OGC API - Features - Part 1

Core
# Table of Contents

1. Scope ......................................................................................................................... 8  
2. Conformance ............................................................................................................... 9  
3. References .................................................................................................................. 11  
4. Terms and Definitions ................................................................................................. 12  
   4.1. dataset .................................................................................................................. 12  
   4.2. distribution ........................................................................................................... 12  
   4.3. feature .................................................................................................................. 12  
   4.4. feature collection; collection ............................................................................... 12  
   4.5. Web API ............................................................................................................... 12  
5. Conventions ................................................................................................................... 13  
   5.1. Identifiers ............................................................................................................ 13  
   5.2. UML model .......................................................................................................... 13  
   5.3. Link relations ...................................................................................................... 13  
   5.4. Use of HTTPS ...................................................................................................... 14  
   5.5. HTTP URIs .......................................................................................................... 14  
   5.6. API definition ..................................................................................................... 14  
      5.6.1. General remarks ............................................................................................ 14  
      5.6.2. Role of OpenAPI ........................................................................................... 14  
      5.6.3. References to OpenAPI components in normative statements ................... 15  
      5.6.4. Paths in OpenAPI definitions ....................................................................... 15  
      5.6.5. Reusable OpenAPI components .................................................................. 16  
6. Overview ...................................................................................................................... 17  
   6.1. Design considerations ......................................................................................... 17  
   6.2. Encodings ............................................................................................................ 18  
   6.3. Examples ............................................................................................................. 19  
7. Requirement Class "Core" .............................................................................................. 20  
   7.1. Overview ............................................................................................................. 20  
   7.2. API landing page ................................................................................................. 22  
      7.2.1. Operation ....................................................................................................... 22  
      7.2.2. Response ....................................................................................................... 22  
      7.2.3. Error situations .............................................................................................. 23  
   7.3. API definition ....................................................................................................... 23  
      7.3.1. Operation ....................................................................................................... 23  
      7.3.2. Response ....................................................................................................... 24  
      7.3.3. Error situations .............................................................................................. 25  
   7.4. Declaration of conformance classes ....................................................................... 25  
      7.4.1. Operation ....................................................................................................... 25  
      7.4.2. Response ....................................................................................................... 25
OGC API - Features - Part 1: Core

Copyright notice

Copyright © 2019 Open Geospatial Consortium

To obtain additional rights of use, visit http://www.opengeospatial.org/legal/

Warning

This document is not an OGC Standard. This document is distributed for review and comment. This document is subject to change without notice and may not be referred to as an OGC Standard.

Recipients of this document are invited to submit, with their comments, notification of any relevant patent rights of which they are aware and to provide supporting documentation.

Document type: OGC® Standard
Document subtype: Interface
Document stage: Draft
Document language: English
License Agreement

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

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

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

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

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

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

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

The OGC API family of standards is organized by resource type. This standard 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.

If you are unfamiliar 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 is a separate standard. This part, 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 standard 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 standard 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 standard 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 to determine its capabilities and retrieve information about this distribution of the dataset, including the API definition and metadata about the feature collections provided by the API.

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 in the process to be 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 organisation names in the titles of ISO standards, standards from the "OGC API" series are published by ISO as "Geospatial API". That is, 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 standard defines the resources listed in Table 1. For an overview of the resources, see section 7.1 Overview.

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

<table>
<thead>
<tr>
<th>Resource</th>
<th>Path</th>
<th>HTTP method</th>
<th>Document reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Landing page</td>
<td>/</td>
<td>GET</td>
<td>7.2 API landing page</td>
</tr>
<tr>
<td>API definition</td>
<td>/api</td>
<td>GET</td>
<td>7.3 API definition</td>
</tr>
<tr>
<td>Conformance declaration</td>
<td>/conformance</td>
<td>GET</td>
<td>7.4 Declaration of conformance classes</td>
</tr>
<tr>
<td>Feature collections</td>
<td>/collections</td>
<td>GET</td>
<td>7.12 Feature collections</td>
</tr>
<tr>
<td>Feature collection</td>
<td>/collections/{collectionId}</td>
<td>GET</td>
<td>7.13 Feature collection</td>
</tr>
<tr>
<td>Features</td>
<td>/collections/{collectionId}/items</td>
<td>GET</td>
<td>7.14 Features</td>
</tr>
<tr>
<td>Feature</td>
<td>/collections/{collectionId}/items/{featureId}</td>
<td>GET</td>
<td>7.15 Feature</td>
</tr>
</tbody>
</table>

CAUTION ISSUE 236
Remove /api row, if the API definition may be located at any URI.

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

In the first approach, clients are implemented with knowledge about this standard and its resource types. The clients navigate the resources based on this knowledge and based on the responses provided by the API. The API definition may be used to determine details, e.g., on filter parameters, but this may not be necessary depending on the needs of the client. These are clients that are in general able to use multiple APIs as long as they implement OGC API 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 will study and use the API definition - typically an OpenAPI document - to understand the API and implement the code to interact with the API. This assumes familiarity with the API definition language and the related tooling, but it should not be necessary to study the OGC API standards.
This is a DRAFT version of the first part of the OGC API - Features standards. This draft is not complete and there are open issues that are still under discussion. These discussion topics are identified as annotations like this one with a link to the issue on GitHub and a brief summary of the issue. The current expectation is to have a release candidate by August 2019 after resolving all open issues in GitHub for this part.

ii. Keywords

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

ogcdoc, OGC document, OGC API, ISO, ISO/TC 211, geographic information, Geospatial API, Web Feature Service, WFS, feature, features, property, geographic information, spatial data, spatial things, dataset, distribution, API, OpenAPI, GeoJSON, GML, HTML, schema.org

iii. Preface

OGC Declaration

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

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

ISO Declaration

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.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 2.

The main task of technical committees is to prepare International Standards. Draft International Standards adopted by the technical committees are circulated to the member bodies for voting. Publication as an International Standard requires approval by at least 75 % of the member bodies casting a vote.

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.

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

- CubeWerx Inc.
- Hexagon
- interactive instruments GmbH
- Planet Labs

v. Submitters

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Affiliation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chris Holmes</td>
<td>Planet Labs</td>
</tr>
<tr>
<td>Clemens Portele (editor)</td>
<td>interactive instruments GmbH</td>
</tr>
<tr>
<td>Frédéric Houbie</td>
<td>Hexagon</td>
</tr>
<tr>
<td>Panagiotis (Peter) A. Vretanos (editor)</td>
<td>CubeWerx Inc.</td>
</tr>
</tbody>
</table>

CAUTION

ISSUE 224
Update submitters before publication.
Chapter 1. Scope

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

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

Query operations enable clients to retrieve features from the underlying data store based upon simple selection criteria, defined by the client.
Chapter 2. Conformance

This standard 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**.

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

Conformance with this standard 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.
Chapter 3. References

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

- W3C: HTML5, W3C Recommendation, [http://www.w3.org/TR/html5/](http://www.w3.org/TR/html5/)
Chapter 4. Terms and Definitions

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

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

4.1. dataset

collection of data, published or curated by a single agent, and available for access or download in one or more formats [DCAT]

NOTE The use of 'collection' in the definition from [DCAT] is broader than the use of the term collection in this specification. See the definition of 'feature collection'.

4.2. distribution

represents an accessible form of a dataset [DCAT]

EXAMPLE: a downloadable file, an RSS feed or an API.

4.3. feature

abstraction of real world phenomena [ISO 19101-1:2014]

NOTE If you are unfamiliar 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.

4.4. feature collection; collection

a set of features from a dataset

NOTE In this specification, 'collection' is used as a synonym for 'feature collection'. This is done to make, for example, URI path expressions shorter and easier to understand for those that are not geo-experts.

4.5. Web API

API using an architectural style that is founded on the technologies of the Web [DWBP]

NOTE Best Practice 24: Use Web Standards as the foundation of APIs in the W3C Data on the Web Best Practices provides more detail.
Chapter 5. Conventions

5.1. Identifiers

The normative provisions in this standard 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. UML model

UML diagrams are included in this standard to illustrate the conceptual model that underpins implementations of OGC API Features. The UML model is not normative. The UML profile used is specified in ISO 19103:2015.

Resources are modeled as UML interfaces.

5.3. Link relations

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

The following registered link relation types 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**: Indicates a URI that can be used to retrieve a service document.
  - API definitions are considered "service documents".

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 lists the specifications that the link's context conforms to.

• **data**: Indicates that the link's context is a distribution of a dataset that is an API and refers to a root resource of the dataset in the 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.

### 5.4. 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 and simply is a shorthand notation for "HTTP or HTTPS". In fact, most servers are expected to use HTTPS, not HTTP.

### 5.5. 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 Clause 2 of RFC 3986 for details.

### 5.6. API definition

#### 5.6.1. 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 standard 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 standard - or other parts of the Web Feature Service standard series - and will support additional operations, parameters, etc. that are specific to the API or the software tool used to implement the API.

#### 5.6.2. 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 is provided at path `/api`.

---

**CAUTION**

ISSUE 236

Change wording, if the API definition may be located at any URI.
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) 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.

### 5.6.3. References to OpenAPI components in normative statements

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

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

- If the server supports an XML encoding, `xml` properties may be added to the relevant OpenAPI schema components.
- The range of values of a parameter or property may be extended (additional values) or constrained (if a subset of all possible values are 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, like 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.

### 5.6.4. 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:

```json
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.

5.6.5. Reusable OpenAPI components

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

**CAUTION**

During the development phase, these components use a base URL of "https://raw.githubusercontent.com/opengeospatial/WFS_FES/master/", but during publication they will move to a base URL of "http://schemas.opengis.net/ogcapi/features/part1/1.0/".
Chapter 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 using 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:

• 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 are 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 as map images.

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

In the first approach, clients are implemented with knowledge about this standard and its resource types. The clients navigate the resources based on this knowledge and based on the responses provided by the API. The API definition may be used to determine details, e.g., on filter parameters, but this may not be necessary depending on the needs of the client. These are clients that are in general able to use multiple APIs as long as they implement OGC API 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 will study and use the API definition - typically an OpenAPI document - to understand the API and implement the code to interact with the API. This assumes familiarity with the API definition language and the related tooling, but it should not be necessary to study the OGC
6.2. Encodings

This standard does not mandate any encoding or format for representing features or feature collections. In addition to HTML as 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 the Web Feature Service standard 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 that is not based on WGS 84 longitude/latitude (e.g. an authoritative national reference system),
- 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. Conformance classes for GML are, therefore, included in this standard. 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 standard to provide guidance on using other encodings.

This part of the OGC API Features standards 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.
A server that implements this conformance class provides access to the features in a dataset. In other words, the API is a distribution of that dataset. A file download, for example, would be another distribution.

NOTE Other parts of this standard may define API extensions that support multiple datasets. The statement that the features are from “a dataset” is not meant to preclude such extensions. It just 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 All paths (e.g., /) are relative to the base URL of the distribution of the dataset. If the API covers other resources beyond those specified in this document, the landing page may also be, for example, a sub-resource of the base URL of the API.

The Landing page provides links to

- the API definition (path /api, link relation service),
- the Conformance declaration (path /conformance, link relation conformance), and
- the Collections (path /collections, link relation data).

CAUTION ISSUE 236 Change wording, if the API definition may be located at any URI and/or if service-desc and service-doc are used as link relations.

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 Conformance declaration states the requirements classes from standards or community specifications, identified by a URI, that the API conforms to. Clients can but are not required to use this information. Accessing the Conformance declaration using HTTP GET returns the list of URIs of
requirements 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. That is, each collection considered by this document is a feature collection. Other OGC API standards may add requirements for other types of collections.

**NOTE** 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 standard 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 available at path `/collections/{collectionId}`, too, 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 behavior 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 that 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 matches all features in the collection that are not associated with a location, too. The `datetime` parameter matches all features in the collection that are not associated with a time stamp or interval, too.
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

<table>
<thead>
<tr>
<th>Requirement 1</th>
<th>/req/core/root-op</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>A</strong></td>
<td>The server SHALL support the HTTP GET operation at the path /</td>
</tr>
</tbody>
</table>

#### 7.2.2. Response

<table>
<thead>
<tr>
<th>Requirement 2</th>
<th>/req/core/root-success</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>A</strong></td>
<td>A successful execution of the operation SHALL be reported as a response with a HTTP status code 200.</td>
</tr>
<tr>
<td><strong>B</strong></td>
<td>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:</td>
</tr>
<tr>
<td></td>
<td>• /api (relation type 'service')</td>
</tr>
<tr>
<td></td>
<td>• /conformance (relation type 'conformance')</td>
</tr>
<tr>
<td></td>
<td>• /collections (relation type 'data')</td>
</tr>
</tbody>
</table>

**CAUTION**

ISSUE 236

Change wording, if the API definition may be located at any URI.
Schema for the landing page

```json
type: object
required:
  - links
properties:
  title:
    type: string
  description:
    type: string
  links:
    type: array
    items:
      $ref: https://raw.githubusercontent.com/opengeospatial/WFS_FES/master/core/openapi/schemas/link.yaml
```

Example 2. Landing page response document

```json
{
  "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", "type": "application/vnd.oai.openapi+json;version=3.0",
      "title": "the API definition"
    },
    {
      "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": "Metadata 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.

<table>
<thead>
<tr>
<th>Requirement 3</th>
<th>/req/core/api-definition-op</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>The API SHALL support the HTTP GET operation at the path /api.</td>
</tr>
</tbody>
</table>

**CAUTION**  
ISSUE 236  
Change wording, if the API definition may be located at any URI.

### 7.3.2. Response

<table>
<thead>
<tr>
<th>Requirement 4</th>
<th>/req/core/api-definition-success</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>A successful execution of the operation SHALL be reported as a response with a HTTP status code 200.</td>
</tr>
<tr>
<td>B</td>
<td>The server SHALL return an API definition document.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Recommendation 1</th>
<th>/rec/core/api-definition-oas</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>If the API definition document uses the OpenAPI Specification 3.0, the document SHOULD conform to the OpenAPI Specification 3.0 requirements class.</td>
</tr>
</tbody>
</table>

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

The API definition document describes the API. In other words, there is no need to include the /api operation in the API definition itself.

**CAUTION**  
ISSUE 236  
Change wording, if the API definition may be located at any URI.

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 standard to access the data via the API.

Two example OpenAPI documents are included in Annex B. The documents differ in their representation of the feature collections. The first example uses a path parameter `collectionId` and the API definition only has a single path entry for all feature collections. The second example explicitly defines each feature collection in a separate path and without a path parameter - this allows to specify filter parameters or explicit feature schemas per feature collection. Both variants
are valid.

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 requirements classes it implements and conforms to.

<table>
<thead>
<tr>
<th>Requirement 5</th>
<th>/req/core/conformance-op</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>The server SHALL support the HTTP GET operation at the path <code>/conformance</code>.</td>
</tr>
</tbody>
</table>

7.4.2. Response

<table>
<thead>
<tr>
<th>Requirement 6</th>
<th>/req/core/conformance-success</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>A successful execution of the operation SHALL be reported as a response with a HTTP status code 200.</td>
</tr>
<tr>
<td>B</td>
<td>The content of that response SHALL be based upon the OpenAPI 3.0 schema <code>req-classes.yaml</code> and list all OGC API requirements classes that the server conforms to.</td>
</tr>
</tbody>
</table>

Schema for the list of requirements classes

```json

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

25
Example 3. Requirements class 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.

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

7.4.3. Error situations

See [HTTP status codes](#) for general guidance.

7.5. HTTP 1.1

<table>
<thead>
<tr>
<th>Requirement 7</th>
<th>/req/core/http</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>The server SHALL conform to <a href="#">HTTP 1.1</a>.</td>
</tr>
</tbody>
</table>

If the server supports HTTPS, the server SHALL also conform to [HTTP over TLS](#).

This includes the correct use of status codes, headers, etc.

<table>
<thead>
<tr>
<th>Recommendation 2</th>
<th>/rec/core/head</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>The server SHOULD support the HTTP 1.1 method HEAD for all resources that support the method GET.</td>
</tr>
</tbody>
</table>

Supporting the method HEAD in addition to GET can be useful for clients and is simple to implement.

Servers implementing [CORS](#) will implement the method OPTIONS, too.

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 2 are of particular relevance to implementors of this standard. 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 2 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 2. Typical HTTP status codes

<table>
<thead>
<tr>
<th>Status code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>A successful request.</td>
</tr>
<tr>
<td>304</td>
<td>An entity tag was provided in the request and the resource has not been changed since the previous request.</td>
</tr>
<tr>
<td>400</td>
<td>The server cannot or will not process the request due to an apparent client error. For example, a query parameter had an incorrect value.</td>
</tr>
<tr>
<td>401</td>
<td>The request requires user authentication. The response includes a WWW-Authenticate header field containing a challenge applicable to the requested resource.</td>
</tr>
<tr>
<td>403</td>
<td>The server understood the request, but is refusing to fulfill it. While status code 401 indicates missing or bad authentication, status code 403 indicates that authentication is not the issue, but the client is not authorised to perform the requested operation on the resource.</td>
</tr>
<tr>
<td>404</td>
<td>The requested resource does not exist on the server. For example, a path parameter had an incorrect value.</td>
</tr>
<tr>
<td>405</td>
<td>The request method is not supported. For example, a POST request was submitted, but the resource only supports GET requests.</td>
</tr>
<tr>
<td>406</td>
<td>The Accept header submitted in the request did not support any of the media types supported by the server for the requested resource.</td>
</tr>
<tr>
<td>500</td>
<td>An internal error occurred in the server.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Permission 1</th>
<th>/per/core/additional-status-codes</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Servers MAY support other capabilities of the HTTP protocol and, therefore, MAY return other status codes than those listed in Table 2, too.</td>
</tr>
</tbody>
</table>

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**

<table>
<thead>
<tr>
<th>Requirement 8</th>
<th>/req/core/query-param-unknown</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>A</strong></td>
<td>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.</td>
</tr>
</tbody>
</table>

If a server wants to support vendor specific parameters, these have to 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. That is, 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*

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

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

```json
{
  "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`.

<table>
<thead>
<tr>
<th>Requirement 9</th>
<th>/req/core/query-param-invalid</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>A</strong></td>
<td>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.</td>
</tr>
</tbody>
</table>

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 [HTTP/1.1 (RFC 2616)](https://tools.ietf.org/html/rfc2616).

<table>
<thead>
<tr>
<th>Recommendation 3</th>
<th>/rec/core/etag</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>The service SHOULD support entity tags and the associated headers as specified by HTTP/1.1.</td>
</tr>
</tbody>
</table>

### 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.

<table>
<thead>
<tr>
<th>Recommendation 4</th>
<th>/rec/core/cross-origin</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>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.</td>
</tr>
</tbody>
</table>

Two common mechanisms to support cross-origin requests are:

- Cross-origin resource sharing (CORS)
- 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)](https://www.opengis.net/isis/rfc0000.html) for a discussion.

<table>
<thead>
<tr>
<th>Recommendation 5</th>
<th>/rec/core/html</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>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 to support an HTML encoding.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Recommendation 6</th>
<th>/rec/core/geojson</th>
</tr>
</thead>
</table>
If the feature data can be represented for the intended use in GeoJSON, implementations SHOULD consider to support 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 how this is supported by the server.

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

Two common approaches are:

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

### 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.

For encodings that support string internationalization, the server SHOULD include information about the language for each string value that includes linguistic text.

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

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

7.11. Coordinate reference systems

As discussed in Chapter 9 of the W3C/OGC Spatial Data on the Web Best Practices document, how to express and share the location of 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.

<table>
<thead>
<tr>
<th>Requirement 10</th>
<th>/req/core/crs84</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Unless the client explicitly requests a different coordinate reference system, all spatial geometries SHALL be in the coordinate reference system <a href="http://www.opengis.net/def/crs/OGC/1.3/CRS84">http://www.opengis.net/def/crs/OGC/1.3/CRS84</a> (WGS 84 longitude/latitude).</td>
</tr>
</tbody>
</table>

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. 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.

7.12. Link headers

<table>
<thead>
<tr>
<th>Recommendation 8</th>
<th>/rec/core/link-header</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>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.</td>
</tr>
</tbody>
</table>

7.13. Feature collections

7.13.1. Operation
### Requirement 11

<table>
<thead>
<tr>
<th>/req/core/fc-md-op</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
</tr>
</tbody>
</table>

### 7.13.2. Response

#### Requirement 12

<table>
<thead>
<tr>
<th>/req/core/fc-md-success</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
</tr>
<tr>
<td>B</td>
</tr>
</tbody>
</table>

**Schema for the collections resource**

```yaml

type: object
required:
- links
- collections
properties:
links:
type: array
items:
$ref: https://raw.githubusercontent.com/opengeospatial/WFS_FES/master/core/openapi/schemas/link.yaml
collections:
type: array
items:
$ref: https://raw.githubusercontent.com/opengeospatial/WFS_FES/master/core/openapi/schemas/collection.yaml
```

#### Requirement 13

<table>
<thead>
<tr>
<th>/req/core/fc-md-links</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

---

32
B All links SHALL include the **rel** and **type** link parameters.

**Recommendation 9**  
/\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, etc. |

**Recommendation 10**  
/\rec/core/fc-md-license

| A | For each feature collection in this distribution of the dataset, the **links** property of the collection SHOULD include an item for each supported encoding with a link to the collection resource (relation: **license**). |
| B | Alternatively, if all data shared via the API is available under the same license, the link MAY instead be added to the top-level **links** property of the response. |
| C | Multiple links to the license in different content types MAY be provided. At least a link to content type **text/html** or **text/plain** SHOULD be provided. |

**CAUTION**  
**ISSUE 56**  
Lack of DescribeFeatureType request

**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**. |
A To support servers with many collections, servers MAY limit the number of items in the property `collections`.

This document does not specify mechanisms 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.

### Recommendation 11

/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 be support multiple bounding boxes or time intervals.

### Permission 3

/per/core/fc-md-extent-extensions
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.

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`).

Extension to the Core MAY add additional reference systems to the *extent* object.

---

**Schema for a feature collection**

```json

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:
      type: object
      required:
        - href
      properties:
        href:
          type: string
          example: http://data.example.com/buildings/123
        rel:
          type: string
          example: alternate
        type:
          type: string
          example: application/geo+json
        hreflang:
          type: string
          example: en
        title:
          type: string
```
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: >-
        West, south, east, north edges of the bounding box. The coordinates
        are in the coordinate reference system specified in 'crs'. By default
        this is WGS 84 longitude/latitude.
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'
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 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:
            - 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 reference system. Extensions may support additional temporal reference systems and add additional enum values.
      type: string
      enum:
        - 'http://www.opengis.net/def/uom/ISO-8601/0/Gregorian'
        default: 'http://www.opengis.net/def/uom/ISO-8601/0/Gregorian'
      itemsType:
        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;
The first item is the default coordinate reference system

```
- http://www.opengis.net/def/crs/OGC/1.3/CRS84
```

**NOTE**

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

*Example 4. Feature collections response document*

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").

Representations of the 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".

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

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

See HTTP status codes for general guidance.

7.14. Feature collection

7.14.1. Operation
### 7.14.2. Response

<table>
<thead>
<tr>
<th>Requirement 18</th>
<th>/req/core/sfc-md-success</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>A</strong></td>
<td>A successful execution of the operation SHALL be reported as a response with a HTTP status code <strong>200</strong>.</td>
</tr>
<tr>
<td><strong>B</strong></td>
<td>The content of that response SHALL be consistent with the content for this feature collection in the /collections response. That is, the values for id, title, description and extent SHALL be identical.</td>
</tr>
</tbody>
</table>

### 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 2).

### 7.15. Features

#### 7.15.1. Operation

<table>
<thead>
<tr>
<th>Requirement 19</th>
<th>/req/core/fc-op</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>A</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>B</strong></td>
<td>The parameter <code>collectionId</code> is each id property in the feature collections response (JSONPath: $.collections[*].id).</td>
</tr>
</tbody>
</table>

#### 7.15.2. Parameter limit
<table>
<thead>
<tr>
<th>Requirement 20</th>
<th>/req/core/fc-limit-definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>The operation SHALL support a parameter limit with the following characteristics (using an OpenAPI Specification 3.0 fragment):</td>
</tr>
</tbody>
</table>
|                | name: limit  
|                | in: query  
|                | required: false  
|                | schema:  
|                |   type: integer  
|                |   minimum: 1  
|                |   maximum: 10000  
|                |   default: 10  
|                |   style: form  
|                |   explode: false |

<table>
<thead>
<tr>
<th>Permission 4</th>
<th>/per/core/fc-limit-default-maximum</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>The values for maximum and default in requirement /req/core/fc-limit-definition are only examples and MAY be changed.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Requirement 21</th>
<th>/req/core/fc-limit-response-1</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>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.</td>
</tr>
<tr>
<td>B</td>
<td>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.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Permission 5</th>
<th>/per/core/fc-limit-response-2</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>The server MAY return less features than requested (but not more).</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Requirement 22</th>
<th>/req/core/fc-bbox-definition</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>A</strong></td>
<td>The operation SHALL support a parameter <strong>bbox</strong> with the following characteristics (using an OpenAPI Specification 3.0 fragment):</td>
</tr>
<tr>
<td></td>
<td>name: bbox</td>
</tr>
<tr>
<td></td>
<td>in: query</td>
</tr>
<tr>
<td></td>
<td>required: false</td>
</tr>
<tr>
<td></td>
<td>schema:</td>
</tr>
<tr>
<td></td>
<td>type: array</td>
</tr>
<tr>
<td></td>
<td>minItems: 4</td>
</tr>
<tr>
<td></td>
<td>maxItems: 6</td>
</tr>
<tr>
<td></td>
<td>items:</td>
</tr>
<tr>
<td></td>
<td>type: number</td>
</tr>
<tr>
<td></td>
<td>style: form</td>
</tr>
<tr>
<td></td>
<td>explode: false</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Requirement 23</th>
<th>/req/core/fc-bbox-response</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>A</strong></td>
<td>Only features that have a spatial geometry that intersects the bounding box SHALL be part of the result set, if the <strong>bbox</strong> parameter is provided.</td>
</tr>
<tr>
<td><strong>B</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>C</strong></td>
<td>The <strong>bbox</strong> parameter SHALL match all features in the collection that are not associated with a spatial geometry, too.</td>
</tr>
</tbody>
</table>
The bounding box is provided as four or six numbers, depending on whether the coordinate reference system includes a vertical axis (height or depth):

- Lower left corner, coordinate axis 1
- Lower left corner, coordinate axis 2
- Lower left corner, coordinate axis 3 (optional)
- Upper right corner, coordinate axis 1
- Upper right corner, coordinate axis 2
- Upper right corner, coordinate axis 3 (optional)

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`.

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).

This standard 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 5. 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.

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

<table>
<thead>
<tr>
<th>Requirement 24</th>
<th>/req/core/fc-time-definition</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>A</strong></td>
<td>The operation SHALL support a parameter <code>datetime</code> with the following characteristics (using an OpenAPI Specification 3.0 fragment):</td>
</tr>
</tbody>
</table>
|   | ```
|     name: datetime
|     in: query
|     required: false
|     schema:
|       type: string
|       style: form
|       explode: false
| ``` |

<table>
<thead>
<tr>
<th>Requirement 25</th>
<th>/req/core/fc-time-response</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>A</strong></td>
<td>Only features that have a temporal geometry that intersects the temporal information in the <code>datetime</code> parameter SHALL be part of the result set, if the parameter is provided.</td>
</tr>
<tr>
<td><strong>B</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>C</strong></td>
<td>The <code>datetime</code> parameter SHALL match all features in the collection that are not associated with a temporal geometry, too.</td>
</tr>
<tr>
<td><strong>D</strong></td>
<td>Temporal geometries are either a date-time value or a time interval. The parameter value SHALL conform to the following syntax (using ABNF):</td>
</tr>
</tbody>
</table>
|   | ```
|     interval-closed     = date-time "/" date-time
|     interval-open-start = "../" date-time
|     interval-open-end   = date-time "/.."
|     interval            = interval-closed / interval-open-start / interval-open-end
|     datetime            = date-time / interval
| ``` |
| **E** | The syntax of `date-time` is specified by [RFC 3339, 5.6](https://tools.ietf.org/html/rfc3339). |
Open ranges in time intervals at the start or end are supported using a double-dot (..).

"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.

**Example 6. A date-time**

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

```plaintext
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 7. Intervals**

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

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

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

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

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

```plaintext
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 | /rec/core/fc-filters |
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 8. 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 9. An additional parameter to filter buildings based on their name

```json
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

<table>
<thead>
<tr>
<th>Requirement 26</th>
<th>/req/core/fc-response</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>A successful execution of the operation SHALL be reported as a response with a HTTP status code 200.</td>
</tr>
<tr>
<td>B</td>
<td>The response SHALL only include features selected by the request.</td>
</tr>
</tbody>
</table>

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 don’t specify a limit, you will get 0 to 10 (default) and if there are more a next link;
• If you ask for 50000, you might 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 might get up to 10000 additional features and if there are more a next link.

<table>
<thead>
<tr>
<th>Requirement 27</th>
<th>/req/core/fc-links</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>A 200-response SHALL include the following links:</td>
</tr>
<tr>
<td></td>
<td>• a link to this response document (relation: self),</td>
</tr>
<tr>
<td></td>
<td>• a link to the response document in every other media type supported by the service (relation: alternate).</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Recommendation 13</th>
<th>/rec/core/fc-next-1</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>A 200-response SHOULD include a link to the next “page” (relation: next), if more features have been selected than returned in the response.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Recommendation 14</th>
<th>/rec/core/fc-next-2</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Dereferencing a next link SHOULD return additional features from the set of selected features that have not yet been returned.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Recommendation 15</th>
<th>/rec/core/fc-next-3</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>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.</td>
</tr>
</tbody>
</table>

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** Additional conformance classes for safe paging or an index parameter may be added in extensions to this specification.

<table>
<thead>
<tr>
<th>Permission 6</th>
<th>/per/core/fc-prev</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>A response to a next link MAY include a prev link to the resource that included the next link.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Requirement 28</th>
<th>/req/core/fc-rel-type</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>All links SHALL include the rel and type link parameters.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Requirement 29</th>
<th>/req/core/fc-timeStamp</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>If a property timeStamp is included in the response, the value SHALL be set to the time stamp when the response was generated.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Requirement 30</th>
<th>/req/core/fc-numberMatched</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>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.</td>
</tr>
<tr>
<td>B</td>
<td>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.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Requirement 31</th>
<th>/req/core/fc-numberReturned</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>If a property numberReturned is included in the response, the value SHALL be identical to the number of features in the response.</td>
</tr>
</tbody>
</table>
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
The representation of the links and the other properties in the payload depends on the encoding of the feature collection.

Example 10. Links

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 startIndex to implement next links - and the optional prev links):

```plaintext
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?startIndex=10>; rel="next"; type="application/geo+json"
```

Following the next link could return:

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

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

```plaintext
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"
```

Following the next link could return:
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

<table>
<thead>
<tr>
<th>Requirement 32</th>
<th>/req/core/f-op</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>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}.</td>
</tr>
<tr>
<td>B</td>
<td>The parameter collectionId is each id property in the feature collections response (JSONPath: $.collections[*].id). featureId is a local identifier of the feature.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Permission 7</th>
<th>/per/core/f-id</th>
</tr>
</thead>
</table>
The Core requirements class only requires that the feature URI is unique. Implementations MAY apply stricter rules and, for example, use unique id values per dataset or collection.

### 7.16.2. Response

<table>
<thead>
<tr>
<th>Requirement 33</th>
<th>/req/core/f-success</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>A successful execution of the operation SHALL be reported as a response with a HTTP status code 200.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Requirement 34</th>
<th>/req/core/f-links</th>
</tr>
</thead>
</table>
| 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 11. Links**

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

```plaintext
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.
Chapter 8. Requirements classes for encodings

8.1. Overview

This clause specifies four pre-defined requirements classes for encodings to be used by a 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. Requirement 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, that is the search engines of the Web;
- The data can not be viewed directly in a browser - additional tools are required to view the data.

Therefore, sharing data on the Web should include publication in HTML. To be consistent with the Web, 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 [SDWBP]. This standard therefore recommends supporting HTML as an encoding.

<table>
<thead>
<tr>
<th>Requirements Class</th>
<th><a href="http://www.opengis.net/spec/ogcapi-features-1/1.0/req/html">http://www.opengis.net/spec/ogcapi-features-1/1.0/req/html</a></th>
</tr>
</thead>
<tbody>
<tr>
<td>Target type</td>
<td>Web API</td>
</tr>
<tr>
<td>Dependency</td>
<td>Conformance Class &quot;Core&quot;</td>
</tr>
<tr>
<td>Dependency</td>
<td>HTML5</td>
</tr>
<tr>
<td>Dependency</td>
<td>Schema.org</td>
</tr>
</tbody>
</table>
**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 16**  
/rec/html/schema-org  

A  
A **200**-response with the media type **text/html**, SHOULD include **Schema.org** annotations.

### 8.3. Requirement 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

<table>
<thead>
<tr>
<th>Target type</th>
<th>Web API</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dependency</td>
<td>Conformance Class &quot;Core&quot;</td>
</tr>
<tr>
<td>Dependency</td>
<td>GeoJSON</td>
</tr>
</tbody>
</table>

**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.
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 a 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 12. A GeoJSON FeatureCollection Object response**

In the example below, only the first and tenth feature is shown. Coordinates are not shown.
Example 13. A GeoJSON Feature 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/items",
            "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. Requirement 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 standard specifies requirement 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 supported by most tools. In addition, the Level 0 profile is limited to features that can be stored in a tabular data structure.

<table>
<thead>
<tr>
<th>Requirements Class</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://www.opengis.net/spec/ogcapi-features-1/1.0/req/gmlsf0">http://www.opengis.net/spec/ogcapi-features-1/1.0/req/gmlsf0</a></td>
</tr>
</tbody>
</table>

Target type  Web API
**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 3 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. |

**Table 3. Media types and XML elements for each resource**

<table>
<thead>
<tr>
<th>Resource</th>
<th>Path</th>
<th>XML root element</th>
</tr>
</thead>
<tbody>
<tr>
<td>Landing page</td>
<td>/</td>
<td>core:LandingPage</td>
</tr>
<tr>
<td>Conformance declaration</td>
<td>/conformance</td>
<td>core:ConformsTo</td>
</tr>
<tr>
<td>Feature collections</td>
<td>/collections</td>
<td>core:Collections</td>
</tr>
<tr>
<td>Feature collection</td>
<td>/collections/{collectionId}</td>
<td>core:Collections, with just one entry for the collection collectionId</td>
</tr>
<tr>
<td>Features</td>
<td>/collections/{collectionId}/items</td>
<td>core:FeatureCollection</td>
</tr>
<tr>
<td>Feature</td>
<td>/collections/{collectionId}/items/{featureId}</td>
<td>substitutable for gml:AbstractFeature</td>
</tr>
</tbody>
</table>

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

- **core**: http://www.opengis.net/ogcapi-features-core/1.0
- **gml**: http://www.opengis.net/gml/3.2
- **atom**: http://www.w3.org/2005/Atom
The API definition resource at path `/api` is not included in Table 3. This requirements class does not prescribe any API definition approach.

**CAUTION**

**ISSUE 236**
Remove paragraph, if the API definition may be located at any URI.

The mapping of the content from the responses specified in the Core requirements class to the XML is straightforward. All links are encoded using `atom:link` elements except in GML features where simple Xlinks are used.

Annex C has example responses in XML.

**NOTE**

The `core:FeatureCollection` element deliberately goes beyond the permitted content specified in the GML Simple Features Profile, section 8.4.2. This is necessary to support the hypermedia controls and other relevant content for a Web Feature Service API.

8.5. Requirement Class "Geography Markup Language (GML), Simple Features Profile, Level 2"

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

<table>
<thead>
<tr>
<th>Requirements Class</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://www.opengis.net/spec/ogcapi-features-1/1.0/req/gmlsf2">http://www.opengis.net/spec/ogcapi-features-1/1.0/req/gmlsf2</a></td>
</tr>
<tr>
<td>Target type</td>
</tr>
<tr>
<td>Dependency</td>
</tr>
<tr>
<td>Dependency</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Requirement 41</th>
<th>/req/gmlsf2/definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>200-responses of the server SHALL support the following media types:</td>
</tr>
<tr>
<td></td>
<td>• <code>application/gml+xml; version=3.2; profile=http://www.opengis.net/def/profile/ogc/2.0/gml-sf2</code> for resources that include feature content,</td>
</tr>
<tr>
<td></td>
<td>• <code>application/xml</code> for all other resources.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Requirement 42</th>
<th>/req/gmlsf2/content</th>
</tr>
</thead>
</table>
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".
Chapter 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

<table>
<thead>
<tr>
<th>Requirements Class</th>
<th><a href="http://www.opengis.net/spec/ogcapi-features-1/1.0/req/oas30">http://www.opengis.net/spec/ogcapi-features-1/1.0/req/oas30</a></th>
</tr>
</thead>
<tbody>
<tr>
<td>Target type</td>
<td>Web API</td>
</tr>
<tr>
<td>Dependency</td>
<td>Conformance Class “Core”</td>
</tr>
<tr>
<td>Dependency</td>
<td>OpenAPI Specification 3.0.2</td>
</tr>
</tbody>
</table>

#### Requirement 43 /req/oas30/oas-definition-1

| A | The server SHALL provide an OpenAPI definition in JSON at the path `/api` 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`. |

**CAUTION**

**ISSUE 236**

Change wording, if the API definition may be located at any URI.

**CAUTION**

**ISSUE 117**

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

#### Requirement 44 /req/oas30/oas-definition-2

| A | The JSON representation SHALL conform to the OpenAPI Specification, version 3.0. |

Two example OpenAPI documents are included in Annex B. The documents differ in their representation of the feature collections. The first example uses a path parameter `collectionId` and the API definition only has a single path entry for all feature collections. The second example explicitly defines each feature collection in a separate path and without a path parameter - this allows to specify filter parameters or explicit feature schemas per feature collection. Both variants are valid.

#### Requirement 45 /req/oas30/oas-impl

| A | The server SHALL implement all capabilities specified in the OpenAPI definition. |
9.2. Complete definition

<table>
<thead>
<tr>
<th>Requirement 46</th>
<th>/req/oas30/completeness</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>A</strong></td>
<td>The OpenAPI definition SHALL specify for each operation all HTTP Status Codes and Response Objects that the server uses in responses.</td>
</tr>
<tr>
<td><strong>B</strong></td>
<td>This includes the successful execution of an operation as well as all error situations that originate from the server.</td>
</tr>
</tbody>
</table>

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.

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

9.3. Exceptions

<table>
<thead>
<tr>
<th>Requirement 47</th>
<th>/req/oas30/exceptions-codes</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>A</strong></td>
<td>For error situations that originate from the server, the API definition SHALL cover all applicable HTTP Status Codes.</td>
</tr>
</tbody>
</table>

Example 14. An exception response object definition

```json
description: An error occurred.
content:
  application/json:
    schema:
      $ref: https://raw.githubusercontent.com/opengeospatial/WFS_FES/master/core/openapi/schemas/exception.yaml
  text/html:
    schema:
      type: string
```

9.4. Security

| Requirement 48 | /req/oas30/security |
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 RFC6749, and
- OpenID Connect Discovery.

9.5. Features

<table>
<thead>
<tr>
<th>Recommendation</th>
<th>/rec/oas30/f-key-properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>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).</td>
</tr>
</tbody>
</table>
Chapter 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 vnd.oai.openapi+json;version=3.0 (JSON) or application/vnd.oai.openapi;version=3.0 (YAML).

**CAUTION**  ISSUE 117
The OpenAPI media type has not been registered yet with IANA and may change.
Chapter 11. Section 11: Security Considerations

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 section 15 of IETF RFC 2616.

The following sections 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.

### 11.1. Multiple Access Routes

APIs deliver a representation of a resource. OGC APIs can deliver multiple representations (formats) of the same resource. An attacker may 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 must be protected to the same level across all access routes.

11.2. 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 though one server may be accessible through another.

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

11.3. Path Manipulation on GET

RFC-2626 section 15.2 states “If an HTTP server translates HTTP URIs directly into file system calls, the server MUST 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.

11.4. 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 though 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: Abstract Test Suite (Normative)

**CAUTION**

**ISSUE 112**

The ATS is outdated and confusing in its current state. The existing text has been removed from the published draft (the adoc file is still in the GitHub repository). Annex A will be updated and completed before submission.
Annex B: OpenAPI definition example (Informative)

B.1. Overview

This annex includes two complete examples of an OpenAPI definition for a Web API implementing OGC API Features (the Core, HTML, GeoJSON and OpenAPI 3.0 conformance classes).

The first example (Generic OpenAPI definition) is a generic example that uses path parameters to describe all feature collections and all features. This OpenAPI definition does not provide any details on the collections or the feature content. This information is only available from the feature collection metadata.

The second example (OpenAPI definition with details on the collection and its features) does not use a path parameter for the collections and explicitly provides information about the feature collection 'buildings' (paths /collections/buildings etc.), the schema of the building features (schema buildingGeoJSON) and a filter parameter for building features (parameter function).

B.2. Generic OpenAPI definition

```yaml
openapi: 3.0.1
info:
  title: A sample API conforming to the OGC Web Feature Service standard
  version: 0.0.1
  description: This is a sample OpenAPI definition that conforms to the OGC Web Feature Service specification (conformance classes: "Core", "GeoJSON", "HTML" and "OpenAPI 3.0").
  contact:
    name: Acme Corporation
    email: info@example.org
    url: 'http://example.org/
  license:
    name: CC-BY 4.0 license
    url: 'https://creativecommons.org/licenses/by/4.0/
servers:
- url: 'https://dev.example.org/
  description: Development server
- url: 'https://data.example.org/
  description: Production server
paths:
  '/':
    get:
      summary: landing page of this API
      description: The landing page provides links to the API definition, the Conformance statements and the metadata about the feature data in this dataset.
```

operationId: getLandingPage
tags:
  - Capabilities
responses:
  '200':
    description: links to the API capabilities
    content:
      application/json:
        schema:
          $ref: '#/components/schemas/root'
      text/html:
        schema:
          type: string
'/conformance':
  get:
    summary: information about standards that this API conforms to
    description: list all requirements classes specified in a standard (e.g., OGC API - Features - Part 1: Core) that the server conforms to
    operationId: getRequirementsClasses
tags:
      - Capabilities
        responses:
          '200':
            description: the URIs of all requirements classes supported by the server
            content:
              application/json:
                schema:
                  $ref: '#/components/schemas/req-classes'
          default:
            description: An error occurred.
            content:
              application/json:
                schema:
                  $ref: '#/components/schemas/exception'
'/collections':
  get:
    summary: describe the feature collections in the dataset
    operationId: describeCollections
tags:
      - Capabilities
        responses:
          '200':
            description: Metadata about the feature collections shared by this API.
            content:
              application/json:
                schema:
                  $ref: '#/components/schemas/content'
              text/html:
                schema:
                  type: string
default:
  description: An error occurred.
content:
  application/json:
    schema:
      $ref: '#/components/schemas/exception'
  text/html:
    schema:
      type: string
'/collections/{collectionId}':
  get:
    summary: 'describe the {collectionId} feature collection'
    operationId: describeCollection
    tags:
      - Capabilities
    parameters:
      - $ref: '#/components/parameters/collectionId'
    responses:
      '200':
        description: 'Metadata about the {collectionId} collection shared by this API.'
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/collectionInfo'
          text/html:
            schema:
              type: string
default:
  description: An error occurred.
content:
  application/json:
    schema:
      $ref: '#/components/schemas/exception'
  text/html:
    schema:
      type: string
'/collections/{collectionId}/items':
  get:
    summary: 'retrieve features of feature collection {collectionId}'
    description: >-
      Every feature in a dataset belongs to a collection. A dataset may consist of multiple feature collections. A feature collection is often a collection of features of a similar type, based on a common schema.
      Use content negotiation to request HTML or GeoJSON.
    operationId: getFeatures
    tags:
      - Features
    parameters:
      - $ref: '#/components/parameters/collectionId'
$ref: '#/components/parameters/limit'
$ref: '#/components/parameters/bbox'
$ref: '#/components/parameters/datetime'

responses:
  '200':
    description: 
      Information about the feature collection plus the first features
      matching the selection parameters.
    content:
      application/geo+json:
        schema:
          $ref: '#/components/schemas/featureCollectionGeoJSON'
      text/html:
        schema:
          type: string
    default:
      description: An error occurred.
      content:
        application/json:
          schema:
            $ref: '#/components/schemas/exception'
        text/html:
          schema:
            type: string

'/collections/{collectionId}/items/{featureId}':
  get:
    summary: retrieve a feature; use content negotiation to request HTML or GeoJSON
    operationId: getFeature
    tags:
      - Features
    parameters:
      - $ref: '#/components/parameters/collectionId'
      - $ref: '#/components/parameters/featureId'
    responses:
      '200':
        description: A feature.
        content:
          application/geo+json:
            schema:
              $ref: '#/components/schemas/featureGeoJSON'
          text/html:
            schema:
              type: string
      default:
        description: An error occurred.
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/exception'
          text/html:
            schema:
components:
parameters:
limit:
  name: limit
  in: query
  description: |
  The optional limit parameter limits the number of items that are presented in the response document.
  Only items are counted that are on the first level of the collection in the response document. Nested objects contained within the explicitly requested items shall not be counted.
  * Minimum = 1
  * Maximum = 10000
  * Default = 10
  required: false
  schema:
    type: integer
    minimum: 1
    maximum: 10000
    default: 10
    style: form
    explode: false
bbox:
  name: bbox
  in: query
  description: >
  Only features that have a geometry that intersects the bounding box are selected.
  The bounding box is provided as four or six numbers, depending on whether the coordinate reference system includes a vertical axis (elevation or depth):
  * Lower left corner, coordinate axis 1
  * Lower left corner, coordinate axis 2
  * Lower left corner, coordinate axis 3 (optional)
  * Upper right corner, coordinate axis 1
  * Upper right corner, coordinate axis 2
  * Upper right corner, coordinate axis 3 (optional)

  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 the parameter `bbox-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 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.

required: false

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

datetime:
  name: datetime
  in: query
  description: >-
    Either a date-time or an interval, open or closed. Date and time expressions adhere to RFC 3339. Open intervals are expressed using double-dots.

Examples:

* A date-time: "2018-02-12T23:20:50Z"
* A closed interval: "2018-02-12T00:00:00Z/2018-03-18T12:31:12Z"
* Open intervals: "2018-02-12T00:00:00Z/.." or "../2018-03-18T12:31:12Z"

Only features that have a temporal property that intersects the value of 'datetime' are selected.

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.

required: false

schema:
  type: string
  style: form
  explode: false

collectionId:
  name: collectionId
  in: path
  required: true
  description: Identifier of a specific collection

schema:
  type: string

featureId:
  name: featureId
  in: path
  description: Local identifier of a specific feature
  required: true

schema:
  type: string

schemas:
exception:
  type: object
  required:
    - code
  properties:
    code:
      type: string
    description:
      type: string
root:
  type: object
  required:
    - links
  properties:
    links:
      type: array
      items:
        $ref: '#/components/schemas/link'
  example:
    - href: 'http://data.example.org/'
      rel: self
      type: application/json
      title: this document
    - href: 'http://data.example.org/api'
      rel: service
      type: application/vnd.oai.openapi+json;version=3.0
      title: the API definition
    - href: 'http://data.example.org/conformance'
      rel: conformance
      type: application/json
      title: OGC API Features conformance classes implemented by this server
    - href: 'http://data.example.org/collections'
      rel: data
      type: application/json
      title: Metadata about the feature collections
req-classes:
  type: object
  required:
    - conformsTo
  properties:
    conformsTo:
      type: array
      items:
        type: string
      example:
        - 'http://www.opengis.net/spec/ogcapi-features-1/1.0/req/core'
        - 'http://www.opengis.net/spec/ogcapi-features-1/1.0/req/oas30'
        - 'http://www.opengis.net/spec/ogcapi-features-1/1.0/req/html'
        - 'http://www.opengis.net/spec/ogcapi-features-1/1.0/req/geojson'
link:
  type: object
required:
  - href
properties:
  href:
    type: string
  rel:
    type: string
    example: prev
  type:
    type: string
    example: application/geo+json
  hreflang:
    type: string
    example: en
  title:
    type: string
    example: previous page
content:
  type: object
  required:
  - links
  - collections
properties:
  links:
    type: array
    items:
      $ref: '#/components/schemas/link'
  example:
    - 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/foobar.xsd'
      rel: describedBy
      type: application/xml
      title: XML schema for Acme Corporation data
  collections:
    type: array
    items:
      $ref: '#/components/schemas/collectionInfo'
collectionInfo:
  type: object
  required:
  - name
  - links
  properties:
  name:
description: 'identifier of the collection used, for example, in URIs'
type: string
eexample: buildings
title:
  description: 'human readable title of the collection'
type: string
eexample: Buildings
description: 'a description of the features in the collection'
type: string
eexample: Buildings in the city of Bonn.
links:
type: array
items:
  $ref: '#/components/schemas/link'
eexample:
  - href: 'http://data.example.org/collections/buildings/items'
    rel: item
    type: application/geo+json
    title: Buildings
  - href: 'http://example.org/concepts/building.html'
    rel: describedBy
    type: text/html
    title: Feature catalogue for buildings
extent:
  $ref: '#/components/schemas/extent'
crs:
description: 'The coordinate reference systems in which geometries may be retrieved. Coordinate reference systems are identified by a URI. The first coordinate reference system is the coordinate reference system that is used by default. This is always "http://www.opengis.net/def/crs/OGC/1.3/CRS84", i.e. WGS 84 longitude/latitude.'
type: array
items:
  type: string
default:
  - 'http://www.opengis.net/def/crs/OGC/1.3/CRS84'
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: >-
      West, south, east, north edges of the bounding box. The coordinates
      are in the coordinate reference system specified in `crs`. By
default
      this is WGS 84 longitude/latitude.
    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'
      default: 'http://www.opengis.net/def/crs/OGC/1.3/CRS84'
  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 intervall.
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.

```json
    type: array
    minItems: 1
    items:
        description: >-
            Begin and end times of the time interval. The timestamps are in the 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:
            - null
    trs:
        description: >-
            Coordinate reference system of the coordinates in the temporal extent (property `interval`). The default reference system is the Gregorian calendar.
        type: string
        enum:
            - 'http://www.opengis.net/def/uom/ISO-8601/0/Gregorian'
            default: 'http://www.opengis.net/def/uom/ISO-8601/0/Gregorian'
```

In the Core this is the only supported temporal reference system. Extensions may support additional temporal reference systems and add additional enum values.

```json
    featureCollectionGeoJSON:
        type: object
        required:
            - type
            - features
        properties:
            type:
                type: string
            enum:
                - FeatureCollection
        features:
            type: array
            items:
                $ref: '#/components/schemas/featureGeoJSON'
```

```json
    links:
        type: array
```
items:
  $ref: '#/components/schemas/link'
timeStamp:
  type: string
  format: date-time
numberMatched:
  type: integer
  minimum: 0
numberReturned:
  type: integer
  minimum: 0
featureGeoJSON:
  type: object
  required:
  - type
  - geometry
  - properties
properties:
  type:
    type: string
    enum:
    - Feature
gometry:
  $ref: '#/components/schemas/geometryGeoJSON'
properties:
  type: object
  nullable: true
id:
  oneOf:
  - type: string
  - type: integer
geometryGeoJSON:
  type: object
  required:
  - type
properties:
  type:
    type: string
    enum:
    - Point
    - MultiPoint
    - LineString
    - MultiLineString
    - Polygon
    - MultiPolygon
    - GeometryCollection
tags:
  - name: Capabilities
description: >-
  Essential characteristics of this API including information about the data.
B.3. OpenAPI definition with details on the collection and its features

```
openapi: 3.0.1
info:
  title: A sample API conforming to the OGC Web Feature Service standard
  version: 0.0.1
  description: This is a sample OpenAPI definition that conforms to the OGC Web Feature Service specification (conformance classes: "Core", "GeoJSON", "HTML" and "OpenAPI 3.0.")

The API provides access to a single feature collection: buildings. 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.

contact:
  name: Acme Corporation
  email: info@example.org
  url: 'http://example.org/

license:
  name: CC-BY 4.0 license
  url: 'https://creativecommons.org/licenses/by/4.0/

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

paths:
  '/':
    get:
      summary: landing page of this API
      description: The landing page provides links to the API definition, the Conformance statements and the metadata about the feature data in this dataset.
      operationId: getLandingPage
      tags:
        - Capabilities
      responses:
        '200':
          description: links to the API capabilities
          content:
            application/json:
```

- name: Features
description: Access to data (features).

'/conformance':
get:
  summary: information about standards that this API conforms to
  description: list all requirements classes specified in a standard (e.g., OGC API - Features - Part 1: Core) that the server conforms to
  operationId: getRequirementsClasses
  tags:
  - Capabilities
  responses:
    '200':
      description: the URIs of all requirements classes supported by the server
      content:
        application/json:
          schema:
            $ref: '#/components/schemas/req-classes'
    default:
      description: An error occurred.
      content:
        application/json:
          schema:
            $ref: '#/components/schemas/exception'
'/collections':
get:
  summary: describe the feature collections in the dataset
  operationId: describeCollections
  tags:
  - Capabilities
  responses:
    '200':
      description: Metadata about the feature collections shared by this API.
      content:
        application/json:
          schema:
            $ref: '#/components/schemas/content'
    default:
      description: An error occurred.
      content:
        application/json:
          schema:
            $ref: '#/components/schemas/exception'
'/collections/buildings':
  get:
    summary: 'describe the buildings feature collection'
    operationId: describeCollection
    tags:
    - Capabilities
    responses:
      '200':
        description: 'Metadata about the buildings collection shared by this API.'
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/collectionInfo'
          text/html:
            schema:
              type: string
              default:
                description: An error occurred.
                content:
                  application/json:
                    schema:
                      $ref: '#/components/schemas/exception'
                  text/html:
                    schema:
                      type: string
'/collections/buildings/items':
  get:
    summary: 'retrieve features of buildings feature collection'
    description: 'Every feature in a dataset belongs to a collection. A dataset may consist of multiple feature collections. A feature collection is often a collection of features of a similar type, based on a common schema. Use content negotiation to request HTML or GeoJSON.'
    operationId: getFeatures
    tags:
    - Features
    parameters:
    - $ref: '#/components/parameters/limit'
    - $ref: '#/components/parameters/bbox'
    - $ref: '#/components/parameters/datetime'
    - $ref: '#/components/parameters/function'
    responses:
      '200':
        description: 'Information about the feature collection plus the first features matching the selection parameters.'
        content:
          application/geo+json:
            schema:
Every feature in a dataset belongs to a collection. A dataset may consist of multiple feature collections. A feature collection is often a collection of features of a similar type, based on a common schema.

This operation returns GeoJSON.

operationId: getFeaturesJSON
tags:
  - Features
parameters:
  - $ref: '#/components/parameters/limit'
  - $ref: '#/components/parameters/bbox'
  - $ref: '#/components/parameters/datetime'
  - $ref: '#/components/parameters/function'
responses:
  '200':
    description: >-
      Information about the feature collection plus the first features matching the selection parameters.
    content:
      application/geo+json:
        schema:
          $ref: '#/components/schemas/featureCollectionGeoJSON'
default:
  description: An error occurred.
  content:
    application/json:
      schema:
        $ref: '#/components/schemas/exception'
'/collections/buildings/items/{featureId}':
get:
  operationId: getFeature
tags:
  - Features
parameters:
- $ref: '#/components/parameters/featureId'
responses:
  '200':
    description: A feature.
    content:
      application/geo+json:
        schema:
          $ref: '#/components/schemas/buildingGeoJSON'
      text/html:
        schema:
          type: string
    default:
      description: An error occurred.
      content:
        application/json:
          schema:
            $ref: '#/components/schemas/exception'
        text/html:
          schema:
            type: string
'/collections/buildings/items/{featureId}.json':
  get:
    summary: retrieve a feature in GeoJSON
    operationId: getFeatureJSON
    tags:
      - Features
    parameters:
      - $ref: '#/components/parameters/featureId'
    responses:
      '200':
        description: A feature.
        content:
          application/geo+json:
            schema:
              $ref: '#/components/schemas/buildingGeoJSON'
      default:
        description: An error occurred.
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/exception'
          text/html:
            schema:
              type: string
components:
  parameters:
    limit:
      name: limit
      in: query
      description: |
        The optional limit parameter limits the number of items that are presented in the response document.
Only items are counted that are on the first level of the collection in the response document. Nested objects contained within the explicitly requested items shall not be counted.

* Minimum = 1
* Maximum = 10000
* Default = 10
required: false

schema:
  type: integer
  minimum: 1
  maximum: 10000
  default: 10
  style: form
  explode: false
bbox:
  name: bbox
  in: query
  description: >
  Only features that have a geometry that intersects the bounding box are selected.
  The bounding box is provided as four or six numbers, depending on whether the coordinate reference system includes a vertical axis (elevation or depth):

* Lower left corner, coordinate axis 1
* Lower left corner, coordinate axis 2
* Lower left corner, coordinate axis 3 (optional)
* Upper right corner, coordinate axis 1
* Upper right corner, coordinate axis 2
* Upper right corner, coordinate axis 3 (optional)

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 the parameter 'bbox-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 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.
required: false

schema:
  type: array
  minItems: 4
  maxItems: 6
  items:
    type: number
Either a date-time or an interval, open or closed. Date and time expressions adhere to RFC 3339. Open intervals are expressed using double-dots.

Examples:

* A date-time: "2018-02-12T23:20:50Z"
* A closed interval: "2018-02-12T00:00:00Z/2018-03-18T12:31:12Z"
* Open intervals: "2018-02-12T00:00:00Z/.." or "../2018-03-18T12:31:12Z"

Only features that have a temporal property that intersects the value of `datetime` are selected.

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.
exception:
  type: object
required:
  - code
properties:
  code:
    type: string
description:
  type: string
root:
  type: object
required:
  - links
properties:
  links:
    type: array
    items:
      $ref: '#/components/schemas/link'
example:
  - href: 'http://data.example.org/
    rel: self
    type: application/json
    title: this document
  - href: 'http://data.example.org/api'
    rel: service
    type: application/vnd.oai.openapi+json;version=3.0
    title: the API definition
  - href: 'http://data.example.org/conformance'
    rel: conformance
    type: application/json
    title: OGC API Features conformance classes implemented by this server
  - href: 'http://data.example.org/collections'
    rel: data
    type: application/json
    title: Metadata about the feature collections
req-classes:
  type: object
required:
  - conformsTo
properties:
  conformsTo:
    type: array
    items:
      type: string
example:
  - 'http://www.opengis.net/spec/ogcapi-features-1/1.0/req/core'
  - 'http://www.opengis.net/spec/ogcapi-features-1/1.0/req/oas30'
  - 'http://www.opengis.net/spec/ogcapi-features-1/1.0/req/html'
  - 'http://www.opengis.net/spec/ogcapi-features-1/1.0/req/geojson'
link:
  type: object
required:
  - href
properties:
  href:
    type: string
  rel:
    type: string
    examples: [prev]
  type:
    type: string
    examples: [application/geo+json]
  hreflang:
    type: string
    examples: [en]
  title:
    type: string
    examples: [previous page]
content:
  type: object
required:
  - links
  - collections
properties:
  links:
    type: array
    items:
      $ref: '#/components/schemas/link'
example:
  - 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/foobar.xsd'
    rel: describedBy
    type: application/xml
    title: XML schema for Acme Corporation data
collections:
  type: array
  items:
    $ref: '#/components/schemas/collectionInfo'
collectionInfo:
  type: object
required:
  - name
  - links
properties:
  name:
identifier of the collection used, for example, in URIs

type: string
example: buildings

title:

description: 'human readable title of the collection'
type: string
example: Buildings
description:

description: 'a description of the features in the collection'
type: string
example: Buildings in the city of Bonn.

links:

type: array
items:
  $ref: '#/components/schemas/link'
example:
  - href: 'http://data.example.org/collections/buildings/items'
    rel: item
    type: application/geo+json
    title: Buildings
  - href: 'http://example.org/concepts/building.html'
    rel: describedBy
    type: text/html
    title: Feature catalogue for buildings

extent:
  $ref: '#/components/schemas/extent'

crs:

description: 'The coordinate reference systems in which geometries may be retrieved. Coordinate reference systems are identified by a URI. The first coordinate reference system is the coordinate reference system that is used by default. This is always "http://www.opengis.net/def/crs/OGC/1.3/CRS84", i.e. WGS 84 longitude/latitude.'
type: array
items:
  type: string
default:
  - 'http://www.opengis.net/def/crs/OGC/1.3/CRS84'

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: >-
        West, south, east, north edges of the bounding box. The coordinates are in the coordinate reference system specified in 'crs'. By default this is WGS 84 longitude/latitude.
      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'
      default: 'http://www.opengis.net/def/crs/OGC/1.3/CRS84'
  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.
        type: array
        description: >-
          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.

```json
type: array
minItems: 1
items:
  description: >-
    Begin and end times of the time interval. The timestamps are in the 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:
    - 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 reference system. Extensions may support additional temporal reference systems and add additional enum values.
  type: string
enum:
  - 'http://www.opengis.net/def/uom/ISO-8601/0/Gregorian'
  default: 'http://www.opengis.net/def/uom/ISO-8601/0/Gregorian'
```

featureCollectionGeoJSON:
  type: object
  required:
    - type
    - features
  properties:
    type:
      type: string
    enum:
      - FeatureCollection
  features:
    type: array
    items:
      $ref: '#/components/schemas/featureGeoJSON'

links:
  type: array
items:
  $ref: '#/components/schemas/link'
timeStamp:
  type: string
  format: date-time
numberMatched:
  type: integer
  minimum: 0
numberReturned:
  type: integer
  minimum: 0
featureGeoJSON:
  type: object
  required:
  - type
  - geometry
  - properties
  properties:
    type:
      type: string
      enum:
      - Feature
gtometry:
      $ref: '#/components/schemas/geometryGeoJSON'
  properties:
    type: object
    nullable: true
id:
  oneOf:
  - type: string
  - type: integer
buildingGeoJSON:
  type: object
  required:
  - type
  properties:
  type:
    type: string
    enum:
    - Point
    - MultiPoint
    - LineString
    - MultiLineString
    - Polygon
    - MultiPolygon
    - GeometryCollection
buildingGeoJSON:
  type: object
  required:
  - type
  - geometry
- properties
  properties:
    type:
      type: string
    enum:
      - Feature
  geometry:
    $ref: '#/components/schemas/geometryGeoJSON'
properties:
  type: object
  nullable: true
  properties:
    name:
      type: string
    function:
      type: string
    enum:
      - residential
      - commercial
      - public use
    floors:
      type: integer
      minimum: 1
    lastUpdate:
      type: string
      format: date-time
  tags:
    - name: Capabilities
description: >-
        Essential characteristics of this API including information about the data.
    - name: Features
description: >-
        Access to data (features).
Annex C: XML examples (Informative)

C.1. Overview

This annex includes examples of XML/GML responses to illustrate how the OpenAPI fragments used to define the requirements for the Core requirements class are expressed in XML using the OGC API Features Core XML Schema.

C.2. A landing page

```xml
<?xml version="1.0" encoding="UTF-8"?>
<LandingPage
  service="OGCAPI-Features"
  version="1.0.0"
  xmlns="http://www.opengis.net/ogcapi-features-core/1.0"
  xmlns:atom="http://www.w3.org/2005/Atom"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.opengis.net/ogcapi-features-core/1.0 ../core.xsd">
  <Title xml:lang="en">Some Title for this API</Title>
  <Description xml:lang="en">Some description for this API.</Description>
  <atom:link rel="self"
    type="application/json"
    title="This Document"
    href="http://www.acme.com/3.0/wfs?f=application%2Fjson"/>
  <atom:link rel="alternate"
    type="application/xml"
    title="This Document as XML"
    href="http://www.acme.com/3.0/wfs?f=application%2Fxml"/>
  <atom:link rel="alternate"
    type="text/html"
    title="This Document as HTML"
  <atom:link rel="service"
    type="application/json"
    title="API definition for this endpoint as JSON"
    href="http://www.acme.com/3.0/wfs/api?f=application%2Fjson"/>
  <atom:link rel="service"
    type="application/vnd.ogc_wfs+xml"
    title="API definition for this endpoint as XML"
  <atom:link rel="conformance"
    type="application/json"
    title="Conformance Declaration as JSON"
    href="http://www.acme.com/3.0/wfs/conformance?f=application%2Fjson"/>
  <atom:link rel="conformance"
    type="application/xml"
    title="Conformance Declaration as XML"
```
C.3. Conformance declaration

This server conforms to the recommended requirements classes, plus the GML Simple Features Profile, Level 0.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<ConformsTo
  service="OGCAPI-Features"
  version="1.0.0"
  xmlns="http://www.opengis.net/ogcapi-features-core/1.0"
  xmlns:atom="http://www.w3.org/2005/Atom"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.opengis.net/ogcapi-features-core/1.0 .. /core.xsd">
  <atom:link href="http://www.opengis.net/spec/ogcapi-features-1/1.0/req/core"/>
  <atom:link href="http://www.opengis.net/spec/ogcapi-features-1/1.0/req/oas30"/>
  <atom:link href="http://www.opengis.net/spec/ogcapi-features-1/1.0/req/geojson"/>
  <atom:link href="http://www.opengis.net/spec/ogcapi-features-1/1.0/req/gmlsf0"/>
</ConformsTo>
```

C.4. Feature collections

This service offers three feature collections (airport facilities, roads, inland water areas).

```xml
<?xml version="1.0" encoding="UTF-8"?>
<Collections
  service="OGCAPI-Features"
  version="1.0.0"
/>
<Collection>
  <Name>aerofacp_1m</Name>
  <Title>Airport Facilities Points</Title>
  <atom:link rel="items"
    title="Airport Facilities Points"
    type="application/geo+json"
    href="http://www.acme.com/3.0/wfs/collections/aerofacp_1m/items?f=application%2Fjson"/>
  <atom:link rel="items"
    title="Airport Facilities Points"
    type="application/gml+xml;version=3.2;profile=http://www.opengis.net/def/profile/ogc/2.0/gml-sf0"
  <atom:link rel="alternate"
    title="Airport Facilities Points"
    type="text/html"
    href="http://www.acme.com/3.0/wfs/collections/aerofacp_1m/items?f=text%2Fhtml"/>
  <atom:link rel="describedBy"
    title="Schema for Airport Facilities Points"
    type="application/xml"
    href="http://www.acme.com/3.0/wfs/collections/aerofacp_1m/schema"/>
  <Extent>
    <Spatial crs="http://www.opengis.net/def/crs/OGC/1.3/CRS84">
      <LowerCorner>-179.878326416016 -54.9311103820801</LowerCorner>
      <UpperCorner>179.339859008789 79.52944183349609</UpperCorner>
    </Spatial>
  </Extent>
</Collection>
C.5. Feature collection

Only the information for the selected feature collection (roads) is included in the response.
<?xml version="1.0" encoding="UTF-8"?>
<Collections
    service="OGCAPI-Features"
    version="1.0.0"
    xmlns="http://www.opengis.net/ogcapi-features-core/1.0"
    xmlns:atom="http://www.w3.org/2005/Atom"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.opengis.net/ogcapi-features-core/1.0 ..../core.xsd">
    <Collection>
        <Name>roadl_1m</Name>
        <Title>Roads</Title>
        <atom:link rel="items"
            title="Roads"
            type="application/geo+json"
            href="http://www.acme.com/3.0/wfs/collections/roadl_1m/items?f=application%2Fgeo%2Bjson"/>
        <atom:link rel="items"
            title="Roads"
            type="application/gml+xml;version=3.2;profile=http://www.opengis.net/def/profile/ogc/2.0/gml-sf0"
        <atom:link rel="alternate"
            title="Roads"
            type="text/html"
            href="http://www.acme.com/3.0/wfs/collections/roadl_1m/items?f=text%2Fhtml"/>
        <atom:link rel="describedBy"
            title="Schema for Roads"
            type="application/xml"
            href="http://www.acme.com/3.0/wfs/collections/roadl_1m/schema"/>
        <Extent>
            <Spatial crs="http://www.opengis.net/def/crs/OGC/1.3/CRS84">
                <LowerCorner>-179.999420166016 -54.88802337646479</LowerCorner>
                <UpperCorner>179.9999 74.740592956543</UpperCorner>
            </Spatial>
            <Temporal trs="http://www.opengis.net/def/uom/ISO-8601/0/Gregorian">
                <begin>2017-01-01T00:00:00Z</begin>
                <end>2017-12-31T23:59:59Z</end>
            </Temporal>
        </Extent>
        <DefaultCRS>http://www.opengis.net/def/crs/OGC/1.3/CRS84</DefaultCRS>
    </Collection>
</Collections>
C.6. Features

This response contains 2 features of the airport facilities collection and has a link to the next features.
<core:member>
  <atom:link rel="self" title="This feature" type="application/gml+xml;version=3.2;profile=http://www.opengis.net/def/profile/ogc/2.0/gml-sf0"
  <atom:link rel="alternate" title="This feature as GeoJSON" type="application/geo+json"
    href="http://www.acme.com/3.0/wfs/collections/aerofacp_1m/items/1?f=application%2Fgeo%2Bjson"/>
  <atom:link rel="alternate" title="This feature as HTML" type="text/html"
    href="http://www.acme.com/3.0/wfs/collections/aerofacp_1m/items/1?f=application%2Ftext%2Bhtml"/>
  <aerofacp_1m gml:id="F1">
    <geometry>
      <gml:Point gml:id="geom1" srsName="http://www.opengis.net/def/crs/OGC/1.3/CRS84">
        <gml:pos>-176.466049194336 -43.81286239624023</gml:pos>
      </gml:Point>
    </geometry>
    <id>1455</id>
    <f_code>GB005</f_code>
    <iko>NZCI</iko>
    <nam>CHATHAM ISLANDS</nam>
    <na3>NZ58133</na3>
  </aerofacp_1m>
</core:member>
<use>999</use>
<zv3>13</zv3>
<tile_id>434</tile_id>
<end_id>1</end_id>
</aerofacp_1m>
</core:member>
<core:member>

<atom:link rel="self" title="This feature" type="application/gml+xml;version=3.2;profile=http://www.opengis.net/def/profile/ogc/2.0/gml-sf0"

<atom:link rel="alternate" title="This feature as GeoJSON" type="application/geo+json"
href="http://www.acme.com/3.0/wfs/collections/aerofacp_1m/items/2?f=application%2Fgeo%2Bjson"/>

<atom:link rel="alternate" title="This feature as HTML" type="text/html"
href="http://www.acme.com/3.0/wfs/collections/aerofacp_1m/items/2?f=application%2Ftext%2Bhtml"/>

<aerofacp_1m gml:id="F2">
<geometry>
  <gml:Point gml:id="geom2"
srsName="http://www.opengis.net/def/crs/OGC/1.3/CRS84">
    <gml:pos>-149.5207672119141 -23.3629035949707</gml:pos>
  </gml:Point>
</geometry>

{id>4421</id>
<f_code>G6005</f_code>
<iko>NTAT</iko>
<nam>TUBUAI</nam>
<na3>FP67494</na3>
<use>49</use>
<zv3>3</zv3>
<tile_id>397</tile_id>
<end_id>1</end_id>
</aerofacp_1m>
</core:member>
</core:FeatureCollection>
## Annex D: Revision History

<table>
<thead>
<tr>
<th>Date</th>
<th>Release</th>
<th>Editor</th>
<th>Primary clauses modified</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2017-10-09</td>
<td>3.0.0-SNAPSHOT</td>
<td>C. Portele</td>
<td>all</td>
<td>initial version</td>
</tr>
<tr>
<td>2017-10-11</td>
<td>3.0.0-SNAPSHOT</td>
<td>C. Portele</td>
<td>all</td>
<td>changes discussed in SWG/PT call on 2017-10-09</td>
</tr>
<tr>
<td>2017-12-13</td>
<td>3.0.0-SNAPSHOT</td>
<td>C. Portele</td>
<td>all</td>
<td>address issues #2, #5, #6, #7, #8, #14, #15, #19</td>
</tr>
<tr>
<td>2018-01-22</td>
<td>3.0.0-SNAPSHOT</td>
<td>C. Portele</td>
<td>7</td>
<td>add description of the UML diagram</td>
</tr>
<tr>
<td>2018-02-01</td>
<td>3.0.0-SNAPSHOT</td>
<td>C. Portele</td>
<td>2, 3, 5, 7</td>
<td>add links to recent issues on GitHub; address issues #31, #32</td>
</tr>
<tr>
<td>2018-02-11</td>
<td>3.0.0-SNAPSHOT</td>
<td>C. Portele</td>
<td>2, 6, 7, 8</td>
<td>address issue #25</td>
</tr>
<tr>
<td>2018-02-27</td>
<td>3.0.0-SNAPSHOT</td>
<td>C. Portele</td>
<td>all</td>
<td>address issues #3, #9, #12, #22, #23, #24, #44; add links to issues #41, #45, #46, #47</td>
</tr>
<tr>
<td>2018-03-04</td>
<td>3.0.0-SNAPSHOT</td>
<td>T. Schaub</td>
<td>7, B</td>
<td>JSON schema fixes #54, #55</td>
</tr>
<tr>
<td>2018-03-12</td>
<td>3.0.0-SNAPSHOT</td>
<td>C. Portele</td>
<td>all</td>
<td>Updates after the WFS 3.0 Hackathon #59, #61, #62, #63, #64, #69, #72, #77, #78; resolve #4; editorial edits</td>
</tr>
<tr>
<td>2018-03-15</td>
<td>3.0.0-SNAPSHOT</td>
<td>J. Amara</td>
<td>7</td>
<td>Uniqueness of feature id #83</td>
</tr>
<tr>
<td>2018-03-21</td>
<td>3.0.0-SNAPSHOT</td>
<td>I. Rinne</td>
<td>7</td>
<td>Clarified the requirement /req/core/crs84 #92</td>
</tr>
<tr>
<td>2018-03-28</td>
<td>3.0.0-SNAPSHOT</td>
<td>C. Portele</td>
<td>3, 4, 7</td>
<td>Temporal support #57, bbox no longer restricted to CRS84 #60, clarify ‘collection’ #86, clarify feature id constraints #84</td>
</tr>
<tr>
<td>2018-04-02</td>
<td>3.0.0-SNAPSHOT</td>
<td>C. Portele</td>
<td>7, B</td>
<td>Clarify ‘item’ links #81, clean up OpenAPI example in Annex B</td>
</tr>
<tr>
<td>2018-04-03</td>
<td>3.0.0-SNAPSHOT</td>
<td>C. Portele</td>
<td>4 to 9</td>
<td>Clean-up asciidoc #100</td>
</tr>
<tr>
<td>2018-04-04</td>
<td>3.0.0-SNAPSHOT</td>
<td>P. Vretanos, C. Portele</td>
<td>8.4, 8.5, C</td>
<td>Clarify XML encoding #58</td>
</tr>
<tr>
<td>2018-04-05</td>
<td>3.0.0-SNAPSHOT</td>
<td>C. Heazel</td>
<td>A</td>
<td>Initial version of the Abstract Test Suite #112</td>
</tr>
<tr>
<td>2018-04-05</td>
<td>3.0.0-SNAPSHOT</td>
<td>C. Portele</td>
<td>C</td>
<td>Fix axis order in example #113</td>
</tr>
<tr>
<td>2018-04-07</td>
<td>3.0.0-SNAPSHOT</td>
<td>C. Portele</td>
<td>7, 9, 10</td>
<td>Add HTTP status code guidance #105, add warning about OpenAPI media type #117</td>
</tr>
<tr>
<td>2018-04-07</td>
<td>3.0.0-SNAPSHOT</td>
<td>C. Reed, C. Portele</td>
<td>all</td>
<td>Edits after review #119</td>
</tr>
<tr>
<td>2018-04-07</td>
<td>3.0.0-draft.1</td>
<td>C. Portele</td>
<td>iv, v</td>
<td>First draft release</td>
</tr>
<tr>
<td>2019-02-14</td>
<td>3.0.0-SNAPSHOT</td>
<td>C. Portele, C. Holmes</td>
<td>all</td>
<td>Bugfixes #149 and #176, change rel-item to rel-items #175, use {collectionId}, {featureId} and id consistently #171</td>
</tr>
</tbody>
</table>

104
<table>
<thead>
<tr>
<th>Date</th>
<th>Release</th>
<th>Editor</th>
<th>Primary clauses modified</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2019-05-02</td>
<td>3.0.0-SNAPSHOT</td>
<td>C. Portele</td>
<td>all</td>
<td>Temporal data support #155, extents #168, result set consistency during paging #192</td>
</tr>
<tr>
<td>2019-05-20</td>
<td>1.0.0-SNAPSHOT</td>
<td>C. Portele</td>
<td>all</td>
<td>Change document title to &quot;OGC API - Features&quot; #189, minor editorial issues #204, introduce yaml #201, HEAD/OPTIONS #115, /collections path structure #90, resource names #199, /items #164, bbox/time parameter behavior for features without spatial/temporal data #122, change language in overview #124, update XMI #209</td>
</tr>
<tr>
<td>2019-06-11</td>
<td>1.0.0-SNAPSHOT</td>
<td>C. Portele</td>
<td>5.6, 7.2, 7.11</td>
<td>Add clarification about default parameter values #215, add title/description to landing page #227, correct informative wording about coordinate reference systems</td>
</tr>
<tr>
<td>2019-06-13</td>
<td>1.0.0-SNAPSHOT</td>
<td>C. Heazel, C. Portele, P. Vretanos</td>
<td>0, 7, 8, 11 (new), A, C</td>
<td>Listing of all applicable HTTP Status Codes #45, Deviations between XML and JSON encoding of various structures #133, Add section &quot;Security Considerations&quot; #137, Issues with the UML model and resource descriptions #217</td>
</tr>
<tr>
<td>2019-06-22</td>
<td>1.0.0-SNAPSHOT</td>
<td>C. Portele</td>
<td>all</td>
<td>Editorial cleanup, Add anchors #225</td>
</tr>
</tbody>
</table>
Annex E: Bibliography

- IANA: Link Relation Types, https://www.iana.org/assignments/link-relations/link-relations.xml
- OGC: Web Feature Service 2.0, http://docs.opengeospatial.org/is/09-025r2/09-025r2.html
- W3C: Data Catalog Vocabulary, W3C Recommendation 16 January 2014, https://www.w3.org/TR/vocab-dcat/