schemas.opengis.net/ogcapi/features/part1/1.0/openapi/schemas/link.yaml
```

EXAMPLE 1 Landing page response document.

```
{
  "title": "Buildings in Bonn",
  "description": "Access to data about buildings in the city of Bonn via a Web API that
conforms to the OGC API Features specification.",
  "links": [
    { "href": "http://data.example.org/",
      "rel": "self", "type": "application/json", "title": "this document" },
    { "href": "http://data.example.org/api",
      "rel": "service-desc", "type": "application/vnd.oai.openapi+json;version=3.0",
"title": "the API definition" },
    { "href": "http://data.example.org/api.html",
      "rel": "service-doc", "type": "text/html", "title": "the API documentation" },
    { "href": "http://data.example.org/conformance",
      "rel": "conformance", "type": "application/json", "title": "OGC API conformance
classes implemented by this server" },
    { "href": "http://data.example.org/collections",
      "rel": "data", "type": "application/json", "title": "Information about the feature
collections" }
  ]
}
```

7.2.3 Error situations

See HTTP status codes for general guidance.

7.3 API definition

7.3.1 Operation

Every API is expected to provide a definition that describes the capabilities of the server and which can be used by developers to understand the API, by software clients to connect to the server, or by development tools to support the implementation of servers and clients.

Requirement 3	**/req/core/api-definition-op**
A	The URIs of all API definitions referenced from the landing page shall support the HTTP GET method.
Permission 1	**/per/core/api-definition-uri**
A	The API definition is metadata about the API and strictly not part of the API itself, but it may be hosted as a sub-resource to the base path of the API, for example, at path /api. There is no need to include the path of the API definition in the API definition itself.
NOTE Multiple API definition formats can be supported.	

7.3.2 Response

Requirement 4	**/req/core/api-definition-success**
A	A GET request to the URI of an API definition linked from the landing page (link relations `service-desc` or `service-doc`) with an `Accept` header with the value of the link property type shall return a document consistent with the requested media type.
Recommendation 1	**/rec/core/api-definition-oas**
A	If the API definition document uses the OpenAPI Specification 3.0, the document should conform to the OpenAPI Specification 3.0 requirements class.

If the server hosts the API definition under the base path of the API (for example, at path /api, see above), there is no need to include the path of the API definition in the API definition itself.

The idea is that any OGC API Features implementation can be used by developers that are familiar with the API definition language(s) supported by the server. For example, if an OpenAPI definition is used, it should be possible to create a working client using the OpenAPI definition. The developer may need to learn a little bit about geometry data types, etc., but it should not be required to read this document to access the data via the API.

In case the API definition is based on OpenAPI 3.0, consider the two approaches discussed in OpenAPI requirements class.

7.3.3 Error situations

See HTTP status codes for general guidance.

7.4 Declaration of conformance classes

7.4.1 Operation

To support "generic" clients that want to access multiple OGC API Features implementations, and not "just" a specific API/server, the server has to declare the conformance classes it implements and conforms to.

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

7.4.2 Response

Requirement 6	/req/core/conformance-success
A	A successful execution of the operation shall be reported as a response with a HTTP status code `200`.
B	The content of that response shall be based upon the OpenAPI 3.0 schema confClasses.yaml and list all OGC API conformance classes that the server conforms to.

Schema for the list of conformance classes

```
type: object
required:
  - conformsTo
properties:
  conformsTo:
    type: array
    items:
      type: string
```

EXAMPLE 1 Conformance declaration response document.

This example response in JSON is for a server that supports OpenAPI 3.0 for the API definition and HTML and GeoJSON as encodings for features.

```
{
  "conformsTo": [
    "http://www.opengis.net/spec/ogcapi-features-1/1.0/conf/core",
    "http://www.opengis.net/spec/ogcapi-features-1/1.0/conf/oas30",
    "http://www.opengis.net/spec/ogcapi-features-1/1.0/conf/html",
    "http://www.opengis.net/spec/ogcapi-features-1/1.0/conf/geojson"
  ]
}
```

7.4.3 Error situations

See HTTP status codes for general guidance.

7.5 HTTP 1.1

Requirement 7	/req/core/http
A	The server shall conform to HTTP 1.1.
B	If the server supports HTTPS, the server shall also conform to HTTP over TLS.
NOTE This includes the correct use of, e.g. status codes, headers.	

Recommendation 2	/rec/core/head
A	The server should support the HTTP 1.1 method HEAD for all resources that support the method GET.
NOTE 1 Supporting the method HEAD in addition to GET can be useful for clients and is simple to implement. NOTE 2 Servers implementing CORS will also implement the method OPTIONS.	

7.5.1 HTTP status codes

This API standard does not impose any restrictions on which features of the HTTP and HTTPS protocols may be used. API clients should be prepared to handle any legal HTTP or HTTPS status code.

The status codes listed in Table 3 are of particular relevance to implementors of this document. Status codes 200, 400, and 404 are called out in API requirements. Therefore, support for these status codes is mandatory for all compliant implementations. The remainder of the status codes in Table 3 are not mandatory, but are important for the implementation of a well-functioning API. Support for these status codes is strongly encouraged for both client and server implementations.

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, e.g. 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 fulfil 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, e.g. a path parameter had an incorrect value.
405	The request method is not supported, e.g. a POST request was submitted, but the resource only supports GET requests.
406	Content negotiation failed. For example, the Accept header submitted in the request did not support any of the media types supported by the server for the requested resource.
500	An internal error occurred in the server.

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

Permission 2	**/per/core/additional-status-codes**
A	Servers may support other capabilities of the HTTP protocol and, therefore, may return other status codes than those listed in Table 3.

The API Description Document describes the HTTP status codes generated by that API. This should not be an exhaustive list of all possible status codes. It is not reasonable to expect an API designer to control the use of HTTP status codes which are not generated by their software. Therefore, it is recommended that the API Description Document limit itself to describing HTTP status codes relevant to the proper operation of the API application logic. Client implementations should be prepared to receive HTTP status codes in addition to those described in the API Description Document.

7.6 Unknown or invalid query parameters

Requirement 8	**/req/core/query-param-unknown**
A	The server shall respond with a response with the status code `400`, if the request URI includes a query parameter that is not specified in the API definition.

If a server wants to support vendor specific parameters, these shall be explicitly declared in the API definition.

If OpenAPI is used to represent the API definition, a capability exists to allow additional parameters without explicitly declaring them, i.e. parameters that have not been explicitly specified in the API definition for the operation will be ignored.

OpenAPI schema for additional "free-form" query parameters

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

NOTE The name of the parameter does not matter as the actual query parameters are the names of the object properties.

EXAMPLE 1 Assume that the value of `vendorSpecificParameters` 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`.

Requirement 9	/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.

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

7.7 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 RFC 7232 (HTTP/1.1).

Recommendation 3	/rec/core/etag
A	The service should support entity tags and the associated headers as specified by HTTP/1.1.

7.8 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 4	/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), and

— JSONP (JSON with padding).

7.9 Encodings

While OGC API Features does not specify any mandatory encoding, support for the following encodings is recommended. See Clause 6 (Overview) for a discussion.

Recommendation 5	/rec/core/html
A	To support browsing the dataset and its features with a web browser and to enable search engines to crawl and index the dataset, implementations should consider supporting an HTML encoding.
Recommendation 6	**/rec/core/geojson**
A	If the feature data can be represented for the intended use in GeoJSON, implementations should consider supporting GeoJSON as an encoding for features and feature collections.

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

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

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

As clients simply need to dereference the URI of the link, the implementation details and the mechanism on 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") URIs in the browser address bar, can study the API definition.

NOTE 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.

7.10 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 7	**/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.
EXAMPLE If JSON-LD is used as an encoding, the built-in capabilities to annotate a string with its language should be used.	

The link object based on RFC 8288 (Web Linking) includes a `hreflang` attribute that can be used to state the language of the referenced resource. This can be used to include links to the same data in, e.g. English or French. Just like with multiple encodings, a server that wants to use language-specific links will have to support a mechanism to mint language-specific URIs for resources in order to express links to, for example, the same resource in another language. Again, this document does not mandate any particular approach on how such a capability is supported by the server.

7.11 Coordinate reference systems

As discussed in Chapter 9 of the W3C/OGC Spatial Data on the Web Best Practices document[6], how to express and share the location of features in a consistent way is one of the most fundamental aspects of publishing geographic data and it is important to be clear about the coordinate reference system that coordinates are in.

For the reasons discussed in the Best Practices, OGC API Features uses WGS 84 longitude and latitude as the default coordinate reference system for spatial geometries.

Requirement 10	**/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.

Implementations compliant with the Core are not required to support publishing feature geometries in coordinate reference systems other than http://www.opengis.net/def/crs/OGC/1.3/CRS84 (for

coordinates without height) or http://www.opengis.net/def/crs/OGC/0/CRS84h (for coordinates with height), i.e., the (optional) third coordinate number is always the height.

The Core also does not specify a capability to request feature geometries in a different coordinate reference system. Such a capability will be specified in another part of the OGC API Features series.

The same principles apply for temporal geometries, which are measured relative to a temporal coordinate reference system. OGC API Features uses the Gregorian calendar and all dates or timestamps discussed in this document are in the Gregorian calendar and conform to RFC 3339.

Recommendation 8	**/rec/core/rfc3339**
A	RFC 3339 should also be used for feature properties that are temporal instants or intervals, where applicable, but feature properties may be represented in another format or in other temporal coordinate reference systems, too.

7.12 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. 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.

7.13 Feature collections

7.13.1 Operation

Requirement 11	**/req/core/fc-md-op**
A	The server shall support the HTTP GET operation at the path /`collections`.

7.13.2 Response

Requirement 12	**/req/core/fc-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.

Schema for the collections resource

```
type: object
required:
  - links
  - collections
properties:
  links:
    type: array
    items:
      $ref: http://schemas.opengis.net/ogcapi/features/part1/1.0/openapi/schemas/link.yaml
  collections:
    type: array
    items:
      $ref: http://schemas.opengis.net/ogcapi/features/part1/1.0/openapi/schemas/
collection.yaml
```

Requirement 13	**/req/core/fc-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 server (relation: `alternate`).
B	All links shall include the `rel` and `type` link parameters.
Recommendation 10	**/rec/core/fc-md-descriptions**
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 the whole dataset.
C	For resources that describe the contents of a feature collection, the links should be set in the `links` property of the appropriate object in the `collections` resource.
D	Examples for descriptions are: XML Schema, Schematron, JSON Schema, RDF Schema, OWL, SHACL, a feature catalogue.
Recommendation 11	**/rec/core/fc-md-license**
A	For each feature collection included in the response, the `links` property of the collection should include a link to the applicable licence (relation: `license`).
B	Alternatively, if all data shared via the API is available under the same licence, the link may instead be added to the top-level `links` property of the response.
C	Multiple links to the license in different media types may be provided. At least a link to media type `text/html` or `text/plain` should be provided.
Requirement 14	**/req/core/fc-md-items**
A	For each feature collection provided by the server, an item shall be provided in the property `collections`.
Permission 3	**/per/core/fc-md-items**
A	To support servers with many collections, servers may limit the number of items in the property `collections`.

This document does not specify mechanisms on how clients may access all collections from servers with many collections. Such mechanisms may be specified in additional parts of OGC API Features. Options include support for paging and/or filtering.

Requirement 15	**/req/core/fc-md-items-links**
A	For each feature collection included in the response, the `links` property of the collection shall include an item for each supported encoding with a link to the features resource (relation: `items`).
B	All links shall include the `rel` and `type` properties.

Requirement 16	/req/core/fc-md-extent
A	For each feature 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 feature has multiple properties with spatial or temporal information, it is the decision of the server whether only a single spatial or temporal geometry property is used to determine the extent or all relevant geometries.

NOTE The member `spatial` only needs to be provided in the `extent object` if features in the feature collection have spatial properties. The same applies to `temporal` and features with temporal properties. For example, a feature collection where features have a spatial, but no temporal property will only provide the `spatial` member.

Recommendation 12	/rec/core/fc-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 support multiple bounding boxes or time intervals.
Permission 4	**/per/core/fc-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 WGS 84 longitude/latitude and temporal extents in the Gregorian calendar (these are the only enum values in extent.yaml).
C	Extension to the Core may add additional reference systems to the `extent` object.

Schema for a feature collection

```
type: object
required:
  - id
  - links
properties:
  id:
    description: identifier of the collection used, for example, in URIs
    type: string
  title:
    description: human readable title of the collection
    type: string
  description:
    description: a description of the features in the collection
    type: string
  links:
    type: array
    items:
      $ref: http://schemas.opengis.net/ogcapi/features/part1/1.0/openapi/schemas/link.yaml
  extent:
    description: >-
      The extent of the features in the collection. In the Core only spatial and temporal
      extents are specified. Extensions may add additional members to represent other
      extents, for example, thermal or pressure ranges.
    type: object
    properties:
      spatial:
        description: >-
          The spatial extent of the features in the collection.
        type: object
```

```
  properties:
    bbox:
      description: >-
        One or more bounding boxes that describe the spatial extent of the dataset.
        In the Core only a single bounding box is supported. Extensions may support
        additional areas. If multiple areas are provided, the union of the bounding
        boxes describes the spatial extent.
      type: array
      minItems: 1
      items:
        description: >-
          Each 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
          * Minimum value, coordinate axis 3 (optional)
          * Upper right corner, coordinate axis 1
          * Upper right corner, coordinate axis 2
          * Maximum value, coordinate axis 3 (optional)

          The coordinate reference system of the values is WGS 84 longitude/latitude
          (http://www.opengis.net/def/crs/OGC/1.3/CRS84) unless a different
          coordinate reference system is specified in `crs`.

          For WGS 84 longitude/latitude the values are in most cases the sequence of
          minimum longitude, minimum latitude, maximum longitude and maximum
          latitude. However, in cases where the box spans the antimeridian the first
          value (west-most box edge) is larger than the third value (east-most box
          edge).

          If the vertical axis is included, the third and the sixth number are
          the bottom and the top of the 3-dimensional bounding box.

          If a feature 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.
        type: array
        minItems: 4
        maxItems: 6
        items:
          type: number
        example:
          - -180
          - -90
          - 180
          - 90
    crs:
      description: >-
        Coordinate reference system of the coordinates in the spatial extent
        (property `bbox`). The default reference system is WGS 84
        longitude/latitude.
        In the Core this is the only supported coordinate reference system.
        Extensions may support additional coordinate reference systems and add
        additional enum values.
      type: string
      enum:
        - 'http://www.opengis.net/def/crs/OGC/1.3/CRS84&#x0027;
      default: 'http://www.opengis.net/def/crs/OGC/1.3/CRS84&#x0027;
temporal:
  description: >-
    The temporal extent of the features in the collection.
  type: object
  properties:
    interval:
      description: >-
        One or more time intervals that describe the temporal extent of the dataset.
        The value `null` is supported and indicates an open time interval.
        In the Core only a single time interval is supported. Extensions may support
        multiple intervals. If multiple intervals are provided, the union of the
```

```
            intervals describes the temporal extent.
          type: array
          minItems: 1
          items:
            description: >-
              Begin and end times of the time interval. The timestamps are in the
              temporal coordinate reference system specified in `trs`. By default
              this is the Gregorian calendar.
            type: array
            minItems: 2
            maxItems: 2
            items:
              type: string
              format: date-time
              nullable: true
            example:
              - '2011-11-11T12:22:11Z'
              - null
        trs:
          description: >-
             Coordinate reference system of the coordinates in the temporal extent
            (property `interval`). The default reference system is the Gregorian
            calendar. In the Core this is the only supported temporal coordinate
            reference system. Extensions may support additional temporal coordinate
            reference systems and add additional enum values.
          type: string
          enum:
            - 'http://www.opengis.net/def/uom/ISO-8601/0/Gregorian&#x0027;
          default: 'http://www.opengis.net/def/uom/ISO-8601/0/Gregorian&#x0027;
  itemType:
    description: indicator about the type of the items in the collection (the default
value is 'feature').
    type: string
    default: feature
  crs:
    description: the list of coordinate reference systems supported by the service
    type: array
    items:
      type: string
    default:
      - http://www.opengis.net/def/crs/OGC/1.3/CRS84
```

NOTE The `crs` property of the `collection` object is not used by this requirements class, but is reserved for future use.

This feature collections example response in JSON is for a dataset with a single collection "buildings". It includes links to the features resource in all formats that are supported by the service (link relation type: "items").

There is a link to the feature collections response itself (link relation type: "self"). Representations of this resource in other formats are referenced using link relation type "alternate".

An additional link is to a GML application schema for the dataset - using link relation type "describedby".

A bulk download of all the features in the dataset is referenced using (link relation type: "enclosure").

Finally, there are also links to the licence information for the building data (using link relation type "licence").

Reference system information is not provided as the service provides geometries only in the default systems (spatial: WGS 84 longitude/latitude; temporal: Gregorian calendar).

EXAMPLE 1 Feature collections response document.

```
{
  "links": [
    { "href": "http://data.example.org/collections.json",
      "rel": "self", "type": "application/json", "title": "this document" },
    { "href": "http://data.example.org/collections.html",
      "rel": "alternate", "type": "text/html", "title": "this document as HTML" },
```

```
    { "href": "http://schemas.example.org/1.0/buildings.xsd",
      "rel": "describedby", "type": "application/xml", "title": "GML application schema
for Acme Corporation building data" },
    { "href": "http://download.example.org/buildings.gpkg",
      "rel": "enclosure", "type": "application/geopackage+sqlite3", "title": "Bulk
download (GeoPackage)", "length": 472546 }
  ],
  "collections": [
    {
      "id": "buildings",
      "title": "Buildings",
      "description": "Buildings in the city of Bonn.",
      "extent": {
        "spatial": {
          "bbox": [ [ 7.01, 50.63, 7.22, 50.78 ] ]
        },
        "temporal": {
          "interval": [ [ "2010-02-15T12:34:56Z", null ] ]
        }
      },
      "links": [
        { "href": "http://data.example.org/collections/buildings/items",
          "rel": "items", "type": "application/geo+json",
          "title": "Buildings" },
        { "href": "https://creativecommons.org/publicdomain/zero/1.0/",
          "rel": "license", "type": "text/html",
          "title": "CC0-1.0" },
        { "href": "https://creativecommons.org/publicdomain/zero/1.0/rdf",
          "rel": "license", "type": "application/rdf+xml",
          "title": "CC0-1.0" }
      ]
    }
  ]
}
```


7.13.3 Error situations

See HTTP status codes for general guidance.

7.14 Feature collection

7.14.1 Operation

Requirement 17	/req/core/sfc-md-op
A	The server shall support the HTTP GET operation at the path /collections/{collectionId}.
B	The parameter collectionId is each id property in the feature collections response (JSONPath: $.collections[*].id).

7.14.2 Response

Requirement 18	/req/core/sfc-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 feature collection in the /collections response, i.e. the values for id, title, description and extent shall be identical.

7.14.3 Error situations

See 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).

7.15 Features

7.15.1 Operation

Requirement 19	**/req/core/fc-op**
A	For every feature collection identified in the feature collections response (path `/collections`), the server shall support the HTTP GET operation at the path `/collections/{collectionId}/items`.
B	The parameter collectionId is each id property in the feature collections response (JSONPath: `$.collections[*].id`).

7.15.2 Parameter limit

Requirement 20	**/req/core/fc-limit-definition**

A	The operation shall support a parameter limit with the following characteristics (using an OpenAPI Specification 3.0 fragment):
	`name: limit`
	`in: query`
	`required: false`
	`schema:`
	`  type: integer`
	`  minimum: 1`
	`  maximum: 10000`
	`  default: 10`
	`style: form`
	`explode: false`

Permission 5	**/per/core/fc-limit-default-minimum-maximum**
A	The values for `minimum`, `maximum` and `default` in requirement `/req/core/fc-limit-definition` are only examples and may be changed.
Requirement 21	**/req/core/fc-limit-response-1**
A	The response shall not contain more features than specified by the optional `limit` parameter. If the API definition specifies a maximum value for `limit` parameter, the response shall not contain more features than this maximum value.
B	Only items are counted that are on the first level of the collection. Any nested objects contained within the explicitly requested items shall not be counted.
Permission 6	**/per/core/fc-limit-response-2**
A	The server may return less features than requested (but not more).

A template for the definition of the parameter in YAML according to OpenAPI 3.0 is available at limit.yaml.

7.15.3 Parameter bbox

Requirement 22	**/req/core/fc-bbox-definition**

A	The operation shall support a parameter bbox with 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 23	**/req/core/fc-bbox-response**
A	Only features that have a spatial geometry that intersects the bounding box shall be part of the result set, if the bbox parameter is provided.
B	If a feature 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 features 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; — Minimum value, coordinate axis 3 (optional); — Upper right corner, coordinate axis 1; — Upper right corner, coordinate axis 2; — Maximum value, coordinate axis 3 (optional).
E	The bounding box shall consist of four numbers and the coordinate reference system of the values shall be interpreted as WGS 84 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`.
F	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 feature. This includes the boundaries of the geometries (e.g. for curves the start and end position and for surfaces the outer and inner rings).

In case of a degenerated bounding box, the resulting geometry is used. For example, if the lower left corner is the same as the upper right corner, all features match where the geometry intersects with this point.

This document does not specify requirements for the parameter `bbox-crs`. Those requirements will be specified in an additional part of the OGC API Features series.

For WGS 84 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 WGS 84 (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`.

Note that according to the requirement to return an error for an invalid parameter value, the server will return an error, if a latitude value of `160.0` is used.

If the vertical axis is included, the third and the sixth number are the bottom and the top of the 3-dimensional bounding box.

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

7.15.4 Parameter datetime

Requirement 24	**/req/core/fc-time-definition**

A	The operation shall support a parameter `datetime` with the following characteristics (using an OpenAPI Specification 3.0 fragment):
	`name: datetime`
	`in: query`
	`required: false`
	`schema:`
	`  type: string`
	`style: form`
	`explode: false`

Requirement 25	**/req/core/fc-time-response**
A	Only features that have a temporal geometry that intersects the temporal information in the datetime parameter shall be part of the result set, if the parameter is provided.
B	If a feature has multiple temporal properties, it is the decision of the server whether only a single temporal property is used to determine the extent or all relevant temporal properties.
C	The `datetime` parameter shall also match all features in the collection that are not associated with a temporal geometry.

D	Temporal geometries are either a date-time value or a time interval. The parameter value shall conform to the following syntax (using ABNF):
	`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 3339:2002, 5.6.
F	Open ranges in time intervals at the start or end are supported using a double-dot (..) or an empty string for the start/end.

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

NOTE ISO 8601-2 distinguishes open start/end timestamps (double-dot) and unknown start/end timestamps (empty string). For queries, an unknown start/end has the same effect as an open start/end.

EXAMPLE 1 A date-time

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

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

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

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

EXAMPLE 2 Intervals.

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

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

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

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

For features with a temporal property that is a timestamp (like `lastUpdate` in the building features), a time interval would match all features where the temporal property is within the interval.

For features with a temporal property that is a date or a time interval, a time interval would match all features where the values overlap.

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

7.15.5 Parameters for filtering on feature properties

Recommendation 13	**/rec/core/fc-filters**

A	If features in the feature collection include a feature property that has a simple value (for example, a string or integer) that is expected to be useful for applications using the service to filter the features of the collection based on this property, a parameter with the name of the feature property and with the following characteristics (using an OpenAPI Specification 3.0 fragment) should be supported:
	
	`in: query`
	`required: false`
	`style: form`
	`explode: false`
	
	The `schema` property should be the same as the definition of the feature property in the response schema.

EXAMPLE 1 *An additional parameter to filter buildings based on their function.*

```
name: function
in: query
description: >-
  Only return buildings of a particular function.\
```

```
  Default = return all buildings.
required: false
schema:
  type: string
  enum:
    - residential
    - commercial
    - public use
style: form
explode: false
example: 'function=public+use'
```

EXAMPLE 2 *An additional parameter to filter buildings based on their name.*

```
name: name
in: query
description: >-
  Only return buildings with a particular name. Use '*' as a wildcard.\

  Default = return all buildings.
required: false
schema:
  type: string
style: form
explode: false
example: 'name=A*'
```

For string-valued properties, servers could support wildcard searches. The example included in the OpenAPI fragment would search for all buildings with a name that starts with "A".

7.15.6 Combinations of filter parameters

Any combination of `bbox`, `datetime` and parameters for filtering on feature properties is allowed. Note that the requirements on these parameters imply that only features matching all the predicates are in the result set, i.e. the logical operator between the predicates is 'AND'.

7.15.7 Response

Requirement 26	**/req/core/fc-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 features selected by the request.

The number of features returned depends on the server and the parameter `limit`:

— The client can request a limit it is interested in.

— The server likely has a default value for the limit, and a maximum limit.

— If the server has any more results available than it returns (the number it returns is less than or equal to the requested/default/maximum limit) then the server will include a link to the next set of results.

So (using the default/maximum values of 10/10000 from the OpenAPI fragment in requirement `/req/core/fc-limit-definition`):

— If you ask for 10, you will get 0 to 10 (as requested) and if there are more, a `next` link;

— If you do not specify a limit, you will get 0 to 10 (default) and if there are more, a `next` link;

— If you ask for 50000, you can get up to 10000 (server-limited) and if there are more, a `next` link;

— If you follow the `next` link from the previous response, you can get up to 10000 additional features and if there are more, a `next` link.

Requirement 27	**/req/core/fc-links**
A	A `200`-response shall include the following links: — a link to this response document (relation: `self`), — a link to the response document in every other media type supported by the service (relation: `alternate`).
Recommendation 14	**/rec/core/fc-next-1**
A	A `200`-response should include a link to the next "page" (relation: `next`), if more features have been selected than returned in the response.
Recommendation 15	**/rec/core/fc-next-2**
A	Dereferencing a `next` link should return additional features from the set of selected features that have not yet been returned.
Recommendation 16	**/rec/core/fc-next-3**
A	The number of features in a response to a `next` link should follow the same rules as for the response to the original query and again include a `next` link, if there are more features in the selection that have not yet been returned.

This document does not mandate any specific implementation approach for the `next` links.

An implementation could use opaque links that are managed by the server. It is up to the server to determine how long these links can be de-referenced. Clients should be prepared to receive a 404 response.

Another implementation approach is to use an implementation-specific parameter that specifies the index within the result set from which the server begins presenting results in the response, like the `startIndex` parameter that was used in WFS 2.0 (and which may be added again in additional parts of the OGC API Features series).

Clients should not assume that paging is safe against changes to dataset while a client iterates through next links. If a server provides opaque links these could be safe and maintain the dataset state during the original request. Using a parameter for the start index, however, will not be safe.

NOTE 1 Additional requirements classes for safe paging or an index parameter can be added in extensions to this document.

Permission 7	**/per/core/fc-prev**
A	A response to a `next` link may include a `prev` link to the resource that included the `next` link.

Providing `prev` links supports navigating back and forth between pages, but depending on the implementation approach, it may be too complex to implement.

Requirement 28	**/req/core/fc-rel-type**
A	All links shall include the `rel` and `type` link parameters.
Requirement 29	**/req/core/fc-timeStamp**
A	If a property `timeStamp` is included in the response, the value shall be set to the time stamp when the response was generated.
Requirement 30	**/req/core/fc-numberMatched**
A	If a property `numberMatched` is included in the response, the value shall be identical to the number of features in the feature collections that match the selection parameters like `bbox`, `datetime` or additional filter parameters.
B	A server may omit this information in a response, if the information about the number of matching features is not known or difficult to compute.

Requirement 31	/req/core/fc-numberReturned
A	If a property `numberReturned` is included in the response, the value shall be identical to the number of features in the response.
B	A server may omit this information in a response, if the information about the number of features in the response is not known or difficult to compute.

NOTE 2 The representation of the links and the other properties in the payload depends on the encoding of the feature collection.

If the request is to return building features and "10" is the default limit, the links in the response could be (in this example represented as link headers and using an additional parameter `offset` to implement `next` links - and the optional `prev` links):

EXAMPLE 1 *Links*

```
Link: <http://data.example.org/collections/buildings/items.json>; rel="self";
type="application/geo+json"
Link: <http://data.example.org/collections/buildings/items.html>; rel="alternate";
type="text/html"
Link: <http://data.example.org/collections/buildings/items.json?offset=10>; rel="next";
type="application/geo+json"
```

Following the next link could return:

```
Link: <http://data.example.org/collections/buildings/items.json?offset=10>; rel="self";
type="application/geo+json"
Link: <http://data.example.org/collections/buildings/items.html?offset=10>;
rel="alternate"; type="text/html"
Link: <http://data.example.org/collections/buildings/items.json?offset=0>; rel="prev";
type="application/geo+json"
Link: <http://data.example.org/collections/buildings/items.json?offset=20>; rel="next";
type="application/geo+json"
```

If an explicit limit of "50" is used, the links in the response could be:

```
Link: <http://data.example.org/collections/buildings/items.json?limit=50>; rel="self";
type="application/geo+json"
Link: <http://data.example.org/collections/buildings/items.html?limit=50>;
rel="alternate"; type="text/html"
Link: <http://data.example.org/collections/buildings/items.json?limit=50&#x0026;off
set=50>; rel="next"; type="application/geo+json"
```

Following the next link could return:

```
Link: <http://data.example.org/collections/buildings/items.json?limit=50&#x0026;off
set=50>; rel="self"; type="application/geo+json"
Link: <http://data.example.org/collections/buildings/items.html?limit=50&#x0026;off
set=50>; rel="alternate"; type="text/html"
Link: <http://data.example.org/collections/buildings/items.json?limit=50&#x0026;offset=0>;
rel="prev"; type="application/geo+json"
Link: <http://data.example.org/collections/buildings/items.json?limit=50&#x0026;off
set=100>; rel="next"; type="application/geo+json"
```

7.15.8 Error situations

See HTTP status codes for general guidance.

If the path parameter `collectionId` does not exist on the server, the status code of the response will be `404`.

A `400` will be returned in the following situations:

— if query parameter `limit` is not an integer or not between minimum and maximum;

— if query parameter `bbox` does not have 4 (or 6) numbers or they do not form a bounding box;

— if parameter `datetime` is not a valid time stamp or time interval.

7.16 Feature

7.16.1 Operation

Requirement 32	**/req/core/f-op**
A	For every feature in a feature collection (path `/collections/{collectionId}`), the server shall support the HTTP GET operation at the path `/collections/{collectionId}/items/{featureId}`.
B	The parameter `collectionId` is each id property in the feature collections response (JSONPath: `$.collections[*].id`). `featureId` is a local identifier of the feature.
Permission 8	**/per/core/f-id**
A	The Core requirements class only requires that the feature URI be unique. Implementations may apply stricter rules and, for example, use of unique `id` values per dataset or collection.

7.16.2 Response

Requirement 33	**/req/core/f-success**
A	A successful execution of the operation shall be reported as a response with a HTTP status code `200`.
Requirement 34	**/req/core/f-links**
A	A `200`-response shall include the following links in the response: — a link to the response document (relation: `self`), — a link to the response document in every other media type supported by the service (relation: `alternate`), and — a link to the feature collection that contains this feature (relation: `collection`).
B	All links shall include the `rel` and `type` link parameters.

NOTE The representation of the links in the payload will depend on the encoding of the feature.

EXAMPLE 1 *Links*

The links in a feature could be (in this example represented as link headers):

```
Link: <http://data.example.org/collections/buildings/items/123.json>; rel="self";
type="application/geo+json"
Link: <http://data.example.org/collections/buildings/items/123.html>; rel="alternate";
type="text/html"
Link: <http://data.example.org/collections/buildings/items.json>; rel="collection";
type="application/geo+json"
```

7.16.3 Error situations

See HTTP status codes for general guidance.

If the path parameter `collectionId` or the path parameter `featureId` do not exist on the server, the status code of the response will be `404`.

8 Requirements classes for encodings

8.1 Overview

This clause specifies four pre-defined requirements classes for encodings to be used by an OGC API Features implementation. These encodings are commonly used encodings for spatial data on the web:

— HTML;

— GeoJSON;

— Geography Markup Language (GML), Simple Features Profile, Level 0;

— Geography Markup Language (GML), Simple Features Profile, Level 2.

None of these encodings are mandatory and an implementation of the Core requirements class may also implement none of them but implement another encoding instead.

The Core requirements class includes recommendations to support HTML and GeoJSON as encodings, where practical. Clause 6 (Overview) includes a discussion about recommended encodings.

8.2 Requirements Class "HTML"

Geographic information that is only accessible in formats like GeoJSON or GML has two issues:

— The data is not discoverable using the most common mechanism for discovering information, i.e. the search engines of the Web;

— 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, it should be done in a way that enables users and search engines to access all data.

This is discussed in detail in Best Practice 2: Make your spatial data indexable by search engines[6]. This document therefore recommends supporting HTML as an encoding.

Requirements Class	
http://www.opengis.net/spec/ogcapi-features-1/1.0/req/html	
Target type	Web API
Dependency	Requirements Class "Core"
Dependency	HTML5
Dependency	schema.org
Requirement 35	**/req/html/definition**
A	Every `200`-response of an operation of the server shall support the media type `text/html`.
Requirement 36	**/req/html/content**
A	Every `200`-response of the server with the media type "text/html" SHALL be a HTML 5 document that includes the following information in the HTML body: — all information identified in the schemas of the Response Object in the HTML `<body>`, and — all links in HTML `<a>` elements in the HTML `<body>`.
Recommendation 17	**/rec/html/schema-org**
A	A `200`-response with the media type `text/html`, should include schema.org annotations.

8.3 Requirements Class "GeoJSON"

GeoJSON is a commonly used format that is simple to understand and well supported by tools and software libraries. Since most Web developers are comfortable with using a JSON-based format, supporting GeoJSON is recommended, if the feature data can be represented in GeoJSON for the intended use.

Requirements Class	
http://www.opengis.net/spec/ogcapi-features-1/1.0/req/geojson	
Target type	Web API
Dependency	Requirements Class "Core"
Dependency	GeoJSON
Requirement 37	**/req/geojson/definition**
A	`200`-responses of the server shall support the following media types: — `application/geo+json` for resources that include feature content, and — `application/json` for all other resources.
Requirement 38	**/req/geojson/content**
A	Every `200`-response with the media type `application/geo+json` shall be — a GeoJSON FeatureCollection Object for features, and — a GeoJSON Feature Object for a single feature.
B	The links specified in the requirements `/req/core/fc-links` and `/req/core/f-links` shall be added in an extension property (foreign member) with the name `links`.
C	The schema of all responses with the media type `application/json` shall conform with the JSON Schema specified for the resource in the Core requirements class.

Templates for the definition of the schemas for the GeoJSON responses in OpenAPI definitions are available at featureCollectionGeoJSON.yaml and featureGeoJSON.yaml. These are generic schemas that do not include any application schema information about specific feature types or their properties.

EXAMPLE 1 *A GeoJSON FeatureCollection Object response*

In the example below, only the first and tenth feature is shown. Coordinates are not shown.

```
{
  "type" : "FeatureCollection",
  "links" : [ {
    "href" : "http://data.example.com/collections/buildings/items?f=json",
    "rel" : "self",
    "type" : "application/geo+json",
    "title" : "this document"
  }, {
    "href" : "http://data.example.com/collections/buildings/items?f=html",
    "rel" : "alternate",
    "type" : "text/html",
    "title" : "this document as HTML"
  }, {
    "href" : "http://data.example.com/collections/buildings/items?f=json&#x0026;offset=10&
#x0026;limit=10",
    "rel" : "next",
    "type" : "application/geo+json",
    "title" : "next page"
  } ],
  "timeStamp" : "2018-04-03T14:52:23Z",
  "numberMatched" : 123,
  "numberReturned" : 10,
  "features" : [ {
```

```
    "type" : "Feature",
    "id" : "123",
    "geometry" : {
      "type" : "Polygon",
      "coordinates" : [ ... ]
    },
    "properties" : {
      "function" : "residential",
      "floors" : "2",
      "lastUpdate" : "2015-08-01T12:34:56Z"
    }
  }, { ...
  }, {
    "type" : "Feature",
    "id" : "132",
    "geometry" : {
      "type" : "Polygon",
      "coordinates" : [ ... ]
    },
    "properties" : {
      "function" : "public use",
      "floors" : "10",
      "lastUpdate" : "2013-12-03T10:15:37Z"
    }
  } ]
}
```


EXAMPLE 2 *A GeoJSON FeatureCollection Object response*

In the example below, coordinates are not shown.


```
{
  "type" : "Feature",
  "links" : [ {
    "href" : "http://data.example.com/collections/buildings/items/123?f=json",
    "rel" : "self",
    "type" : "application/geo+json",
    "title" : "this document"
  }, {
    "href" : "http://data.example.com/collections/buildings/items/123?f=html",
    "rel" : "alternate",
    "type" : "text/html",
    "title" : "this document as HTML"
  }, {
    "href" : "http://data.example.com/collections/buildings",
    "rel" : "collection",
    "type" : "application/geo+json",
    "title" : "the collection document"
  } ],
  "id" : "123",
  "geometry" : {
    "type" : "Polygon",
    "coordinates" : [ ... ]
  },
  "properties" : {
    "function" : "residential",
    "floors" : "2",
    "lastUpdate" : "2015-08-01T12:34:56Z"
  }
}
```


8.4 Requirements Class "Geography Markup Language (GML), Simple Features Profile, Level 0"

In addition to HTML and GeoJSON, a significant volume of feature data is available in XML-based formats, notably GML. Therefore, this document specifies requirements classes for GML. The Simple Features Profile, Level 0, is the simplest profile of GML and is typically supported by tools.

The GML Simple Features Profile is restricted to data with 2D geometries with linear/planar interpolation (points, line strings, polygons). In addition, the Level 0 profile is limited to features that can be stored in a tabular data structure.

Requirements class	
http://www.opengis.net/spec/ogcapi-features-1/1.0/req/gmlsf0	
Target type	Web API
Dependency	Requirements Class "Core"
Dependency	Geography Markup Language (GML), Simple Features Profile, Level 0
Requirement 39	**/req/gmlsf0/definition**
A	`200`-responses of the server shall support the following media types: — `application/gml+xml; version=3.2; profile="http://www.opengis.net/def/profile/ogc/2.0/gml-sf0"` for resources that include feature content, — `application/xml` for all other resources.
Requirement 40	**/req/gmlsf0/content**
A	Table 4 specifies the XML document root element that the server shall return in a `200`-response for each resource.
B	Every representation of a feature shall conform to the GML Simple Features Profile, Level 0 and be substitutable for `gml:AbstractFeature`.
C	The schema of all responses with a root element in the core namespace shall validate against the OGC API Features Core XML Schema.
Requirement 41	**/req/gmlsf0/headers**
A	If a property `timeStamp` is included in the response, its value shall be reported using the HTTP header named Date (see RFC 7231:2014, 7.1.1.2).
B	If a property `numberMatched` is included in the response, its value SHALL be reported using an HTTP header named `OGC-NumberMatched`.
C	If a property `numberReturned` is included in the response, its value SHALL be reported using an HTTP header named `OGC-NumberReturned`.
D	If links are included in the response, each link SHALL be reported using an HTTP header named `Link` (see RFC 8288:2018, Clause 3).

Table 4 — Media types and XML elements for each resource

Resource	**Path**	**XML root element**
Landing page	`/`	`core:LandingPage`
Conformance declaration	`/conformance`	`core:ConformsTo`
Feature collections	`/collections`	`core:Collections`
Feature collection	`/collections/{collectionId}`	`core:Collections`, with just one entry for the collection `collectionId`
Features	`/collections/{collectionId}/items`	`sf:FeatureCollection`
Feature	`/collections/{collectionId}/items/{featureId}`	substitutable for `gml:AbstractFeature`

The namespace prefixes used above and in the OGC API Features Core XML schemas are:

— `core: http://www.opengis.net/ogcapi-features-1/1.0`

— `sf: http://www.opengis.net/ogcapi-features-1/1.0/sf`

— `gml: http://www.opengis.net/gml/3.2`

— `atom: http://www.w3.org/2005/Atom`

— `xlink: http://www.w3.org/1999/xlink`

The mapping of the content from the responses specified in the Core requirements class to the XML is straightforward. All links have to be encoded as HTTP header `Link`.

See 6.3 for links to example responses in XML.

8.5 Requirements class "Geography Markup Language (GML), Simple Features Profile, Level 2"

The difference between this requirements class and the Level 0 requirements class is that non-spatial feature properties are not restricted to atomic values (e.g. strings, numbers).

Requirements class	
http://www.opengis.net/spec/ogcapi-features-1/1.0/req/gmlsf2	
Target type	Web API
Dependency	Requirements Class "Core"
Dependency	Geography Markup Language (GML), Simple Features Profile, Level 2
Requirement 42	**/req/gmlsf2/definition**
A	`200`-responses of the server shall support the following media types: — `application/gml+xml; version=3.2; profile=http://www.opengis.net/def/profile/ogc/2.0/gml-sf2` for resources that include feature content, — `application/xml` for all other resources.
Requirement 43	**/req/gmlsf2/content**
A	The requirement /req/gmlsf0/content applies, too, with the following changes: — All references to media type `application/gml+xml; version=3.2; profile="http://www.opengis.net/def/profile/ogc/2.0/gml-sf0"` are replaced by `application/gml+xml; version=3.2; profile=http://www.opengis.net/def/profile/ogc/2.0/gml-sf2`. — All references to "GML Simple Features Profile, Level 0" are replaced by "GML Simple Features Profile, Level 2".
Requirement 44	**/req/gmlsf2/headers**
A	The requirement /req/gmlsf0/headers applies.

9 Requirements class "OpenAPI 3.0"

9.1 Basic requirements

Servers conforming to this requirements class define their API by an OpenAPI Document.

Requirements class	
http://www.opengis.net/spec/ogcapi-features-1/1.0/req/oas30	
Target type	Web API
Dependency	Requirements Class "Core"
Dependency	OpenAPI Specification 3.0
Requirement 45	**/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.

The requirements /req/core/root-success and /req/core/api-definition-success in Core require that the API definition documents are referenced from the landing page.

Requirement 46	**/req/oas30/oas-definition-2**
A	The JSON representation shall conform to the OpenAPI Specification, version 3.0.

OpenAPI definitions can be created using different approaches. A typical example is the representation of the feature collections. One approach is to use a path parameter `collectionId`, i.e., the API definition has only a single path entry for all feature collections. Another approach is to explicitly define each feature collection in a separate path and without a path parameter, which allows to specify filter parameters or explicit feature schemas per feature collection. Both approaches are valid.

Requirement 47	**/req/oas30/oas-impl**
A	The server shall implement all capabilities specified in the OpenAPI definition.

9.2 Complete definition

Requirement 48	**/req/oas30/completeness**
A	The OpenAPI definition shall specify for each operation all HTTP Status Codes and Response Objects that the server uses in responses.
B	This includes the successful execution of an operation as well as all error situations that originate from the server.

Note that servers that, for example, are access-controlled (see Security), support web cache validation, 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.

It is possible for Clients to receive responses not documented in the OpenAPI definition. For example, additional errors may occur in the transport layer outside of the server.

9.3 Exceptions

Requirement 49	/req/oas30/exceptions-codes
A	For error situations that originate from the 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/features/part1/1.0/openapi/schemas/
exception.yamltext/html:
    schema:
      type: string
```

9.4 Security

Requirement 50	/req/oas30/security
A	For cases where the operations of the server are access-controlled, the security scheme(s) shall be documented in the OpenAPI definition.

The OpenAPI specification currently supports the following security schemes:

— HTTP authentication,

— an API key (either as a header or as a query parameter),

— OAuth2's common flows (implicit, password, application and access code) as defined in RFC 6749, and

— OpenID Connect Discovery.

9.5 Features

Recommendation 18	/rec/oas30/f-key-properties
A	The schema for the Response Objects of the HTTP GET operation for features should include key feature properties of the features in that feature collection. This is particularly helpful if filter parameters are defined for the collection (see recommendation `/rec/core/fc-filters`).

10 Media types

JSON media types that would typically be used in a server that supports JSON are:

— `application/geo+json` for feature collections and features, and

— `application/json` for all other resources.

XML media types that would typically occur in a server that supports XML are:

— `application/gml+xml;version=3.2` for any GML 3.2 feature collections and features,

— `application/gml+xml;version=3.2;profile="http://www.opengis.net/def/profile/ogc/2.0/gml-sf0"` for GML 3.2 feature collections and features conforming to the GML Simple Feature Level 0 profile,

— `application/gml+xml;version=3.2;profile=http://www.opengis.net/def/profile/ogc/2.0/gml-sf2` for GML 3.2 feature collections and features conforming to the GML Simple Feature Level 2 profile, and

— `application/xml` for all other resources.

The typical HTML media type for all "web pages" in a server would be `text/html`.

The media type for an OpenAPI 3.0 definition is `application/vnd.oai.openapi+json;version=3.0 (JSON)` or .`application/vnd.oai.openapi;version=3.0 (YAML)`.

NOTE The OpenAPI media types have not been registered yet with IANA and can change in the future.

11 Security considerations

11.1 General

A Web API is a powerful tool for sharing information and analysis resources. It also provides many avenues for unscrupulous users to attack those resources. Designers and developers of Web APIs should be familiar with the potential vulnerabilities and how to address them.

A valuable resource is the Common Weakness Enumeration (CWE) registry at http://cwe.mitre.org/data/index.html. The CWE is organized around three views: Research, Architectural, and Development.

— **Research:** facilitates research into weaknesses and can be leveraged to systematically identify theoretical gaps within CWE.

— **Architectural:** organizes weaknesses according to common architectural security tactics. It is intended to assist architects in identifying potential mistakes that can be made when designing software.

— **Development:** organizes weaknesses around concepts that are frequently used or encountered in software development.

API developers should focus on the Development view. These vulnerabilities primarily deal with the details of software design and implementation.

API designers should focus primarily on the Architectural view. However, there are critical vulnerabilities described in the Development view which are also relevant to API design. Vulnerabilities described under the following categories are particularly important:

— Pathname Traversal and Equivalence Errors;

— Channel and Path Errors;

— Web Problems.

Many of the vulnerabilities described in the CWE are introduced through the HTTP protocol. API designers and developers should be familiar with how the HTTP 1.1 addresses these vulnerabilities. This information can be found in the “Security Considerations” sections of the IETF RFCs 7230 to 7235.

The following subclauses describe some of the most serious vulnerabilities which can be mitigated by the API designer and developer. These are high-level generalizations of the more detailed vulnerabilities described in the CWE.

11.2 Multiple access routes

APIs deliver a representation of a resource. OGC APIs can deliver multiple representations (formats) of the same resource. An attacker might find that information which is prohibited in one representation can be accessed through another. API designers must take care that the access controls on their

resources are implemented consistently across all representations. That does not mean that they have to be the same. For example:

— HTML vs. GeoTIFF: The HTML representation may consist of a text description of the resource accompanied by a thumbnail image. This has less information than the GeoTIFF representation and may be subject to more liberal access policies.

— Data Centric Security: Techniques to embed access controls into the representation itself. A GeoTIFF with Data Centric Security would have more liberal access policies than a GeoTIFF without.

Bottom Line: the information content of the resources exposed by an API shall be protected to the same level across all access routes.

11.3 Multiple servers

The implementation of an API may span a number of servers. Each server is an entry point into the API. Without careful management, information which is not accessible through one server can be accessible through another.

Bottom Line: Understand the information flows through your API and verify that information is properly protected along all access paths.

11.4 Path manipulation on GET

RFC 2626:1999, section 15.2 states that if an HTTP server translates HTTP URIs directly into file system calls, the server has to take special care not to serve files that were not intended to be delivered to HTTP clients. The threat is that an attacker could use the HTTP path to access sensitive data, such as password files, which could be used to further subvert the server.

Bottom Line: Validate all GET URLs to make sure they are not trying to access resources they should not have access to.

11.5 Path manipulation on PUT and POST

A transaction operation adds new or updates existing resources on the API. This capability provides a whole new set of tools to an attacker.

Many of the resources exposed through an OGC API include hyperlinks to other resources. API clients follow these hyperlinks to access new resources or alternate representations of a resource. Once a client authenticates to an API, they tend to trust the data returned by that API. However, a resource posted by an attacker could contain hyperlinks which contain an attack. For example, the link to an alternate representation could require the client to re-authenticate prior to passing them on to the original destination. The client sees the representation they asked for and the attacker collects the clients' authentication credentials.

Bottom Line: APIs which support transaction operations should validate that an update does not contain any malignant content prior to exposing it through the API.

Annex A
(normative)

Abstract test suite

A.1 General

OGC API Features is not a Web Service in the traditional sense. Rather, it defines the behaviour and content of a set of Resources exposed through a Web Application Programming Interface (Web API). Therefore, an API may expose resources in addition to those defined by the standard. A test engine must be able to traverse the API, identify and validate test points, and ignore resource paths which are not to be tested.

A.2 Conformance class core

Conformance class	
http://www.opengis.net/spec/ogcapi-features-1/1.0/conf/core	
Target type	Web API
Requirements class	Requirements Class "Core"

A.2.1 General tests

A.2.1.1 HTTP

Abstract test 1	**/conf/core/http**
Test purpose	Validate that the resource paths advertised through the API conform with HTTP 1.1 and, where appropriate, TLS.
Requirement	/req/core/http
Test method	1. All compliance tests shall be configured to use the HTTP 1.1 protocol exclusively. 2. For APIs which support HTTPS, all compliance tests shall be configured to use HTTP over TLS (RFC 2818) with their HTTP 1.1 protocol.

A.2.1.2 CRS 84

Abstract test 2	**/conf/core/crs84**
Test purpose	Validate that all spatial geometries provided through the API are in the CRS84 coordinate reference system unless otherwise requested by the client.
Requirement	/req/core/crs84
Test method	1. Do not specify a coordinate reference system in any request. All spatial data should be in the CRS84 reference system. 2. Validate retrieved spatial data using the CRS84 reference system.

A.2.2 Landing page {root}/

Abstract test 3	**/conf/core/root-op**
Test purpose	Validate that a landing page can be retrieved from the expected location.
Requirement	/req/core/root-op

Test method	1. Issue an HTTP GET request to the URL {root}/ 2. Validate that a document was returned with a status code 200 3. Validate the contents of the returned document using test /conf/core/root-success.
Abstract test 4	**/conf/core/root-success**
Test purpose	Validate that the landing page complies with the required structure and contents.
Requirement	/req/core/root-success
Test method	Validate the landing page for all supported media types using the resources and tests identified in Table 5. For formats that require manual inspection, perform the following: a. Validate that the landing page includes a "service-desc" and/or "service-doc" link to an API Definition; b. Validate that the landing page includes a "conformance" link to the conformance class declaration; c. Validate that the landing page includes a "data" link to the Feature contents.

The landing page may be retrieved in a number of different formats. Table A.1 identifies the applicable schema document for each format and the test to be used to validate the landing page against that schema. All supported formats should be exercised.

Table A.1 — Schema and Tests for Landing Pages

Format	Schema document	Test ID
HTML	landingPage.yaml	/conf/html/content
GeoJSON	landingPage.yaml	/conf/geojson/content
GMLSF0	core.xsd, element core:LandingPage	/conf/gmlsf0/content
GMLSF2	core.xsd, element core:LandingPage	/conf/gmlsf2/content

A.2.3 API definition path {root}/api (link)

Abstract test 5	**/conf/core/api-definition-op**
Test purpose	Validate that the API Definition document can be retrieved from the expected location.
Requirement	/req/core/api-definition-op
Test method	1. Construct a path for each API Definition link on the landing page. 2. Issue a HTTP GET request on each path. 3. Validate that a document was returned with a status code 200. 4. Validate the contents of the returned document using test /conf/core/api-definition-success.
Abstract test 6	**/conf/core/api-definition-success**
Test purpose	Validate that the API Definition complies with the required structure and contents.
Requirement	/req/core/api-definition-success
Test method	Validate the API Definition document against an appropriate schema document.

A.2.4 Conformance path {root}/conformance

Abstract test 7	**/conf/core/conformance-op**
Test purpose	Validate that a Conformance Declaration can be retrieved from the expected location.
Requirement	/req/core/conformance-op
Test method	1. Construct a path for each "conformance" link on the landing page as well as for the {root}/conformance path. 2. Issue an HTTP GET request on each path. 3. Validate that a document was returned with a status code 200. 4. Validate the contents of the returned document using test /conf/core/conformance-success.
Abstract test 8	**/conf/core/conformance-success**
Test purpose	Validate that the Conformance Declaration response complies with the required structure and contents.
Requirement	/req/core/conformance-success
Test method	1. Validate the response document against OpenAPI 3.0 schema confClasses.yaml. 2. Validate that the document includes the conformance class "http://www.opengis.net/spec/ogcapi-features-1/1.0/conf/core". 3. Validate that the document lists all OGC API conformance classes that the API implements.

A.2.5 Feature collections {root}/collections

Abstract test 9	**/conf/core/fc-md-op**
Test purpose	Validate that information about the Collections can be retrieved from the expected location.
Requirement	/req/core/fc-md-op
Test method	1. Issue an HTTP GET request to the URL {root}/collections. 2. Validate that a document was returned with a status code 200. 3. Validate the contents of the returned document using test /conf/core/fc-md-success.
Abstract test 10	**/conf/core/fc-md-success**
Test purpose	Validate that the Collections content complies with the required structure and contents.
Requirement	/req/core/fc-md-success, /req/core/crs84
Test method	1. Validate that all response documents comply with /conf/core/fc-md-links. 2. Validate that all response documents comply with /conf/core/fc-md-items. 3. In case the response includes a "crs" property, validate that the first value is either "http://www.opengis.net/def/crs/OGC/1.3/CRS84" or "http://www .opengis.net/def/crs/OGC/0/CRS84h". 4. Validate the Collections content for all supported media types using the resources and tests identified in Table 6.

The Collections content may be retrieved in a number of different formats. Table A.2 identifies the applicable schema document for each format and the test to be used to validate the collections content against that schema. All supported formats should be exercised.

Table A.2 — Schema and tests for collections content

Format	Schema document	Test ID
HTML	collections.yaml	/conf/html/content
GeoJSON	collections.yaml	/conf/geojson/content
GMLSF0	core.xsd, element `core:Collections`	/conf/gmlsf0/content
GMLSF2	core.xsd, element `core:Collections`	/conf/gmlsf2/content

Abstract test 11	**/conf/core/fc-md-links**
Test purpose	Validate that the required links are included in the Collections Metadata document.
Requirement	/req/core/fc-md-links
Test method	Verify that the response document includes: 1. a link to this response document (relation: `self`), 2. a link to the response document in every other media type supported by the server (relation: `alternate`). Verify that all links include the `rel` and `type` link parameters.

Abstract test 12	**/conf/core/fc-md-items**
Test purpose	Validate that each collection provided by the server is described in the Collections Metadata.
Requirement	/req/core/fc-md-links
Test method	1. Verify that there is an entry in the collections array of the Collections Metadata for each feature collection provided by the API. 2. Verify that each collection entry includes an identifier. 3. Verify that each collection entry includes links in accordance with /conf/core/fc-md-items-links. 4. Verify that if the collection entry includes an extent property, that that property complies with /conf/core/fc-md-extent. 5. Validate each collection entry for all supported media types using the resources and tests identified in Table 7.

The collection entries may be encoded in a number of different formats. Table A.3 identifies the applicable schema document for each format and the test to be used to validate the collection entries against that schema. All supported formats should be exercised.

Table A.3 — Schema and Tests for Collection Entries

Format	Schema document	Test ID
HTML	collection.yaml	Manual Inspection
GeoJSON	collection.yaml	/conf/geojson/content
GMLSF0	core.xsd, element `core:Collections`	/conf/gmlsf0/content
GMLSF2	core.xsd, element `core:Collections`	/conf/gmlsf2/content