OGC Web Feature Service 3.0

Part 1 - Core

Table of Contents

Open Geospatial Consortium

Submission Date: <yyyy-mm-dd>

Approval Date: <yyyy-mm-dd>

Publication Date: <yyyy-mm-dd>

External identifier of this OGC® document: http://www.opengis.net/doc/IS/wfs-1/3.0

Internal reference number of this OGC® document: 17-069

Version: 3.0.0-SNAPSHOT (2018-04-03)

Category: OGC® Implementation Specification

Editor: Clemens Portele, Panagiotis (Peter) A. Vretanos

OGC Web Feature Service 3.0 - Part 1: Core

Copyright notice

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

A Web Feature Service (WFS) offers the capability to create, modify and query spatial data on the Web. WFS is a multi-part standard. This part specifies the core capabilities that every WFS supports and is restricted to read-access to data. Additional capabilities that address specific needs will be specified in additional parts. Examples include support for creating and modifying data, more complex data models, richer queries, additional coordinate reference systems.

By default, every WFS provides access to a single dataset. Rather than sharing the data as a complete dataset, WFS offers direct, fine-grained access to the data at the feature (object) level.

Consistent with the architecture of the Web, this version of WFS uses a resource architecture and specifies a RESTful service interface consistent with the HTTP/HTTPS standards.

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 allow the server to be interrogated to determine its capabilities and retrieve information (metadata) about this distribution of the dataset. This includes the API definition of the server as well as metadata about the feature collections provided by the server.

Query operations allow features or values of feature properties to be retrieved from the underlying data store based upon selection criteria, defined by the client, on feature properties.

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

Resource	Path	HTTP method	Document reference
Landing page	`/`	GET	7.2 API landing page
API definition	`/api`	GET	7.3 API definition
Conformance classes	`/conformance`	GET	7.4 Declaration of conformance classes
Feature collections metadata	`/collections`	GET	7.11 Feature collections metadata
Feature collection metadata	`/collections/{name}`	GET	7.12 Feature collection metadata
Feature collection	`/collections/{name}/items`	GET	7.13 Feature collections
Feature	`/collections/{name}/items/{fid}`	GET	7.14 Feature

ii. Keywords

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

ogcdoc, OGC document, web feature service, wfs, feature, property, geographic information, spatial

data, spatial things, dataset, distribution, API, openapi, geojson, gml, html

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.
- interactive instruments GmbH
- …

v. Submitters

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

Name	Affiliation
Clemens Portele	interactive instruments GmbH

Name	Affiliation
Panagiotis (Peter) A. Vretanos	CubeWerx Inc.
…	…

Chapter 1. Scope

This International Standard specifies the behaviour of a server that provides access to features in a dataset in a manner independent of the underlying data store. It specifies discovery and query operations.

Discovery operations allow the server to be interrogated to determine its capabilities and retrieve information (metadata) about this distribution of the dataset. This includes the API definition of the server as well as metadata about the feature collections provided by the server.

Query operations allow features to be retrieved 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 services".

The main requirements class is:

- Core.

It specifies requirements that all WFS have to meet.

The "Core" does not mandate any 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" requirements class may also decide to use none of them, but to use another encoding instead.

That said, the Core requirements class includes recommendations to support HTML and GeoJSON as encodings, where practical. Clause 6 (Overview) includes a discussion about 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 defined for this, 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 representations of the API definition 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 or an OWS Common 2.0 capabilities document.

The "Core" is intended to be the minimal useful service interface for fine-grained access to a spatial dataset.

Additional capabilities, for example, 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 WFS 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.

- Open API Initiative: OpenAPI Specification 3.0.1, https://github.com/OAI/OpenAPI-Specification/blob/master/versions/3.0.1.md
- Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T.: IETF RFC 2616, HTTP/1.1, http://tools.ietf.org/rfc/rfc2616.txt
- Klyne, G., Newman, C.: IETF RFC 3339, Web Linking, http://tools.ietf.org/rfc/rfc3339.txt
- Nottingham, M.: IETF RFC 5988, Web Linking, http://tools.ietf.org/rfc/rfc5988.txt
- van den Brink, L., Portele, C., Vretanos, P.: OGC 10-100r3, Geography Markup Language (GML) Simple Features Profile, http://portal.opengeospatial.org/files/?artifact_id=42729
- Butler, H., Daly, M., Doyle, A., Gillies, S., Hagen, S., Schaub, T.: IETF RFC 7946, The GeoJSON Format, https://tools.ietf.org/rfc/rfc7946.txt
- W3C: HTML5, W3C Recommendation, http://www.w3.org/TR/html5/
- Schema.org: http://schema.org/docs/schemas.html

Chapter 4. Terms and Definitions

This document uses the terms defined in Sub-clause 5.3 of [OGC 06-121r8], 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.

CAUTION	Add link to the informative WFS Guide, once it is available.

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 a web service that provides the data.

4.3. feature

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

NOTE	If you are unfamiliar with the term 'feature', the explanations in the W3C/OGC Spatial Data on the Web Best Practice document may help, in particular the section on Spatial Things, Features and Geometry.

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.

Chapter 5. Conventions

5.1. Open issues

This version is a draft. It is not complete and there are open issues that are under discussion. These cases are shown as "CAUTION" annotations with a link to the issue on GitHub and a brief summary of the issue.

The current expectation is to have a stable version of the Core specification in 2019. Criteria to move the specification to the next stage in the process are:

- Multiple implementations of each conformance class,
- a conformance test suite for each conformance class,
- multiple implementations of a generic WFS client,
- multiple implementations of clients using the OpenAPI definition of a WFS,
- multiple draft extensions to verify the extensibility, and
- resolution of comments received on GitHub or in formal reviews in OGC and ISO/TC 211.

5.2. Identifiers

The normative provisions in this specification are denoted by the URI `http://www.opengis.net/spec/wfs-1/3.0`.

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

5.3. UML model

UML diagrams are included in this standard to illustrate the conceptual model that underpins Web Feature Service implementations. The UML model is not normative. The UML profile used is specified in ISO 19103:2015.

Resources are modelled as UML interfaces.

5.4. Link relations

To express relationships between resources, RFC 5988 (Web Linking) and registered link relation types are used.

5.5. 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 WFS are expected to use HTTPS, not HTTP.

5.6. API definition

5.6.1. General remarks

Good documentation is essential for every API so that developers can learn how to use it. 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.

The "Core" requirements class, therefore, only requires that an API definition is provided at path `/api`.

A separate requirements class is specified for API definitions that follow the OpenAPI specification 3.0, but this does not preclude that in the future or in parallel other versions of OpenAPI or other 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 landscape will continue to evolve.

In this document, fragments of OpenAPI definitions are shown in YAML since YAML is easier to read than JSON and is typically used in OpenAPI editors.

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 only a subset of all possible values are applicable to the service). An example for a constrained range of values is to explicitly specify the supported values of a string parameter or property using an enum.
- Additional properties may be added to the schema definition of a Response Object.
- Informative text may be changed or added, like comments or description properties.

For 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.

CAUTION	ISSUE 98 Server Ambiguity in OpenAPI

Example 1. URL of the OpenAPI definition

If the OpenAPI Server Object would look like this:

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

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

5.6.5. Reusable OpenAPI components

Reusable components for OpenAPI definitions of a WFS 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 eventually they are expected to be available under the base URL "http://schemas.opengis.net/wfs/3.0/openapi/".

Chapter 6. Overview

6.1. Evolution from previous versions of WFS

The previous versions of the WFS standard used a Remote-Procedure-Call-over-HTTP architectural style using XML for any payloads as it was state-of-the-art in the late 1990s and early 2000s, when WFS was originally designed. This version specifies a modernized service, 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 is modularization. This has a few facets:

- Clear separation between core requirements that almost everyone has who wants to share or use spatial data on a fine-grained level (this document) and more advanced capabilities that communities are using today (extensions specified in additional parts of WFS 3.0).
- 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 WFS modules, but about providing building blocks for fine-grained access to spatial data that can be used in data APIs in general. In other words, a server supporting WFS 3.0 should not be seen as a standalone WFS service. A corollary of this is that it should be possible to implement a data API that at the same time conforms to conformance classes from WFS 3.0 and from other OGC Web Service standards following a similar approach.

This approach intends to support two types of client developers:

- those that have never heard about WFS - it should be possible to create a client using the API definition without the need to read the WFS standard (they may need to learn a little bit about geometry, etc.);
- those that want to write a "generic" client that can access WFSs, i.e. are not specific for a particular API/server.

As a result of this modernization, WFS 3.0 implementations are not backwards compatible with WFS 2.0 implementations per se. However, it has been a design goal to define WFS 3.0 in a way so that the WFS 3.0 interface can be mapped to a WFS 2.0 implementation. WFS 3.0 is intended to be simpler and more modern, but still an evolution from the previous versions and their implementations.

The modernization is discussed in more detail here.

CAUTION	Change this link and point to the WFS 3.0 Guide once a draft is available. The Guide will include a mapping between OGC Capabilities and OpenAPI as well as a mapping between WFS 2.0 operations and WFS 3.0.

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 also decide to use none of them, but to use 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 it 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 to support 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:

- solids are used a geometries (e.g. in a 3D city model),
- geometries include non-linear curve interpolations that cannot be simplified (e.g., use of arcs in authoritative geometries),
- geometries 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 have more than one geometric property, etc.

In addition to HTML and GeoJSON, a significant amount 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 previous paragraph, but as a result GML also more complex to handle for both servers and clients. Conformance classes for GML are, therefore, included in this standard, but it is expected that these will typically be supported by servers where users are known to expect feature data in XML/GML.

The recommendations for 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, too, in future versions of this standard to provide guidance.

This part of WFS 3.0 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 the data incrementally parseable, the media type `application/geo+json-seq` would be used.

In addition, HTTP supports compression and 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.

Chapter 7. Requirement Class "Core"

7.1. Overview

Requirements Class	
http://www.opengis.net/spec/wfs-1/3.0/req/core	
Target type	Web service
Dependency	RFC 2616 (HTTP/1.1)
Dependency	RFC 3339 (Date and Time on the Internet: Timestamps)
Dependency	RFC 5988 (Web Linking)

Figure 1 illustrates the resources supported by the Core requirements class using UML. Each resource type available through the server is an «interface».

A server that implements the WFS API 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.

That is, each WFS has a single `LandingPage` (path `/`) that provides links to

- the `APIDefinition` (path `/api`),
- the `Conformance` statements (path `/conformance`),
- the `DatasetDistribution` metadata (path `/collections`).

The `APIDefinition` describes the capabilities of the server and which can be used by clients to connect to the server or by development tools to support the implementation of servers and clients. Accessing the `APIDefinition` using HTTP GET returns a description of the API.

Accessing the `Conformance` using HTTP GET returns a list of URIs of requirements classes implemented by the server.

The distribution consists of a set of feature collections. This specification does not include any requirements 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 be used, too.

Accessing the `DatasetDistribution` using HTTP GET returns a `DatasetDistributionResponse`, which includes a link to each `Collection` in the distribution along with metadata about each collection:

- a local identifier for the collection that is unique within the WFS;
- a list of coordinate reference systems in which geometries may be returned by the server, where the first one 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.

Each `Collection` (path `/collections/{collection-name}/items`) consists of the features in the collection where each feature in the distribution is part of exactly one collection.

CAUTION	ISSUE 30 Allow also features that do not belong to any collection?

CAUTION	ISSUE 66 Support features that do belong to multiple collections?

Accessing a `Collection` using HTTP GET returns a `CollectionResponse`, which basically consists of features in the collection. The features included in the response are determined by the server based on parameters of the request.

A `bbox` or `time` parameter may be used to select only a subset of the features in the collection (the features that are located in the bounding box or time period).

The `limit` parameter may be used to request only a subset of the selected features and to indicate that the client wants to page through the selected features of the collection.

The `CollectionResponse` may include metadata about the number of selected and returned features (`numberMatched` and `numberReturned`) as well as links to simplify paging (`next` and `prev`).

Each `Feature` (path `/collections/{collection-name}/items/{feature-id}`) is also a separate resource and may be requested individually using HTTP GET.

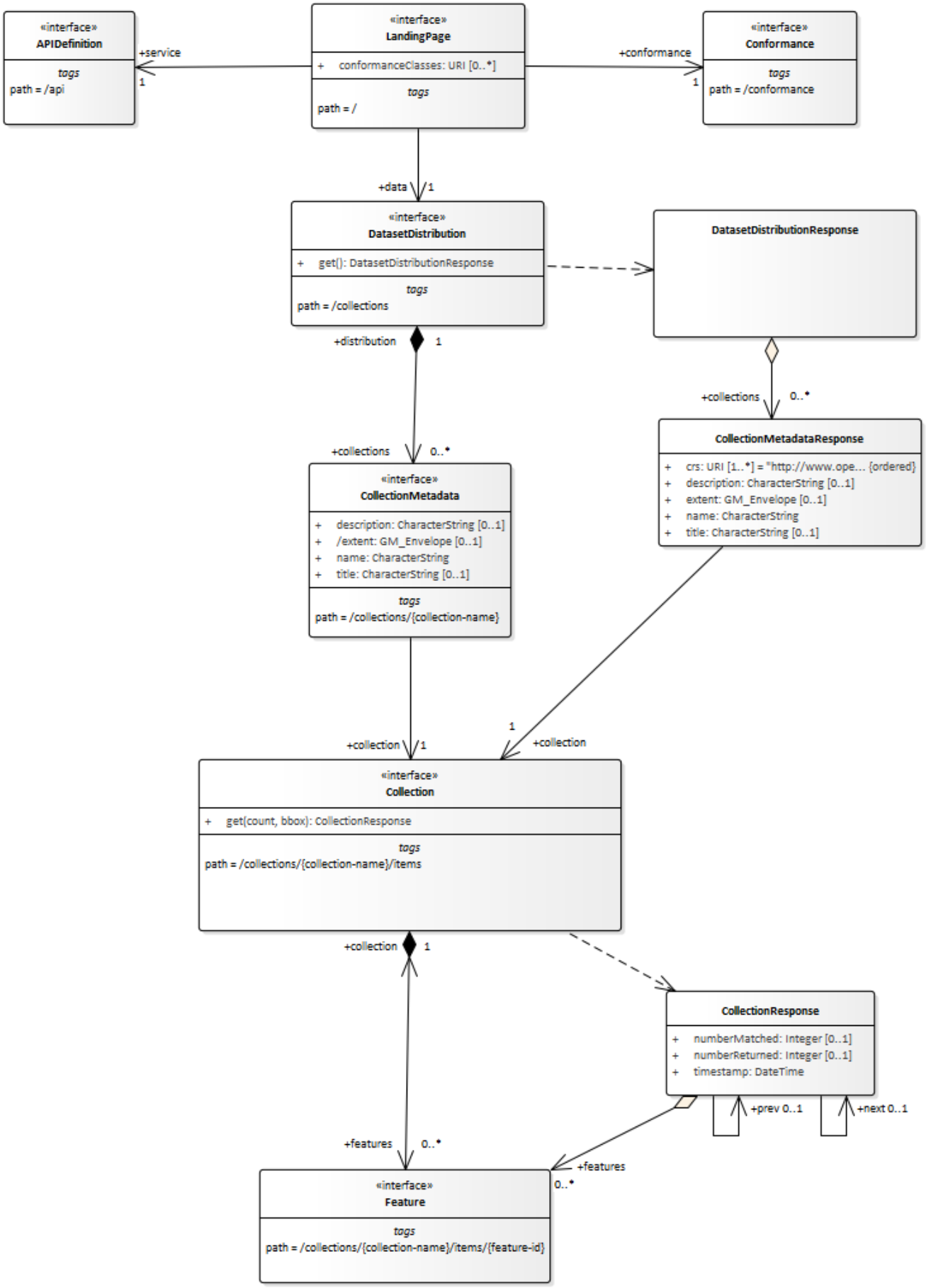

Figure 1. Resources in the Core requirements class

CAUTION	ISSUE 90 More flexible path structure under `/collections`?

7.2. API landing page

7.2.1. Operation

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

7.2.2. Response

Requirement 2	/req/core/root-success A successful execution of the operation SHALL be reported as a response with a HTTP status code `200`. The content of that response SHALL be based upon the OpenAPI 3.0 schema root.yaml and include at least links to the following resources: • `/api` (relation type 'service') • `/conformance` (relation type 'conformance') • `/collections` (relation type 'data')

CAUTION	ISSUE 101 Landing page: Can we reuse existing relation types instead of 'conformance' and 'data'?

Schema for the landing page

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

Example 2. Landing page response document


```
{
  "links": [
    { "href": "http://data.example.org/",
      "rel": "self", "type": "application/json", "title": "this document" },
    { "href": "http://data.example.org/api",
      "rel": "service", "type": "application/openapi+json;version=3.0", "title":
"the API definition" },
    { "href": "http://data.example.org/conformance",
      "rel": "conformance", "type": "application/json", "title": "WFS 3.0
conformance classes implemented by this server" },
    { "href": "http://data.example.org/collections",
      "rel": "data", "type": "application/json", "title": "Metadata about the
feature collections" }
  ]
}
```


7.3. API definition

7.3.1. Operation

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

Requirement 3	/req/core/api-definition-op The server SHALL support the HTTP GET operation at the path `/api`.

7.3.2. Response

Requirement 4	/req/core/api-definition-success A successful execution of the operation SHALL be reported as a response with a HTTP status code `200`. The server SHALL return an API definition document.

Recommendation 1	/rec/core/api-definition-oas If the API definition document uses the OpenAPI Specification 3.0, the document SHOULD conform to the OpenAPI Specification 3.0 requirements class.

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

The API definition document describes the API. I.e., there is no need to include the `/api` operation in the API definition itself.

The idea is that any WFS 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, etc., but it should not be required to read this standard to access the data via the API.

7.4. Declaration of conformance classes

7.4.1. Operation

To support "generic" clients for accessing Web Feature Services in general - and not "just" a specific API / server, the server has to declare the requirements classes it implements and conforms to, too.

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

7.4.2. Response

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

Schema for the list of requirements classes

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

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.

```
{
  "conformsTo": [
    "http://www.opengis.net/spec/wfs-1/3.0/req/core",
    "http://www.opengis.net/spec/wfs-1/3.0/req/oas30",
    "http://www.opengis.net/spec/wfs-1/3.0/req/html",
    "http://www.opengis.net/spec/wfs-1/3.0/req/geojson"
  ]
}
```

7.5. HTTP 1.1

Requirement 7	/req/core/http The server SHALL conform to HTTP 1.1.

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

CAUTION	ISSUE 105 WFS 2.0 defined exception codes and mapped them to HTTP status codes. Should WFS 3.0 provide guidance on which (common) error situations should return which HTTP status code per resource or should we simply require that http status codes are used correctly?

7.6. Web caching

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

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

CAUTION	ISSUE 38 More detail / examples on caching. Add an example OpenAPI operation (headers, response codes) - here or in clause 9.

7.7. Support for cross-origin requests

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

Recommendation 3	/rec/core/cross-origin If the server is intended to be accessed from the browser, cross-origin requests SHOULD be supported. Note that support can also be added in a proxy layer on top of the server.

Two common mechanisms to support cross-origin requests are:

- Cross-origin resource sharing (CORS)
- JSONP (JSON with padding)

7.8. Encodings

While WFS 3.0 does not include any mandatory encoding, it recommends the following encodings. See Clause 6 (Overview) for a discussion.

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

Recommendation 5	/rec/core/geojson 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 specification.

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.

NOTE

Two common approaches are:

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

7.9. Coordinate reference systems

As discussed in Chapter 9 of the W3C/OGC Spatial Data on the Web Best Practices, 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, Web Feature Service 3.0 uses WGS84 longitude and latitude as the default coordinate reference system.

Requirement 8	/req/core/crs84 Unless the client explicitly requests a different coordinate reference system, all spatial geometries SHALL be in the coordinate reference system http://www.opengis.net/def/crs/OGC/1.3/CRS84 (WGS84 longitude/latitude).

The 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 than the native one of the published features. Such a capability will be specified in another part(s) of the WFS 3.0 series.

7.10. Link headers

Recommendation 6	/rec/core/link-header Links included in payload of responses SHOULD also be included as `Link` headers in the HTTP response according to RFC 5988, Clause 5. This recommendation does not apply, if there are a large number of links included in a response or a link is not known when the HTTP headers of the response are created.

7.11. Feature collections metadata

7.11.1. Operation

Requirement 9	/req/core/fc-md-op The server SHALL support the HTTP GET operation at the path `/collections`.

7.11.2. Response

Requirement 10	/req/core/fc-md-success A successful execution of the operation SHALL be reported as a response with a HTTP status code `200`. The content of that response SHALL be based upon the OpenAPI 3.0 schema content.yaml.

Schema for the metadata about feature collections

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

Requirement 11	/req/core/fc-md-links A `200`-response SHALL include the following links in the `links` property of the response: • a link to this response document (relation: `self`), • a link to the response document in every other media type supported by the server (relation: `alternate`). All links SHALL include the `rel` and `type` link parameters.

Recommendation 7	/rec/core/fc-md-descriptions 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`). The `type` link parameter SHOULD be provided for each link. This applies to resources that describe to the whole dataset. 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. Examples for descriptions are: XML Schema, Schematron, JSON Schema, RDF Schema, OWL, SHACL, a feature catalogue, etc.

CAUTION	ISSUE 56 Lack of DescribeFeatureType request

CAUTION	ISSUE 102 Add a recommendation about a link to the dataset metadata (for example, in DCAT). Which link relation type?

Requirement 12	/req/core/fc-md-items For each feature collection in this distribution of the dataset, an item SHALL be provided in the property `collections`.

Requirement 13	/req/core/fc-md-items-links For each feature collection in this distribution of the dataset, the `links` property of the collection SHALL include an item for each supported encoding with a link to the collection resource (relation: `item`). All links SHALL include the `rel` and `type` properties.

CAUTION	ISSUE 103 Can/should we make use of the new Link Object in OpenAPI 3.0?

Requirement 14	/req/core/fc-md-extent For each feature collection, the `extent` property, if provided, SHALL be a bounding box that includes all spatial and temporal geometries in this collection. 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.

Schema for the metadata about a feature collection

```
type: object
required:
  - name
  - links
properties:
  name:
    description: identifier of the collection used, for example, in URIs
    type: string
  title:
    description: human readable title of the collection
    type: string
  description:
    description: a description of the features in the collection
    type: string
  links:
    type: array
    items:
      $ref:
https://raw.githubusercontent.com/opengeospatial/WFS_FES/master/core/openapi/schemas/l
ink.yaml
  extent:
    $ref:
https://raw.githubusercontent.com/opengeospatial/WFS_FES/master/core/openapi/schemas/e
xtent.yaml
  crs:
    description: the list of coordinate reference systems supported by the service;
the first item is the default coordinate reference system
    type: array
    items:
      type: string
    default:
      - http://www.opengis.net/def/crs/OGC/1.3/CRS84
```

NOTE The `crs` property is not used by this conformance class, but reserved for future use.

Example 4. Feature collection metadata response document

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

Representations of the metadata resource in other formats are referenced using link relation type "alternate".

Additional links to a GML application schema for the building data and to a web page that has additional information about buildings are provided, too, using link relation type "describedBy".

Coordinate reference system information is not provided as the service provides geometries only in the default system (WGS84 longitude/latitude).


```
{
  "links": [
    { "href": "http://data.example.org/collections.json",
      "rel": "self", "type": "application/json", "title": "this document" },
    { "href": "http://data.example.org/collections.html",
      "rel": "alternate", "type": "text/html", "title": "this document as HTML" },
    { "href": "http://schemas.example.org/1.0/foobar.xsd",
      "rel": "describedBy", "type": "application/xml", "title": "XML schema for
Acme Corporation data" }
  ],
  "collections": [
    {
      "name": "buildings",
      "title": "Buildings",
      "description": "Buildings in the city of Bonn.",
      "extent": {
        "spatial": [ 7.01, 50.63, 7.22, 50.78 ],
        "temporal": [ "2010-02-15T12:34:56Z", "2018-03-18T12:11:00Z" ]
      },
      "links": [
        { "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" }
      ]
    }
  ]
}
```

7.12. Feature collection metadata

7.12.1. Operation

Requirement 15	/req/core/sfc-md-op The server SHALL support the HTTP GET operation at the path `/collections/{name}`. The parameter `name` is each property of the same name in the feature collections metadata (JSONPath: `$.collections[*].name`).

7.12.2. Response

Requirement 16	/req/core/sfc-md-success A successful execution of the operation SHALL be reported as a response with a HTTP status code `200`. The content of that response SHALL be the same as the content for this feature collection in the `/collections` response.

7.13. Feature collections

7.13.1. Operation

Requirement 17	/req/core/fc-op For every feature collection identified in the metadata about the feature collection (path `/`), the server SHALL support the HTTP GET operation at the path `/collections/{name}/items`. The parameter `name` is each property of the same name in the feature collections metadata (JSONPath: `$.collections[*].name`).

7.13.2. Parameter limit

Requirement 18	/req/core/fc-limit-definition
	
	Each feature collection operation SHALL support a parameter `limit` with the following characteristics (using an OpenAPI Specification 3.0 fragment):
	
	`name: limit`
	`in: query`
	`required: false`
	`schema:`
	`  type: integer`
	`  minimum: 1`
	`  maximum: 10000`
	`  default: 10`
	`style: form`
	`explode: false`

Permission 1	/per/core/fc-limit-default-maximum The values for `maximum` and `default` in requirement `/req/core/fc-limit-definition` are only examples and MAY be changed.

Requirement 19	/req/core/fc-limit-response-1 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. Only items are counted that are on the first level of the collection. Any nested objects contained within the explicitly requested items SHALL not be counted.

Permission 2	/per/core/fc-limit-response-2 The server MAY return less features than requested (but not more).

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

7.13.3. Parameter bbox

Requirement 20	/req/core/fc-bbox-definition
	
	Each feature collection operation SHALL support a parameter `bbox` with the following characteristics (using an OpenAPI Specification 3.0 fragment):
	
	`name: bbox`
	`in: query`
	`required: false`
	`schema:`
	`  type: array`
	`  minItems: 4`
	`  maxItems: 6`
	`  items:`
	`  type: number`
	`style: form`
	`explode: false`

Requirement 21	/req/core/fc-bbox-response Only features that have a spatial geometry that intersects the bounding box SHALL be part of the result set, if the `bbox` parameter is provided. 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 SHALL be interpreted as WGS84 longitude/latitude (http://www.opengis.net/def/crs/OGC/1.3/CRS84) unless a different coordinate reference system is specified in a parameter `bbox-crs`.

"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 specification does not specify the parameter `bbox-crs`. This parameter will be specified in an

additional part of the WFS 3.0 series.

For WGS84 longitude/latitude the bounding box is in most cases the sequence of minimum longitude, minimum latitude, maximum longitude and maximum latitude. However, in cases where the box spans the antimeridian 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 WGS84 (from 160.6°E to 170°W and from 55.95°S to 25.89°S) would be represented in JSON as `[ 160.6, -55.95, -170, -25.89 ]` and in a query as `bbox=160.6,-55.95,-170,-25.89`.

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

7.13.4. Parameter time

Requirement 22	/req/core/fc-time-definition
	
	Each feature collection operation SHALL support a parameter `time` with the following characteristics (using an OpenAPI Specification 3.0 fragment):
	
	`name: time`
	`in: query`
	`required: false`
	`schema:`
	`  type: string`
	`style: form`
	`explode: false`

Requirement 23	/req/core/fc-time-response Only features that have a temporal geometry that intersects the timestamp or time period SHALL be part of the result set, if the `time` parameter is provided. The temporal information is either a date-time or a period string that adheres to RFC 3339. 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.

"Intersects" means that the time (instant or period) specified in the parameter `time` includes a

timestamp that is part of the temporal geometry of the feature (again, a time instant or period). For time periods this includes the start and end time.

Example 6. A date-time

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

`time=2018-02-12T23%3A20%3A50Z`

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 period, a date-time value would match all features where the timestamp is on that day or within the time period.

Example 7. A period using start and end time

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

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

Example 8. A period using start time and a duration

A duration of 1 month, 6 days, 12 hours, 31 minutes and 12 seconds from February 12, 2018, 00:00:00 GMT:

`time=2018-02-12T00%3A00%3A00Z%2FP1M6DT12H31M12S`

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

For features with a temporal property that is a date or a time period, a time period 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 time.yaml.

7.13.5. Parameters for filtering on feature properties

Recommendation 8	/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, you SHOULD support a parameter with the name of the feature property and with the following characteristics (using an OpenAPI Specification 3.0 fragment):
	
	`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 9. 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 10. An additional parameter to filter buildings based on their name

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

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

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

CAUTION	ISSUE 20 Query parameter name collisions.

7.13.6. Response

Requirement 24	/req/core/fc-response A successful execution of the operation SHALL be reported as a response with a HTTP status code `200`. The response SHALL only include features selected by the request.

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

- The client can request a limit it is interested in.
- The server likely has a default value for the limit, and a maximum limit.
- If the server has any more results available than it returns (the number it returns is less than or equal to the requested/default/maximum limit) then the server will include a link to the next set of results.

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

- If you ask for 10, you will get 0 to 10 (as requested) and a `next` link, if there are more.
- If you don't specify a limit, you will get 0 to 10 (default) and a `next` link, if there are more.
- If you ask for 50000, you might get up to 10000 (server-limited) and a `next` link, if there are

more.

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

Requirement 25	/req/core/fc-links A `200`-response SHALL include the following links: • a link to this response document (relation: `self`), • a link to the response document in every other media type supported by the service (relation: `alternate`).

Recommendation 9	/rec/core/fc-next-1 A `200`-response SHOULD include a link to the next "page" (relation: `next`), if more features have been selected than returned in the response.

Recommendation 10	/rec/core/fc-next-2 Dereferencing a `next` link SHOULD return additional features from the set of selected features that have not yet been returned.

Recommendation 11	/rec/core/fc-next-3 The number of features in a response to a `next` link SHOULD follow the same rules as for the response to the original query and again include a `next` link, if there are more features in the selection that have not yet been returned.

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

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

Another implementation approach is to use an implementation-specific parameter like the `startIndex` parameter that was used in previous versions of WFS (and which may be added again in an extension to this specification).

Permission 3	/per/core/fc-prev A response to a `next` link MAY include a `prev` link to the resource that included the `next` link.

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

Requirement 26	/req/core/fc-rel-type All links SHALL include the `rel` and `type` link parameters.

Requirement 27	/req/core/fc-timeStamp If a property `timeStamp` is included in the response, the value SHALL be set to the time stamp when the response was generated.

Requirement 28	/req/core/fc-numberMatched 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`, `time` or additional filter parameters. A server MAY omit this information in a response, if the information about the number of matching features is not known or difficult to compute.

Requirement 29	/req/core/fc-numberReturned If a property `numberReturned` is included in the response, the value SHALL be identical to the number of features in the response. 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.

CAUTION	Related to ISSUE 8 Define these as headers or include them in the payload? `timeStamp`, for example, may not be needed given the 'Date' HTTP header. For `numberMatched` and `numberReturned` headers do not seem to be a good idea as, for example, `numberReturned` can only be included at the end, if streaming is used.

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

Example 11. 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):

```
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:

```
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:

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

Following the `next` link could return:

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

7.14. Feature

7.14.1. Operation

Requirement 30	/req/core/f-op For every feature in a feature collection (path `/collections/{name}/items`), the service SHALL support the HTTP GET operation at the path `/collections/{name}/items/{id}`. The parameter `name` is each property of the same name in the feature collections metadata (JSONPath: `$.collections[*].name`). `id` is a local identifier of the feature.

Permission 4	/per/core/f-id 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.

CAUTION	ISSUE 47 There are two types of feature identifiers and we need to make sure we distinguish between them.

7.14.2. Response

Requirement 31	/req/core/f-success A successful execution of the operation SHALL be reported as a response with a HTTP status code `200`.

Requirement 32	/req/core/f-links 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`). 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 12. Links

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

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

Chapter 8. Requirements classes for encodings

8.1. Overview

This clause specifies four pre-defined requirements classes for encodings to be used in a WFS. 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 is mandatory and an implementation of the Core requirements class may also decide to use none of them, but to use 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:

- it is not discoverable using the most common mechanism for discovering information, that is the search engines of the Web;
- it 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 to support HTML as an encoding.

Requirements Class	
http://www.opengis.net/spec/wfs-1/3.0/req/html	
Target type	Web service
Dependency	WFS 3.0 Core
Dependency	HTML5
Dependency	Schema.org

Requirement 33	/req/html/definition Every `200`-response of an operation of the server SHALL support the media type `text/html`.

Requirement 34	/req/html/content 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 12	/rec/html/schema-org 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/wfs-1/3.0/req/geojson	
Target type	Web service
Dependency	WFS 3.0 Core
Dependency	GeoJSON

Requirement 35	/req/geojson/definition `200`-responses of the server SHALL support the following media types: • `application/geo+json` for feature collections and features, and • `application/json` for all other resources.

Requirement 36	/req/geojson/content Every `200`-response with the media type `application/geo+json` SHALL be • a GeoJSON FeatureCollection Object for feature collections, and • a GeoJSON Feature Object for features. 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`. The schema of all responses with the media type `application/json` SHALL conform with the JSON Schema specified for the resource in the requirements class WFS 3.0 Core.

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 13. A GeoJSON FeatureCollection Object response

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

```
{
  "type" : "FeatureCollection",
  "links" : [ {
    "href" : "http://data.example.com/collections/buildings/items/?f=json",
    "rel" : "self",
    "type" : "application/geo+json",
    "title" : "this document"
  }, {
    "href" : "http://data.example.com/collections/buildings/items/?f=html",
    "rel" : "alternate",
    "type" : "text/html",
    "title" : "this document as HTML"
  }, {
    "href" :
"http://data.example.com/collections/buildings/items/?f=json&startIndex=10&limit=1
0",
    "rel" : "next",
    "type" : "application/geo+json",
    "title" : "next page"
  } ],
  "timeStamp" : "2018-04-03T14:52:23Z",
  "numberMatched" : 123,
  "numberReturned" : 10,
  "features" : [ {
    "type" : "Feature",
    "id" : "123",
    "geometry" : {
      "type" : "Polygon",
      "coordinates" : [ ... ]
    },
    "properties" : {
      "function" : "residential",
      "floors" : "2",
      "lastUpdate" : "2015-08-01T12:34:56Z"
    }
  }, { ...
  }, {
    "type" : "Feature",
    "id" : "132",
    "geometry" : {
      "type" : "Polygon",
      "coordinates" : [ ... ]
    },
    "properties" : {
      "function" : "public use",
      "floors" : "10",
      "lastUpdate" : "2013-12-03T10:15:37Z"
    }
  } ]
}
```

Example 14. 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 amount 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. It is restricted to data with 2D geometries supported by most tools. In addition, the profile is limited to features that can be stored in a tabular data structure.

Requirements Class	
http://www.opengis.net/spec/wfs-1/3.0/req/gmlsf0	
Target type	Web service

Dependency	WFS 3.0 Core
Dependency	Geography Markup Language (GML), Simple Features Profile, Level 0

Requirement 37	/req/gmlsf0/definition 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 feature collections and features, • application/xml for all other resources.

Requirement 38	/req/gmlsf0/content Every 200-response with the media type application/gml+xml;version=3.2;profile=http://www.opengis.net/def/profile/ogc/2.0/gml-sf0 SHALL be • a WFS 3.0 Core FeatureCollection Object for feature collections, and • a GML 3.2 Feature for features. Every feature SHALL conform to the GML Simple Features Profile, Level 0. The schema of all responses with the media type application/xml SHALL conform with XML Schema …. The links specified in the requirements /req/core/fc-links and /req/core/f-links SHALL be added in …

CAUTION	ISSUE 58 Add and describe XML schema for XML/GML responses. Clarify the mapping of the resources from Core to the XML schema. Add examples.

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.).

Requirements Class	
http://www.opengis.net/spec/wfs-1/3.0/req/gmlsf2	
Target type	Web service

Dependency	WFS 3.0 Core
Dependency	Geography Markup Language (GML), Simple Features Profile, Level 2

Requirement 39	/req/gmlsf2/definition 200-responses of the server SHALL support the following media types: • application/gml+xml;version=3.2;profile=http://www.opengis.net/def/profile/ogc/2.0/gml-sf2 for feature collections and features, • application/xml for all other resources.

Requirement 40	/req/gmlsf0/content Every 200-response with the media type application/gml+xml;version=3.2;profile=http://www.opengis.net/def/profile/ogc/2.0/gml-sf2 SHALL be • a WFS 3.0 Core FeatureCollection Object for feature collections, and • a GML 3.2 Feature for features. Every feature SHALL conform to the GML Simple Features Profile, Level 2. The schema of all responses with the media type application/xml SHALL conform with XML Schema …. The links specified in the requirements /req/core/fc-links and /req/core/f-links SHALL be added in …

CAUTION	ISSUE 58 See the same note in the "Geography Markup Language (GML), Simple Features Profile, Level 0" requirements class above.

Chapter 9. Requirements class "OpenAPI 3.0"

9.1. Basic requirements

The API of servers conforming to this requirements class are defined by an OpenAPI Document.

Requirements Class	
http://www.opengis.net/spec/wfs-1/3.0/req/oas30	
Target type	Web service
Dependency	WFS 3.0 Core
Dependency	OpenAPI Specification 3.0.1

Requirement 41	/req/oas30/oas-definition-1 The service SHALL provide an OpenAPI definition in JSON and HTML at the path `/api` using the media type `application/openapi+json;version=3.0`.

Requirement 42	/req/oas30/oas-definition-2 The JSON representation SHALL conform to the OpenAPI Specification, version 3.0.

CAUTION	Related to ISSUE 90 If we have a rigid path pattern there seems to be no need to add requirements for fixed operationId values. However, if the path pattern would be flexible, maybe we should require specific operationIds for selected resources?

Two example OpenAPI documents are included in Annex B.

Requirement 43	/req/oas30/oas-impl The server SHALL implement all capabilities specified in the OpenAPI definition.

CAUTION	ISSUE 46 Currently, no tool is known to validate that a server implements the API specified in its OpenAPI definition.

9.2. Complete definition

Requirement 44	/req/oas30/completeness The OpenAPI definition SHALL specify for each operation all HTTP Status Codes and Response Objects that the server uses in responses. This includes the successful execution of an operation as well as all error situations that originate from the server.

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

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

Requirement 45	/req/oas30/exceptions-codes For error situations that originate from the server, the API definition SHALL cover all applicable HTTP Status Codes.

CAUTION	ISSUE 45 Listing of all applicable HTTP Status Codes

CAUTION	ISSUE 105 WFS 2.0 defined exception codes and mapped them to HTTP status codes. Should WFS 3.0 do this, too, or should we simply require that http status codes are used correctly?

Example 15. An exception response object definition

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

9.4. Security

Requirement 46	/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.

CAUTION	ISSUE 41 How does a client determine which security protocols/standards/etc. a server supports?

9.5. Features

Recommendation 13	/rec/oas30/f-key-properties 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 in particular helpful, if filter parameters are defined for the collection (see recommendation `/rec/core/fc-filters`).

Chapter 10. Media Types

JSON media types that would typically be used in a WFS 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 WFS 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 WFS would be `text/html`.

The media type for an OpenAPI definition in JSON is `application/openapi+json;version=3.0`.

NOTE	The media type for the OpenAPI definition has not yet been registered with IANA. See https://github.com/OAI/OpenAPI-Specification/issues/110.

Annex A: Conformance Class Abstract Test Suite (Normative)

NOTE	TODO
CAUTION	ISSUE 46 OpenAPI Validation

Annex B: OpenAPI definition example (Informative)

B.1. Overview

This annex includes two complete examples of an OpenAPI definition for a WFS.

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

```
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., WFS 3.0
      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 occured.
        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: Metdata 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 occured.
          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 occured.
          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/time'
    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 occured.
        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 occured.
        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 WGS84 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 WGS84 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
    time:
      name: time
      in: query
      description: >-
        Either a date-time or a period string that adheres to RFC 3339. Examples:

        * A date-time: "2018-02-12T23:20:50Z"
        * A period: "2018-02-12T00:00:00Z/2018-03-18T12:31:12Z" or "2018-02-
12T00:00:00Z/P1M6DT12H31M12S"

        Only features that have a temporal property that intersects the value of
        `time` 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 (name) 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/openapi+json;version=3.0
            title: the API definition
          - href: 'http://data.example.org/conformance'
            rel: conformance
            type: application/json
            title: WFS 3.0 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/wfs-1/3.0/req/core'
          - 'http://www.opengis.net/spec/wfs-1/3.0/req/oas30'
          - 'http://www.opengis.net/spec/wfs-1/3.0/req/html'
          - 'http://www.opengis.net/spec/wfs-1/3.0/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
  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
        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.
            WGS84 longitude/latitude.
          type: array
          items:
            type: string
          default:
            - 'http://www.opengis.net/def/crs/OGC/1.3/CRS84'
    extent:
      type: object
      properties:
        crs:
          description: >-
            Coordinate reference system of the coordinates in the spatial extent
(property `spatial`).
            In the Core, only WGS84 longitude/latitude is supported. Extensions may
support additional
            coordinate reference systems.
          type: string
          enum:
            - 'http://www.opengis.net/def/crs/OGC/1.3/CRS84'
          default: 'http://www.opengis.net/def/crs/OGC/1.3/CRS84'
        spatial:
          description: >-
            West, north, east, south edges of the spatial extent. The minimum and
            maximum values apply to the coordinate reference system WGS84
longitude/latitude
```

```
            that is supported in the Core. If, for example, a projected coordinate
reference
            system is used, the minimum and maximum values need to be adjusted.
          type: array
          minItems: 4
          maxItems: 6
          items:
            type: number
          example:
            - -180
            - -90
            - 180
            - 90
        trs:
          description: >-
            Temporal reference system of the coordinates in the temporal extent
(property `temporal`).
            In the Core, only the Gregorian calendar is supported. Extensions may
support additional
            temporal reference systems.
          type: string
          enum:
            - 'http://www.opengis.net/def/uom/ISO-8601/0/Gregorian'
          default: 'http://www.opengis.net/def/uom/ISO-8601/0/Gregorian'
        temporal:
          description: Begin and end times of the temporal extent.
          type: array
          minItems: 2
          maxItems: 2
          items:
            type: string
            format: dateTime
          example:
            - '2011-11-11T12:22:11Z'
            - '2012-11-24T12:32:43Z'
    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: dateTime
        numberMatched:
          type: integer
          minimum: 0
        numberReturned:
          type: integer
          minimum: 0
    featureGeoJSON:
      type: object
      required:
        - type
        - geometry
        - properties
      properties:
        type:
          type: string
          enum:
            - Feature
        geometry:
          $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.
```

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

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:
```

```
            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., WFS 3.0
      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 occured.
        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: Metdata 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 occured.
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/exception'
          text/html:
            schema:
```

```
              type: string
  '/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 occured.
          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/time'
      - $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'
          text/html:
            schema:
              type: string
      default:
        description: An error occured.
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/exception'
          text/html:
            schema:
              type: string
'/collections/buildings/items.json':
  get:
    summary: 'retrieve features of buildings feature collection in GeoJSON'
    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.\

      This operation returns GeoJSON.
    operationId: getFeaturesJSON
    tags:
      - Features
    parameters:
      - $ref: '#/components/parameters/limit'
      - $ref: '#/components/parameters/bbox'
      - $ref: '#/components/parameters/time'
      - $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 occured.
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/exception'
'/collections/buildings/items/{featureId}':
  get:
    summary: retrieve a feature; use content negotiation to request HTML or GeoJSON
    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 occured.
          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 occured.
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/exception'
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 WGS84 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 WGS84 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
    time:
      name: time
      in: query
      description: >-
        Either a date-time or a period string that adheres to RFC 3339. Examples:

        * A date-time: "2018-02-12T23:20:50Z"
        * A period: "2018-02-12T00:00:00Z/2018-03-18T12:31:12Z" or "2018-02-
12T00:00:00Z/P1M6DT12H31M12S"

        Only features that have a temporal property that intersects the value of
        `time` 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
    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'
    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/openapi+json;version=3.0
            title: the API definition
          - href: 'http://data.example.org/conformance'
            rel: conformance
            type: application/json
            title: WFS 3.0 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/wfs-1/3.0/req/core'
          - 'http://www.opengis.net/spec/wfs-1/3.0/req/oas30'
          - 'http://www.opengis.net/spec/wfs-1/3.0/req/html'
          - 'http://www.opengis.net/spec/wfs-1/3.0/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
  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
        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.
            WGS84 longitude/latitude.
          type: array
          items:
            type: string
          default:
            - 'http://www.opengis.net/def/crs/OGC/1.3/CRS84'
    extent:
      type: object
      properties:
        crs:
          description: >-
            Coordinate reference system of the coordinates in the spatial extent
(property `spatial`).
            In the Core, only WGS84 longitude/latitude is supported. Extensions may
support additional
            coordinate reference systems.
          type: string
          enum:
            - 'http://www.opengis.net/def/crs/OGC/1.3/CRS84'
          default: 'http://www.opengis.net/def/crs/OGC/1.3/CRS84'
        spatial:
          description: >-
            West, north, east, south edges of the spatial extent. The minimum and
            maximum values apply to the coordinate reference system WGS84
```

```
longitude/latitude
            that is supported in the Core. If, for example, a projected coordinate
reference
            system is used, the minimum and maximum values need to be adjusted.
          type: array
          minItems: 4
          maxItems: 6
          items:
            type: number
          example:
            - -180
            - -90
            - 180
            - 90
        trs:
          description: >-
            Temporal reference system of the coordinates in the temporal extent
(property `temporal`).
            In the Core, only the Gregorian calendar is supported. Extensions may
support additional
            temporal reference systems.
          type: string
          enum:
            - 'http://www.opengis.net/def/uom/ISO-8601/0/Gregorian'
          default: 'http://www.opengis.net/def/uom/ISO-8601/0/Gregorian'
        temporal:
          description: Begin and end times of the temporal extent.
          type: array
          minItems: 2
          maxItems: 2
          items:
            type: string
            format: dateTime
          example:
            - '2011-11-11T12:22:11Z'
            - '2012-11-24T12:32:43Z'
    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: dateTime
      numberMatched:
        type: integer
        minimum: 0
      numberReturned:
        type: integer
        minimum: 0
  featureGeoJSON:
    type: object
    required:
      - type
      - geometry
      - properties
    properties:
      type:
        type: string
        enum:
          - Feature
      geometry:
        $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
  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: dateTime
tags:
  - name: Capabilities
    description: >-
      Essential characteristics of this API including information about the
      data.
  - name: Features
    description: >-
      Access to data (features).
```

Annex C: Revision History

Date	Release	Editor	Primary clauses modified	Description
2017-10-09	SNAPSHOT	C. Portele	all	initial version
2017-10-11	SNAPSHOT	C. Portele	all	changes discussed in SWG/PT call on 2017-10-09
2017-12-13	SNAPSHOT	C. Portele	all	address issues #2, #5, #6, #7, #8, #14, #15, #19
2018-01-22	SNAPSHOT	C. Portele	7	add description of the UML diagram
2018-02-01	SNAPSHOT	C. Portele	2,3,5,7	add links to recent issues on GitHub; address issues #31, #32
2018-02-11	SNAPSHOT	C. Portele	2,6,7,8	address issue #25
2018-02-27	SNAPSHOT	C. Portele	all	address issues #3, #9, #12, #22, #23, #24, #44; add links to issues #41, #45, #46, #47
2018-03-04	SNAPSHOT	T. Schaub	7,B	JSON schema fixes #54, #55
2018-03-12	SNAPSHOT for ISO NWIP	C. Portele	all	Updates after the WFS 3.0 Hackathon #59, #61, #62, #63, #64, #69, #72, #77, #78; resolve #4; editorial edits
2018-03-15	SNAPSHOT	J. Amara	7	Uniqueness of feature id #83
2018-03-21	SNAPSHOT	I. Rinne	7	Clarified the requirement /req/core/crs84 #92

Date	Release	Editor	Primary clauses modified	Description
2018-03-28	SNAPSHOT	C. Portele	3,4,7	Temporal support #57, bbox no longer restricted to CRS84 #60, clarify 'collection' #86, clarify feature id constraints #84
2018-04-02	SNAPSHOT	C. Portele	7,B	Clarify 'item' links #81, clean up OpenAPI example in Annex B
2018-04-03	SNAPSHOT	C. Portele	4,5,6,7,8,9	Clean-up asciidoc #100

Annex D: Bibliography

- W3C/OGC: Spatial Data on the Web Best Practices, W3C Working Group Note 28 September 2017, https://www.w3.org/TR/sdw-bp/
- W3C: Data on the Web Best Practices, W3C Recommendation 31 January 2017, https://www.w3.org/TR/dwbp/
- W3C: Data Catalog Vocabulary, W3C Recommendation 16 January 2014, https://www.w3.org/TR/vocab-dcat/
- IANA: Link Relation Types, https://www.iana.org/assignments/link-relations/link-relations.xml