OGC® DOCUMENT: 17-069

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

OGC®
Making location count.

OGC WEB FEATURE SERVICE 3.0: PART 1 - CORE

STANDARD
Implementation

Submission Date: 1000-01-01
Approval Date: 1000-01-01
Publication Date: 1000-01-01
Editor: Clemens Portele, Panagiotis (Peter) A. Vretanos

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.

None of the Intellectual Property or underlying information or technology may be downloaded or otherwise exported or reexported in violation of U.S. export laws and regulations. In addition, you are responsible for complying with any local laws in your jurisdiction which may impact your right to import, export or use the Intellectual Property, and you represent that you have complied with any regulations or registration procedures required by applicable law to make this license enforceable.

Suggested additions, changes and comments on this document are welcome and encouraged. Such suggestions may be submitted using the online change request form on OGC web site: http://portal.opengeospatial.org/public_ogc/change_request.php

Copyright notice

Note

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 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.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

A Web Feature Service (WFS) offers the capability to create, modify and query spatial data on the Web. The WFS standard is a multi-part document. This part specifies the core capabilities that every WFS has to support and is restricted to read-access to spatial 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, and additional coordinate reference systems.

By default, every WFS instance 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 the WFS standard uses a resource architecture and specifies a RESTful service interface consistent with the IETF HTTP/HTTPS RFCs.

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 METHOI	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

RESOURCE	PATH	HTTP METHOI	DOCUMENT REFERENCE
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 SECURITY CONSIDERATIONS

No security considerations have been made for this document.

V SUBMITTING ORGANIZATIONS

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

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

VI SUBMITTERS

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

NAME	AFFILIATION
Chris Holmes	Planet Labs
Clemens Portele *(editor)*	interactive instruments GmbH
Frédéric Houbie	Hexagon
Panagiotis (Peter) A. Vretanos *(editor)*	CubeWerx Inc.

CAUTION

A list of contributors will be added later.

1

SCOPE

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. This standard 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.

2

CONFORMANCE

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.

The *Core* specifies requirements that all WFS have to implement.

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

- HTML,
- GeoJSON,
- Geography Markup Language (GML), Simple Features Profile, Level 0, and
- Geography Markup Language (GML), Simple Features Profile, Level 2.

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

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

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

- OpenAPI specification 3.0.

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

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

Additional capabilities such as support for transactions, complex data structures, rich queries, other coordinate reference systems, subscription/notification, returning aggregated results, etc., may be specified in future parts of 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.

3

NORMATIVE REFERENCES

3 NORMATIVE REFERENCES

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

Open API Initiative: **OpenAPI Specification 3.0.1**, https://github.com/OAI/OpenAPI-Specification/blob/master/versions/3.0.1.md

R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, T. Berners-Lee: IETF RFC 2616, *Hypertext Transfer Protocol — HTTP/1.1*. (1999). https://www.rfc-editor.org/info/rfc2616.

E. Rescorla: IETF RFC 2818, *HTTP Over TLS*. (2000). https://www.rfc-editor.org/info/rfc2818.

G. Klyne, C. Newman: IETF RFC 3339, *Date and Time on the Internet: Timestamps*. (2002). https://www.rfc-editor.org/info/rfc3339.

T. Berners-Lee, R. Fielding, L. Masinter: IETF RFC 3986, *Uniform Resource Identifier (URI): Generic Syntax*. (2005). https://www.rfc-editor.org/info/rfc3986.

M. Nottingham: IETF RFC 5988, *Web Linking*. (2010). https://www.rfc-editor.org/info/rfc5988.

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

H. Butler, M. Daly, A. Doyle, S. Gillies, S. Hagen, T. Schaub: IETF RFC 7946, *The GeoJSON Format*. (2016). https://www.rfc-editor.org/info/rfc7946.

W3C: **HTML5**, W3C Recommendation, http://www.w3.org/TR/html5/

Schema.org: http://schema.org/docs/schemas.html

OGC Web Services Common, https://www.opengeospatial.org/standards/common

ISO/IEC: ISO/IEC DIR 2, *ISO/IEC Directives, Part 2, Rules for the structure and drafting of International Standards*, . ISO, IEC

ISO: ISO 19101-1:2014, *Geographic information — Reference model — Part 1: Fundamentals*. International Organization for Standardization, Geneva (2014). https://www.iso.org/standard/59164.html.

ISO: ISO 19103:2015, *Geographic information — Conceptual schema language*. International Organization for Standardization, Geneva (2015). https://www.iso.org/standard/56734.html.

4

TERMS AND DEFINITIONS

4 TERMS AND DEFINITIONS

This document uses the terms defined in OGC Policy Directive 49, 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 document and OGC documents do not use the equivalent phrases in the ISO/IEC Directives, Part 2.

This document also uses terms defined in the OGC Standard for Modular specifications (OGC 08-131r3), also known as the 'ModSpec'. The definitions of terms such as standard, specification, requirement, and conformance test are provided in the ModSpec.

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

This document uses the terms defined in Sub-clause 5.3 of OGC 06-121r8, which is based on ISO/IEC DIR 2. 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.

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

Note 1 to entry: 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'.

[**SOURCE:** DCAT]

4.2. distribution

represents an accessible form of a **dataset**

EXAMPLE: a downloadable file, an RSS feed or a web service that provides the data.

[**SOURCE:** DCAT]

4.3. **feature**

abstraction of real world phenomena

Note 1 to entry: 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.

[**SOURCE:** ISO 19101-1:2014]

4.4. **feature collection; collection**

a set of **features** from a **dataset**

Note 1 to entry: 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.

5

CONVENTIONS

5 CONVENTIONS

5.1. Open issues

This is a DRAFT version of WFS 3.0, Part 1. This draft is not complete and there are open issues that are still under discussion. These discussion topics are identified 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 candidate WFS 3.0, Part 1, standard in 2019. Criteria to move the candidate standard 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 draft standard 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 servers 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 more easily learn how to use the API. In the best case, documentation will be available in HTML and in a format that can be processed by software to connect to the API.

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

5.6.2. Role of OpenAPI

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

Therefore, the *Core* requirements class 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. 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 API 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 a subset of all possible values are applicable to the server). An example for a constrained range of values is to explicitly specify the supported values of a string parameter or property using an enum.
- 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 — URL of the OpenAPI definition: If the OpenAPI Server Object looks like this:

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

The path "/mypath" in the OpenAPI definition of a 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 for a WFS 3.0 server 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/".*

6

OVERVIEW

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. When the WFS standard was originally designed in the late 1990s and early 2000s this was the state-of-the-art. The WFS 3.0 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 for this version of the WFS standard is modularization. This goal has several facets:

- Clear separation between core requirements and more advanced capabilities. This document specifies the core requirements that are relevant for almost everyone who wants to share or use spatial data on a fine-grained level. Additional capabilities that several communities are using today will be specified as extensions 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. Developers should be able 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, a design goal was 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 implement none of them but implement another encoding instead.

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

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

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

- When solids are used a geometries (e.g. in a 3D city model),
- Geometries that include non-linear curve interpolations that cannot be simplified (e.g., use of arcs in authoritative geometries),
- Geometries 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.

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

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

This part of the WFS 3.0 standard does not provide any guidance on other encodings. The supported encodings, or more precisely the media types of the supported encodings, can be determined from the API definition. The desired encoding is selected using HTTP content negotiation.

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

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

6.3. Examples

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

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

7

REQUIREMENT CLASS "CORE"

7 REQUIREMENT CLASS "CORE"

7.1. Overview

Table 2

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 2818 (HTTP over TLS)
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.

More specifically, 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 that can be used by clients to connect to the server or by development tools to support the implementation of servers and clients. Accessing the `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 WFS server.

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

Accessing the `DatasetDistribution` using HTTP GET returns a `DatasetDistributionResponse`. This response includes a link to each `Collection` in the distribution along with metadata about each collection including:

- A local identifier for the collection that is unique for the server;
- A list of coordinate reference systems (CRS) in which geometries may be returned by the server. The first CRS is the default coordinate reference system (in the *Core*, the default is always WGS 84 with axis order longitude/latitude);
- An optional title and description for the collection;
- An optional extent that can be used to provide an indication of the spatial and temporal extent of the collection — typically derived from the data.

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`. This response 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.

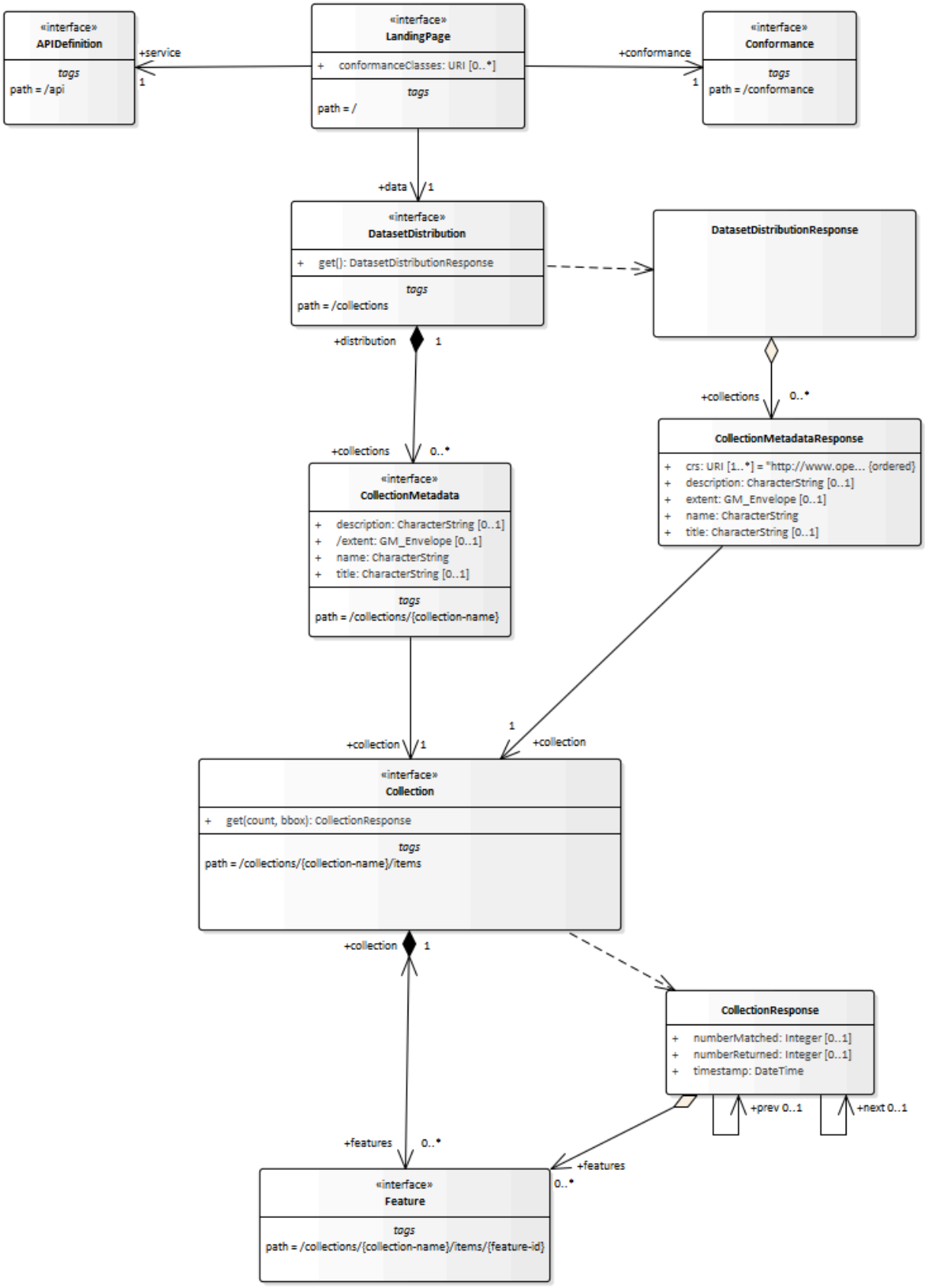

Figure 1 — Resources in the Core requirements class

CAUTION

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

7.2. API landing page

7.2.1. Operation

Table 3

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

7.2.2. Response

Table 4

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'?

```
type: object
required:
  - links
properties:
  links:
    type: array
    items:
```

```
      $ref: https://raw.githubusercontent.com/opengeospatial/WFS_FES/master/cor
e/openapi/schemas/link.yaml
```

Figure 2 — Schema for the landing page

Example — 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" }
  ]
}
```

7.2.3. Error situations

See Clause 7.5.1 for general guidance.

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.

Table 5

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

7.3.2. Response

Table 6

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.

Table 7

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. In other words, 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 data types, etc., but it should not be required to read this standard to access the data via the API.

7.3.3. Error situations

See Clause 7.5.1 for general guidance.

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.

Table 8

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

7.4.2. Response

Table 9

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.

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

Figure 3 — Schema for the list of requirements classes

Example — 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"
  ]
}
```

7.4.3. Error situations

See Clause 7.5.1 for general guidance.

7.5. HTTP 1.1

Table 10

Requirement 7	/req/core/http

The server SHALL conform to HTTP 1.1.
If the server supports HTTPS, the server SHALL also conform to HTTP over TLS.

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

> ***CAUTION***
>
> *ISSUE 115*
> *Currently, the Core only requires support for the GET method. Support for HEAD and OPTIONS for all resources should be considered, too.*

7.5.1. HTTP status codes

Table 11 lists the main HTTP status codes that clients should be prepared to receive.

This includes, for example, support for specific security schemes or URI redirection.

In addition, other error situations may occur in the transport layer outside of the server.

Table 11 — Typical HTTP status codes

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

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

Table 12

Permission 1	/per/core/additional-status-codes Servers MAY support other capabilities of the HTTP protocol and, therefore, MAY return other status codes than those listed in Table 11, too.

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).

Table 13

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

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

Table 14

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 the WFS 3.0 standard does not specify any mandatory encoding, we recommend the following encodings. See Clause 6 (Overview) for a discussion.

Table 15

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.

Table 16

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 RFC.

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

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

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

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 document, how to express and share the location of features in a consistent way is one of the most fundamental aspects of publishing geographic data and it is important to be clear about the coordinate reference system that coordinates are in.

For the reasons discussed in the Best Practices, Web Feature Service 3.0 uses WGS84 longitude and latitude as the default coordinate reference system.

Table 17

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

Table 18

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

Table 19

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

7.11.2. Response

Table 20

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.

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

Figure 4 — Schema for the metadata about feature collections

Table 21

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.

Table 22

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?

Table 23

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`.

Table 24

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?

Table 25

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.

```
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/cor
e/openapi/schemas/link.yaml
  extent:
    $ref: https://raw.githubusercontent.com/opengeospatial/WFS_FES/master/core/
openapi/schemas/extent.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
```

Figure 5 — Schema for the metadata about a feature collection

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

Example — 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 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" }
      ]
    }
  ]
}
```

7.11.3. Error situations

See Clause 7.5.1 for general guidance.

7.12. Feature collection metadata

7.12.1. Operation

Table 26

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

Table 27

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.

7.12.3. Error situations

See Clause 7.5.1 for general guidance.

If the parameter `name` does not exist on the server, the status code of the response will be `404` (see Table 11).

7.13. Feature collections

7.13.1. Operation

Table 28

	/req/core/fc-op
Requirement 17	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

Table 29

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`

Table 30

Permission 2	/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.

Table 31

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.

Table 32

Permission 3	/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

Table 33

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`

Table 34

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 (height or depth): • Lower left corner, coordinate axis 1 • Lower left corner, coordinate axis 2 • Lower left corner, coordinate axis 3 (optional) • Upper right corner, coordinate axis 1 • Upper right corner, coordinate axis 2 • Upper right corner, coordinate axis 3 (optional) The coordinate reference system of the values SHALL be interpreted as 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 standard does not specify requirements for the parameter `bbox-crs`. Those requirements 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 anti-meridian the first value (west-most box edge) is larger than the third value (east-most box edge).

Example — 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

Table 35

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`

Table 36

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 1 — 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 2 — A period using a 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 3 — 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

Table 37

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 1 — An additional parameter to filter buildings based on their function

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

  Default = return all buildings.
required: false
schema:
  type: string
  enum:
    - residential
```

```
    - commercial
    - public use
style: form
explode: false
example: 'function=public+use'
```

Example 2 — An additional parameter to filter buildings based on their name

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

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

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

> ***CAUTION***
>
> *ISSUE 20*
> *Query parameter name collisions.*

7.13.6. Response

Table 38

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 if there are more a `next` link;
- If you don't specify a limit, you will get 0 to 10 (default) and if there are more a `next` link;
- If you ask for 50000, you might get up to 10000 (server-limited) and if there are more a `next` link;
- If you follow the next link from the previous response, you might get up to 10000 additional features and if there are more a `next` link.

Table 39

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`).

Table 40

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.

Table 41

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.

Table 42

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).

Table 43

Permission 4	/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 too complex to implement.

Table 44

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

Table 45

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.

Table 46

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.

Table 47

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 — 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"
```

7.13.7. Error situations

See Clause 7.5.1 for general guidance.

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

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

- If query parameter `limit` is not an integer or not between minimum and maximum;

- if query parameter `bbox` does not have 4 (or 6) numbers or they do not form a bounding box;
- if parameter `time` is not a valid time stamp or time period.

7.14. Feature

7.14.1. Operation

Table 48

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.

Table 49

Permission 5	/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

Table 50

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

Table 51

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

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

7.14.3. Error situations

See Clause 7.5.1 for general guidance.

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

8

REQUIREMENTS CLASSES FOR ENCODINGS

8 REQUIREMENTS CLASSES FOR ENCODINGS

8.1. Overview

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

- HTML
- GeoJSON
- Geography Markup Language (GML), Simple Features Profile, Level 0
- Geography Markup Language (GML), Simple Features Profile, Level 2

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

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

8.2. Requirement Class “HTML”

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

- The data is not discoverable using the most common mechanism for discovering information, that is the search engines of the Web;
- The data can not be viewed directly in a browser — additional tools are required to view the data.

Therefore, sharing data on the Web should include publication in HTML. To be consistent with the Web, it should be done in a way that enables users and search engines to access all data.

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

Table 52

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

Table 53

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

Table 54

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/>`.

Table 55

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.

Table 56

Requirements Class	
http://www.opengis.net/spec/wfs-1/3.0/req/geojson	
Target type	Web service
Dependency	WFS 3.0 Core
Dependency	GeoJSON

Table 57

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.

Table 58

	/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.
Requirement 36	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 1 — A GeoJSON FeatureCollection Object response: In the example below, only the first and tenth feature is shown. Coordinates are not shown.

```
{
  "type" : "FeatureCollection",
  "links" : [ {
    "href" : "http://data.example.com/collections/buildings/items/?f=json",
    "rel" : "self",
    "type" : "application/geo+json",
```

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


Example 2 — A GeoJSON Feature Object response: In the example below, coordinates are not shown.


```
{
  "type" : "Feature",
  "links" : [ {
    "href" : "http://data.example.com/collections/buildings/items/123/?f=json",
    "rel" : "self",
    "type" : "application/geo+json",
    "title" : "this document"
  }, {
    "href" : "http://data.example.com/collections/buildings/items/123/?f=html",
    "rel" : "alternate",
    "type" : "text/html",
    "title" : "this document as HTML"
  }, {
    "href" : "http://data.example.com/collections/buildings/items",
    "rel" : "collection",
    "type" : "application/geo+json",
```

```
    "title" : "the collection document"
  } ],
  "id" : "123",
  "geometry" : {
    "type" : "Polygon",
    "coordinates" : [ ... ]
  },
  "properties" : {
    "function" : "residential",
    "floors" : "2",
    "lastUpdate" : "2015-08-01T12:34:56Z"
  }
}
```

8.4. Requirement Class “Geography Markup Language (GML), Simple Features Profile, Level 0”

In addition to HTML and GeoJSON, a significant volume of feature data is available in XML-based formats, notably GML. Therefore, this standard specifies requirement classes for GML. The Simple Features Profile, Level 0, is the simplest profile of GML and is typically supported by tools. The GML Simple Features Profile is restricted to data with 2D geometries supported by most tools. In addition, the Level 0 profile is limited to features that can be stored in a tabular data structure.

Table 59

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

Table 60

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.

Table 61

Requirement 38	/req/gmlsf0/content Table 62 specifies the XML document root element that the server SHALL return in a `200`-response for each resource. Every feature in a feature collection or feature resource SHALL conform to the GML Simple Features Profile, Level 0 and be substitutable for `gml:AbstractFeature`. The schema of all responses with a root element in the `wfs` namespace SHALL validate against the WFS 3.0 Core XML Schema.

Table 62 — Media types and XML elements for each resource

RESOURCE	PATH	XML ROOT ELEMENT
Landing page	`/`	`wfs:LandingPage`
Conformance classes	`/conformance`	`wfs:ConformsTo`
Feature collections metadata	`/collections`	`wfs:Collections`
Feature collection metadata	`/collections/{name}`	`wfs:Collections`, with just one entry for the collection `name`
Feature collection	`/collections/{name}/items`	`wfs:FeatureCollection`
Feature	`/collections/{name}/items/{fid}`	substitutable for `gml:AbstractFeature`

The namespace prefixes used above and in the WFS 3.0 Core XML Schema are:

- `wfs`: `http://www.opengis.net/wfs/3.0`
- `gml`: `http://www.opengis.net/gml/3.2`
- `atom`: `http://www.w3.org/2005/Atom`
- `xlink`: `http://www.w3.org/1999/xlink`

The API definition resource at path `/api` is not included in Table 62. This requirements class does not prescribe any API definition approach.

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

Annex C has example responses in XML.

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

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

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

Table 63

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

Table 64

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.

Table 65

Requirement 40	/req/gmlsf2/content The requirement `/req/gmlsf0/content` applies, too, with the following changes: • All references to media type `application/gml+xml; version=3.2; profile=http://www.opengis.net/def/profile/ogc/2.0/gml-sf0` are replaced by `application/gml+xml; version=3.2; profile=http://www.opengis.net/def/profile/ogc/2.0/gml-sf2`. • All references to "GML Simple Features Profile, Level 0" are replaced by "GML Simple Features Profile, Level 2".

9

REQUIREMENTS CLASS "OPENAPI 3.0"

9 REQUIREMENTS CLASS "OPENAPI 3.0"

9.1. Basic requirements

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

Table 66

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

Table 67

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`.

CAUTION

ISSUE 117
The OpenAPI media type has not been registered yet with IANA and will likely change. We need to update the media type after registration.

Table 68

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.

Table 69

	/req/oas30/oas-impl
Requirement 43	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

Table 70

	/req/oas30/completeness
Requirement 44	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), support web cache validation, CORS or that use HTTP redirection will make use of additional HTTP status codes beyond regular codes such as `200` for successful GET requests and `400`, `404` or `500` for error situations. See Clause 7.5.1.

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

9.3. Exceptions

Table 71

	/req/oas30/exceptions-codes
Requirement 45	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*

Example — An exception response object definition

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

9.4. Security

Table 72

	/req/oas30/security
Requirement 46	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

Table 73

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`).

10

MEDIA TYPES

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`.

CAUTION

ISSUE 117
The OpenAPI media type has not been registered yet with IANA and will likely change. We need to update the media type after registration.

A

ANNEX A (INFORMATIVE) ABSTRACT TEST SUITE

A

ANNEX A (INFORMATIVE) ABSTRACT TEST SUITE

CAUTION

ISSUE 112
The Abstract Test Suite is work-in-progress. Currently, only a draft for the Core conformance class is available. The other five conformance classes (HTML, GeoJSON, GML-SF0, GML-SF2, OpenAPI 3.0) are not yet covered.

A.1. Overview

WFS 3.0 is not a Web Service in the traditional sense. Rather, it defines the behavior and content of a set of RESTful micro-services exposed through an Application Programing Interface (API). Compliance testing for WFS 3.0 and similar standards must answer three questions:

1. Are the capabilities advertised through the API Description compliant with the standard?
2. Do the micro-services implement those capabilities as advertised?
3. Do the resources returned by the micro-services meet the structure and content requirements of the standard?

Further complicating the issue, an API may expose micro-services in addition to those defined by the standard. A test engine must be able to traverse the API description document, identify test points, and ignore micro-services which are not to be tested. The process for identifying test points is provided in Annex A.4.3.

A.2. Conventions

The following conventions apply to this Abstract Test Suite:

A.2.1. Path Templates

Path templates are used throughout these test suites. Path templating refers to the usage of curly braces "{}" to mark a section of a URL path that can be replaced using path parameters. The terms used to describe portions of these templates are based on the URL syntax described in IETF RFC 3986.

- scheme: http | https
- authority: DNS name of the server with optional port number
- path: The slash delimited identifier for a resource on the server
- query: query parameters following the "?" character
- fragment: identifies an element within the resource. Preceded by the "#" character

A.2.2. API

Description Document

The WFS 3.0 standard does not mandate a standard format for the API Description Document. However, some form of standard is needed if tests are to be accurately described and implemented. Therefore, this Abstract Test Suite assumes that the API Description document is compliant with OpenAPI 3.0. This Test Suite will be updated if and when an alternative is commonly adopted.

A.2.3. Processing

Security Objects

OpenAPI does not provide a standard way to associate a security requirement with a single server URI. Therefore, WFS 3.0 compliance tests will have to make that association through the runtime challenge-response transaction. At this time the role of the Security Objects should be considered advisory.

Security Requirements can be defined at both the OpanAPI root level and at the Operation Object level. The following rules should be followed to understand the scope of a Security Requirement:

- The Security Requirements defined at the root level are the default requirements for all operations and servers.
- If Security Requirements are defined at the Operation level, then those Requirements, and not the ones defined at the OpenAPI level, shall be used with that operation.

- An empty set of Security Requirements at the Operation level indicates that there are no security requirements for that operation.

Note: this allows operations to opt-out of security requirements defined at the OpenAPI level.

A.2.4. Parameters

The following observations apply for WFS 3.0 parameters:

1. WFS 3.0 does not use cookies.
2. Query parameters follow common Web practice
3. Header parameters are restricted to custom headers
4. For path parameters, the name of the parameter must match the name of the variable in the path template in the path object

Parameters are defined at the Path Item and Operation level. Parameters defined at the Path Item level must apply to all operations under that Path item. These parameters may be modified at the Operation level but they may not be removed.

A.2.5. Testable Paths

A testable path is a path which corresponds to one of the paths defined in the WFS 3.0 specification. There are three alternatives for making this determination:

1. The path URI matches – this is the simplest approach but may be subject to error
2. Use mandatory tags in the tags field of the Operation Object
3. Use standardized operation ids for the operationId field of the Operation Object

A testable path is validated against the rules for that path. At a minimum that includes:

1. Building a list of all parameters which are defined in the standard
2. Validate that the mandatory parameters are present and required
3. Validate type, format, etc. for each parameter in the list.
4. Validate that there are no mandatory parameters which are not on the list.

A.3. Requirements Trace Matrix

Table A.1

Requirement 1: API Landing Page Operation
The server SHALL support the HTTP GET operation at the path /.
Tests: A.4.2.1

Requirement 2: API Landing Page Response
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')
Tests: A.4.2.2

Requirement 3: API Definition Operation
The server SHALL support the HTTP GET operation at the path /api.
Tests: A.4.2.3

Requirement 4: API Definition Response
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.
Tests: A.4.2.3, A.4.2.4

Requirement 5: Conformance Class Operation
The server SHALL support the HTTP GET operation at the path /conformance.
Tests: A.4.4.2

Requirement 6: Conformance Class Response
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.
Tests: A.4.4.3

Requirement 7: HTTP 1.1
The server SHALL conform to HTTP 1.1.
If the server supports HTTPS, the server SHALL also conform to HTTP over TLS.
Tests: A.4.1.1

Requirement 8: Coordinate Reference Systems
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)
Tests: A.4.1.2

Requirement 9: Feature Collections Metadata Operation
The server SHALL support the HTTP GET operation at the path /collections.
Tests: A.4.4.4

Requirement 10: Feature Collections Metadata Response
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.
Tests: A.4.4.5

Requirement 11: Feature Collections Metadata 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.
Tests: A.4.4.5

Requirement 12: Feature Collections Metadata Items
For each feature collection in this distribution of the dataset, an item SHALL be provided in the property collections.
Tests: A.4.4.5, A.4.4.6

Requirement 13: Feature Collections Metadata 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.
Tests: A.4.4.6

Requirement 14: Feature Collections Metadata 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.
Tests: A.4.4.6

Requirement 15: Feature Collection Metadta Operation
The server SHALL support the HTTP GET operation at the path /collections/{name}. name is the property of the same name in the feature collections metadata.
Tests: A.4.4.7

Requirement 16: Feature Collection Metadta Response
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.
Tests: A.4.4.8

Requirement 17: Feature Collection Operation
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 where {name} is the property of the same name in the feature collections metadata.
Tests: A.4.4.9

Requirement 18: Feature Collection Operation Limit Parameter
Each feature collection operation SHALL support a parameter limit with the following characteristics (using an OpenAPI Specification 3.0 fragment):

Tests: A.4.4.11

Requirement 19: Feature Collection Operation Limit Parameter Response
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.
Tests: A.4.4.11

Requirement 20: Feature Collection Operation BoundingBox Parameter
Each feature collection operation SHALL support a parameter bbox with the following characteristics (using an OpenAPI Specification 3.0 fragment):
Tests: A.4.4.12

Requirement 21: Feature Collection Operation BoundingBox Parameter 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 (height or depth):
— Lower left corner, coordinate axis 1
— Lower left corner, coordinate axis 2
— Lower left corner, coordinate axis 3 (optional)
— Upper right corner, coordinate axis 1
— Upper right corner, coordinate axis 2
— Upper right corner, coordinate axis 3 (optional)
The coordinate reference system of the values SHALL be interpreted as 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.
Tests: A.4.4.12

Requirement 22: Feature Collection Operation Time Parameter
Each feature collection operation SHALL support a parameter time with the following characteristics (using an OpenAPI Specification 3.0 fragment):
Tests: A.4.4.13

Requirement 23: Feature Collection Operation Time Parameter 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 RFC3339.
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.
Tests: A.4.4.13

Requirement 24: Feature Collection Response
A successful execution of the operation SHALL be reported as a response with a HTTP status code 200.
Tests: A.4.4.10

Requirement 25: Feature Collection Response 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).
Tests: A.4.4.10

Requirement 26: Feature Collection Response Links Parameters
All links SHALL include the rel and type link parameters.
Tests: A.4.4.10

Requirement 27: Feature Collection Response timeStamp
If a property timeStamp is included in the response, the value SHALL be set to the time stamp when the response was generated.
Tests: A.4.4.10

Requirement 28: Feature Collection Response 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.
Tests: A.4.4.10

Requirement 29: Feature Collection Response 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.
Tests: A.4.4.10

Requirement 30: Feature Operation
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} where {name} is the property of the same name in the feature collection metadata and {id} is a local identifier of the feature.
Tests: A.4.4.14

Requirement 31: Feature Opearation Response
A successful execution of the operation SHALL be reported as a response with a HTTP status code 200.
Tests: A.4.4.15

Requirement 32: Feature Operation Response 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.
Tests: A.4.4.15

A.4. Abstract Test

The Test Approach used in the WFS 3.0 Abstract Test Suite includes four steps:

1. Identify the test points
2. Verify that API descriptions of the test points comply with the WFS 3.0 standard
3. Verify that the micro-services at each test point behave in accordance with the WFS 3.0 standard.
4. Verify that the resources returned at each test point are in accordance with the WFS 3.0 standard and any referenced content standard.

Identification of test points is a new requirement with WFS 3.0. Since an API is not a Web Service, there may be RESTful endpoints advertised which are not intended to be targets of the compliance testing. Annex A.4.2 describes the process for crawling the API Description document and extracting those URLs which should be tested as well as the path(s) they should be tested with. The concatenation of a Server URL with a path forms a test point.

Annex A.4.3 describes how the test points are exercised to determine compliance with the WFS 3.0 standard.

A.4.1. General Tests

A.4.1.1. HTTP 1.1

A.4.1.1.1. a) Test Purpose:

Validate that the WFS services advertised through the API conform with HTTP 1.1.

A.4.1.1.2. b) Pre-conditions:

none

A.4.1.1.3. c) Test Method:

1. Build all requests using the HTTP 1.1 protocol.
2. Validate that all responses comply with the HTTP 1.1 protocol

A.4.1.1.4. d) References:

Requirement 7

A.4.1.2. Coordinate Reference Systems

A.4.1.2.1. a) Test Purpose:

Validate that all spatial geometries provided through a WFS service are in the CRS84 spatial reference system unless otherwise requested by the client.

A.4.1.2.2. b) Pre-conditions:

none

A.4.1.2.3. c) Test Method:

1. Do not specify a coordinate reference system in any request. All spatial data should be in the CRS84 reference system.
2. Validate retrieved spatial data using the CRS84 reference system.

A.4.1.2.4. d) References:

Requirement 8

A.4.2. Retrieve the API Description

A.4.2.1. Landing Page Retrieval

A.4.2.1.1. a) Test Purpose:

Validate that a landing page can be retrieved from the expected location.

A.4.2.1.2. b) Pre-conditions:

- A URL to the server hosting the landing page is known.
- The test client can authenticate to the server.
- The test client has sufficient privileges to access the landing page.

A.4.2.1.3. c) Test Method:

1. Issue an HTTP GET request to the URL {root}/
2. Validate that a document was returned with a status code 200
3. Validate the contents of the returned document using test Annex A.4.2.2

A.4.2.1.4. d) References:

Requirement 1

A.4.2.2. Landing Page Validation

A.4.2.2.1. a) Test Purpose:

Validate that the landing page complies with the require structure and contents.

A.4.2.2.2. b) Pre-conditions:

- The landing page has been retrieved from the server

A.4.2.2.3. c) Test Method:

1. Validate the landing page against the root.yaml schema
2. Validate that the landing page includes a “service” link to API Definition
3. Validate that the landing page includes a “conformance” link to the conformance class document
4. Validate that the landing page includes a “data” link to the WFS contents.

A.4.2.2.4. d) References:

Requirement 2

A.4.2.3. OpenAPI Document Retrieval

Note: The URI for the API definition is provided through the landing page. However, that does not mean that the API definition resides on the same server as the landing page. Test clients should be prepared for a WFS 3.0 implementation which is distributed across multiple servers.

A.4.2.3.1. a) Test Purpose:

Validate that the API Definition document can be retrieved from the expected location.

A.4.2.3.2. b) Pre-conditions:

- A URL to the server hosting the API Definition document is known.
- The test client can authenticate to the server.
- The test client has sufficient privileges to assess the API Definition document.

A.4.2.3.3. c) Test Method:

1. Issue an HTTP GET request to the URL {server}/api
2. Validate that a document was returned with a status code 200
3. Validate the contents of the returned document using test Annex A.4.2.4

A.4.2.3.4. d) References:

Requirements 3 and 4

A.4.2.4. API Definition Validation

A.4.2.4.1. a) Test Purpose:

Validate that the API Definition page complies with the require structure and contents.

A.4.2.4.2. b) Pre-conditions:

- The API Definition document has been retrieved from the server

A.4.2.4.3. c) Test Method:

1. Validate the API Definition document against the OpenAPI 3.0 schema
2. Identify the Test Points as described in test Annex A.4.3
3. Process the API Definition document as described in test Annex A.4.4

A.4.2.4.4. d) References:

Requirement 4

A.4.3. Identify the Test Points

Identification of the test points is a pre-condition to performing a compliance test. This process starts with Annex A.4.3.1.

A.4.3.1. Identify Test Points:

A.4.3.1.1. a) Purpose:

To identify the test points associated with each Path in the OpenAPI document

A.4.3.1.2. b) Pre-conditions:

- An OpenAPI document has been obtained
- A list of URLs for the servers to be included in the compliance test has been provided
- A list of the paths specified in the WFS 3.0 specification

A.4.3.1.3. c) Method:

FOR EACH paths property in the OpenAPI document If the path name is one of those specified in the WFS 3.0 specification Retrieve the Server URIs using Annex A.4.3.2. FOR EACH Server

URI Concatenate the Server URI with the path name to form a test point. Add that test point to the list.

A.4.3.1.4. d) References:

None

A.4.3.2. Identify Server URIs:

A.4.3.2.1. a) Purpose:

To identify all server URIs applicable to an OpenAPI Operation Object

A.4.3.2.2. b) Pre-conditions:

- Server Objects from the root level of the OpenAPI document have been obtained
- A Path Item Object has been retrieved
- An Operation Object has been retrieved
- The Operation Object is associated with the Path Item Object
- A list of URLs for the servers to be included in the compliance test has been provided

A.4.3.2.3. c) Method:

1) Identify the Server Objects which are in-scope for this operation

- IF Server Objects are defined at the Operation level, then those and only those Server Objects apply to that Operation.
- IF Server Objects are defined at the Path Item level, then those and only those Server Objects apply to that Path Item.
- IF Server Objects are not defined at the Operation level, then the Server Objects defined for the parent Path Item apply to that Operation.
- IF Server Objects are not defined at the Path Item level, then the Server Objects defined for the root level apply to that Path.
- IF no Server Objects are defined at the root level, then the default server object is assumed as described in the OpenAPI specification.

2) Process each Server Object using Annex A.4.3.3.

3) Delete any Server URI which does not reference a server on the list of servers to test.

A.4.3.2.4. d) References:

None

A.4.3.3. Process Server Object:

A.4.3.3.1. a) Purpose:

To expand the contents of a Server Object into a set of absolute URIs.

A.4.3.3.2. b) Pre-conditions:

- A Server Object has been retrieved

A.4.3.3.3. c) Method:

Processing the Server Object results in a set of absolute URIs. This set contains all of the URIs that can be created given the URI template and variables defined in that Server Object.

1. If there are no variables in the URI template, then add the URI to the return set.
2. For each variable in the URI template which does not have an enumerated set of valid values:
    - generate a URI using the default value,
    - add this URI to the return set,
    - flag this URI as non-exhaustive
3. For each variable in the URI template which has an enumerated set of valid values:
    - generate a URI for each value in the enumerated set,
    - add each generated URI to the return set.
4. Perform this processing in an iterative manner so that there is a unique URI for all possible combinations of enumerated and default values.

5. Convert all relative URIs to absolute URIs by rooting them on the URI to the server hosting the OpenAPI document.

A.4.3.3.4. d) References:

None

A.4.4. Processing the OpenAPI Document

A.4.4.1. Validate /api path

A.4.4.1.1. a) Test Purpose:

Validate the /api path

A.4.4.1.2. b) Pre-conditions:

- Path = /api
- An API Definition document has been retrieved from the server
- A /api path in the OpenAPI document advertises an additional OpenAPI document which may contain additional information about the API.

A.4.4.1.3. c) Test Method:

An OpenAPI document may contain a /api path for a number of reasons including:

- The path points back to this document
- The path indicates an alternate encoding of the API Description
- The path indicates an access point controlled by another authentication scheme.

At this point, none of those cases are addressed through this test suite.

A.4.4.1.4. d) References:

none

A.4.4.2. Validate Conformance Operation

A.4.4.2.1. a) Test Purpose:

Validate that Conformance Operation behaves as required.

A.4.4.2.2. b) Pre-conditions:

- Path = /conformance

A.4.4.2.3. c) Test Method:

DO FOR each /conformance test point

- Issue an HTTP GET request using the test point URI
- Go to test A.4.4.3.

A.4.4.2.4. d) References:

Requirement 5

A.4.4.3. Validate Conformance Operation Response

A.4.4.3.1. a) Test Purpose:

Validate the Response to the Conformance Operation.

A.4.4.3.2. b) Pre-conditions:

- Path = /conformance
- A Conformance document has been retrieved

A.4.4.3.3. c) Test Method:

1. Validate the retrieved document against the classes.yaml schema.
2. Record all reported compliance classes and associate that list with the test point. This information will be used in latter tests.

A.4.4.3.4. d) References:

Requirement 6

A.4.4.4. Validate the Feature Collections Metadata Operation

A.4.4.4.1. a) Test Purpose:

Validate that the Feature Collections Metadata Operation behaves as required

A.4.4.4.2. b) Pre-conditions:

- Path = /collections

A.4.4.4.3. c) Test Method:

DO FOR each /collections test point

- Issue an HTTP GET request using the test point URI
- Go to test A.4.4.5

A.4.4.4.4. d) References:

Requirement 9

A.4.4.5. Validate the Feature Collections Metadata Operation Response

A.4.4.5.1. a) Test Purpose:

Validate that response to the Feature Collection Metadata Operation.

A.4.4.5.2. b) Pre-conditions:

- A Feature Collection Metadata document has been retrieved

A.4.4.5.3. c) Test Method:

1. Validate the retrieved document against the content.yaml schema.
2. Validate that the retrieved document includes links for:
    - Itself
    - Alternate encodings of this document in every other media type as identified by the compliance classes for this server.
3. Validate that each link includes a rel and type parameter
4. Validate that the returned document includes a collections property for each collection in the dataset.
5. For each collections property, validate the metadata for that collection using test A.4.4.6

A.4.4.5.4. d) References:

Requirements 10, 11, and 12

A.4.4.6. Validate a Collections Metadata document

A.4.4.6.1. a) Test Purpose:

Validate a Collections Metadata document.

A.4.4.6.2. b) Pre-conditions:

- A Collection metadata document has been retrieved.

A.4.4.6.3. c) Test Method:

1. Validate the collection metadata against the collectionInfo.yaml schema
2. Validate that the collection metadata document includes links for:
   - Itself
   - Alternate encodings of this document in every other media type as identified by the compliance classes for this server.
3. Validate that each link includes a rel and type parameter
4. Validate the extent property if it is provided
5. Retrieve the collection using the name property and test A.4.4.7.

A.4.4.6.4. d) References:

Requirement 12, 13, 14

A.4.4.7. Validate the Feature Collection Metadata Operation

A.4.4.7.1. a) Test Purpose:

Validate that the Feature Collection Metadata Operation behaves as required

A.4.4.7.2. b) Pre-conditions:

- A feature collection name is provided by test A.4.4.6
- Path = /collections/{name}

A.4.4.7.3. c) Test Method:

DO FOR each /collections{name} test point

- Issue an HTTP GET request using the test point URI
- Go to test A.4.4.8

A.4.4.7.4. d) References:

Requirement 15

A.4.4.8. Validate the Feature Collection Metadata Operation Response

A.4.4.8.1. a) Test Purpose:

Validate that response to the Feature Collection Metadata Operation.

A.4.4.8.2. b) Pre-conditions:

- A Feature Collection Metadata document has been retrieved

A.4.4.8.3. c) Test Method:

1. Validate the retrieved document against the collectionInfo.yaml schema.
2. Validate that this is the same document as that processed in Test A.4.4.6

A.4.4.8.4. d) References:

Requirement 16

A.4.4.9. Validate the Get Features Operation

A.4.4.9.1. a) Test Purpose:

Validate that the Get Features Operation behaves as required.

A.4.4.9.2. b) Pre-conditions:

- A feature collection name is provided by test A.4.4.6
- Path = /collections/{name}/items

A.4.4.9.3. c) Test Method:

DO FOR each /collections{name}/items test point

- Issue an HTTP GET request using the test point URI
- Go to test A.4.4.10

A.4.4.9.4. d) References:

Requirement 17

A.4.4.10. Validate the Get Features Operation Response

A.4.4.10.1. a) Test Purpose:

Validate the Get Feature Operation Response.

A.4.4.10.2. b) Pre-conditions:

- A collection of Features has been retrieved

A.4.4.10.3. c) Test Method:

1. Validate the structure of the response as follows:
   - For HTML use TBD
   - For GeoJSON use featureCollectionGeoJSON.yaml
   - For GML use featureCollectionGML.yaml
2. Validate that the following links are included in the response document:
   - To itself
   - Alternate encodings of this document in every other media type as identified by the compliance classes for this server.
3. Validate that each link includes a rel and type parameter.
4. If a property timeStamp is included in the response, validate that it is close to the current time.
5. If a property numberReturned is included in the response, validate that the number is equal to the number of features in the response.
6. If a property numberMatched is included in the response, iteratively follow the next links until no next link is included and count the aggregated number of features returned in all responses during the iteration. Validate that the value is identical to the numberReturned stated in the initial response.

A.4.4.10.4. d) References:

Requirements 24, 25, 26, 27, 28 and 29

A.4.4.11. Limit Parameter

A.4.4.11.1. a) Test Purpose:

Validate the proper handling of the limit parameter.

A.4.4.11.2. b) Pre-conditions:

- Tests A.4.4.9 and A.4.4.10 have completed successfully.

A.4.4.11.3. c) Test Method:

1. Verify that the OpenAPI document correctly describes the limit parameter for the Get Features operation.
2. Repeat Test A.4.4.9 using different values for the limit parameter.
3. For each execution of Test A.4.4.9, repeat Test A.4.4.10 to validate the results.

A.4.4.11.4. d) References:

Requirements 18 and 19

A.4.4.12. Bounding Box Parameter

A.4.4.12.1. a) Test Purpose:

Validate the proper handling of the bbox parameter.

A.4.4.12.2. b) Pre-conditions:

- Tests A.4.4.9 and A.4.4.10 have completed successfully.

A.4.4.12.3. c) Test Method:

1. Verify that the OpenAPI document correctly describes the bbox parameter for the Get Features operation.
2. Repeat Test A.4.4.9 using different values for the bbox parameter. These should include test cases which cross the meridian, equator, 180^{O} longitude, and polar regions.
3. For each execution of Test A.4.4.9, repeat Test A.4.4.10 to validate the results.

A.4.4.12.4. d) References:

Requirements 20 and 21

A.4.4.13. Time Parameter

A.4.4.13.1. a) Test Purpose:

Validate the proper handling of the time parameter.

A.4.4.13.2. b) Pre-conditions:

- Tests A.4.4.9 and A.4.4.10 have completed successfully.

A.4.4.13.3. c) Test Method:

1. Verify that the OpenAPI document correctly describes the time parameter for the Get Features operation.
2. Repeat Test A.4.4.9 using different values for the time parameter.
3. For each execution of Test A.4.4.9, repeat Test A.4.4.10 to validate the results.

A.4.4.13.4. d) References:

Requirements 22 and 23

A.4.4.14. Get Feature Operation

A.4.4.14.1. a) Test Purpose:

Validate that the Get Feature Operation behaves as required.

A.4.4.14.2. b) Pre-conditions:

- A feature collection name is provided by test A.4.4.6
- A feature identifier is provided by test A.4.4.10
- Path = /collections/{name}/items/(id} where {id} = the feature identifier

A.4.4.14.3. c) Test Method:

DO FOR each /collections{name}/items/{id} test point

- Issue an HTTP GET request using the test point URI
- Go to test A.4.4.15

A.4.4.14.4. d) References:

Requirement 30

A.4.4.15. Validate the Get Feature Operation Response

A.4.4.15.1. a) Test Purpose:

Validate the Get Feature Operation Response.

A.4.4.15.2. b) Pre-conditions:

- The Feature has been retrieved from the server.

A.4.4.15.3. c) Test Method:

1. Validate the structure of the response as follows:
    - For HTML use TBD
    - For GeoJSON use featureGeoJSON.yaml
    - For GML use featureGML.yaml
2. Validate that the following links are included in the response document:
    - To itself
    - To the Feature Collection which contains this Feature
    - Alternate encodings of this document in every other media type as identified by the compliance classes for this server.

A.4.4.15.4. d) References:

Requirements 31 and 32

B

ANNEX B (INFORMATIVE) OPENAPI DEFINITION EXAMPLE

B

ANNEX B (INFORMATIVE) OPENAPI DEFINITION EXAMPLE

B.1. Overview

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

The first example (Annex B.2) 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 (Annex B.3) 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).
```

Figure B.1

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
```

```yaml
tags:
  - name: Capabilities
    description: >-
      Essential characteristics of this API including information about the
      data.
  - name: Features
    description: >-
      Access to data (features).
```

Figure B.2

C

ANNEX C (INFORMATIVE) XML EXAMPLES

C

ANNEX C (INFORMATIVE) XML EXAMPLES

C.1. Overview

This annex includes examples of XML/GML responses to illustrate how the OpenAPI fragments used to define the requirements for the Core requirements class are expressed in XML using the WFS 3.0 Core XML Schema.

C.2. A Landing page

```
<?xml version="1.0" encoding="UTF-8"?>
<LandingPage
   service="WFS"
   version="3.0.0"
   xmlns="http://www.opengis.net/wfs/3.0"
   xmlns:atom="http://www.w3.org/2005/Atom"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://www.opengis.net/wfs/3.0 ../wfs.xsd">
   <atom:link rel="self"
              type="application/json"
              title="This Document"
              href="http://www.acme.com/3.0/wfs?f=application%2Fjson"/>
   <atom:link rel="alternate"
              type="application/xml"
              title="This Document as XML"
              href="http://www.acme.com/3.0/wfs?f=application%2Fxml"/>
   <atom:link rel="alternate"
              type="text/html"
              title="This Document as HTML"
              href="http://www.acme.com/3.0/wfs?f=text%2Fhtml"/>
   <atom:link rel="service"
              type="application/json"
              title="API definition for this enpoint as JSON"
              href="http://www.acme.com/3.0/wfs/api?f=application%2Fjson"/>
   <atom:link rel="service"
              type="application/vnd.ogc_wfs+xml"
              title="API definition for this endpoint as XML"
              href="http://www.acme.com/3.0/wfs/api?f=application%2Fvnd.ogc_wfs
%2Bxml"/>
   <atom:link rel="conformance"
```

```
                type="application/json"
                title="Conformance Declaration as JSON"
                href="http://www.acme.com/3.0/wfs/conformance?f=application
%2Fjson"/>
    <atom:link rel="conformance"
                type="application/xml"
                title="Conformance Declaration as XML"
                href="http://www.acme.com/3.0/wfs/conformance?f=application
%2Fxml"/>
    <atom:link rel="conformance"
                type="text/html"
                title="Conformance Declaration as HTML"
                href="http://www.acme.com/3.0/wfs/conformance?f=text%2Fhtml"/>
    <atom:link rel="collections"
                type="application/json"
                title="Collections Metadata as JSON"
                href="http://www.acme.com/3.0/wfs/collections?f=application
%2Fjson"/>
    <atom:link rel="collections"
                type="application/xml"
                title="Collections Metadata as XML"
                href="http://www.acme.com/3.0/wfs/collections?f=application
%2Fxml"/>
    <atom:link rel="collections"
                type="text/html"
                title="Collections Metadata as HTML"
                href="http://www.acme.com/3.0/wfs/collections?f=text%2Fhtml"/>
</LandingPage>
```

Figure C.1

C.3. Conformance statements

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

```
<?xml version="1.0" encoding="UTF-8"?>
<ConformsTo
    service="WFS"
    version="3.0.0"
    xmlns="http://www.opengis.net/wfs/3.0"
    xmlns:atom="http://www.w3.org/2005/Atom"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.opengis.net/wfs/3.0 ../wfs.xsd">
    <atom:link href="http://www.opengis.net/spec/wfs-1/3.0/req/core"/>
    <atom:link href="http://www.opengis.net/spec/wfs-1/3.0/req/oas30"/>
    <atom:link href="http://www.opengis.net/spec/wfs-1/3.0/req/html"/>
    <atom:link href="http://www.opengis.net/spec/wfs-1/3.0/req/geojson"/>
    <atom:link href="http://www.opengis.net/spec/wfs-1/3.0/req/gmlsf0"/>
</ConformsTo>
```

Figure C.2

C.4. Feature collections metadata

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

```
<?xml version="1.0" encoding="UTF-8"?>
<Collections
   service="WFS"
   version="3.0.0"
   xmlns="http://www.opengis.net/wfs/3.0"
   xmlns:atom="http://www.w3.org/2005/Atom"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://www.opengis.net/wfs/3.0 ../wfs.xsd ">
   <atom:link rel="self"
         title="This document"
         type="application/json"
         href="http://www.acme.com/3.0/wfs/collections?f=application%2Fjson"/>
   <atom:link rel="alternate"
         title="This document as XML"
         type="text/xml"
         href="http://www.acme.com/3.0/wfs/collections?f=text%2Fxml"/>
   <atom:link rel="alternate"
         title="This document as HTML"
         type="text/html"
         href="http://www.acme.com/3.0/wfs/collections?f=text%2Fhtml"/>
   <atom:link rel="describedBy"
         title="XML Schema for this dataset"
         type="application/xml"
         href="http://www.acme.com/3.0/wfs/collections/schema"/>
   <Collection>
      <Name>aerofacp_1m</Name>
      <Title>Airport Facilities Points</Title>
      <atom:link rel="item"
            title="Airport Facilities Points"
            type="application/geo+json"
            href="http://www.acme.com/3.0/wfs/collections/aerofacp_1m/items?f=
application%2Fvnd.geo%2Bjson"/>
      <atom:link rel="item"
            title="Airport Facilities Points"
            type="application/gml+xml;version=3.2;profile=http://www.opengis.
net/def/profile/ogc/2.0/gml-sf0"
            href="http://www.acme.com/3.0/wfs/collections/aerofacp_1m/items?f=a
pplication%2Fgml%2Bxml%3Bversion%3D3.2%3Bprofile%3Dhttp%3A%2F%2Fwww.opengis.net
%2Fdef%2Fprofile%2Fogc%2F2.0%2Fgml-sf0"/>
      <atom:link rel="alternate"
            title="Airport Facilities Points"
            type="text/html"
            href="http://www.acme.com/3.0/wfs/collections/aerofacp_1m/items?f=
text%2Fhtml"/>
      <atom:link rel="describedBy"
            title="Schema for Airport Facilities Points"
            type="application/xml"
            href="http://www.acme.com/3.0/wfs/collections/aerofacp_1m/schema"/>
      <Extent>
         <Spatial crs="http://www.opengis.net/def/crs/OGC/1.3/CRS84">
            <LowerCorner>-179.878326416016 -54.9311103820801</LowerCorner>
            <UpperCorner>179.339859008789 79.52944183349609</UpperCorner>
         </Spatial>
         <Temporal trs="http://www.opengis.net/def/uom/ISO-8601/0/Gregorian">
            <begin>2017-01-01T00:00:00Z</begin>
```

```
            <end>2017-12-31T23:59:59Z</end>
         </Temporal>
      </Extent>
      <DefaultCRS>http://www.opengis.net/def/crs/OGC/1.3/CRS84</DefaultCRS>
   </Collection>
   <Collection>
      <Name>roadl_1m</Name>
      <Title>Roads</Title>
      <atom:link rel="item"
            title="Roads"
            type="application/geo+json"
            href="http://www.acme.com/3.0/wfs/collections/roadl_1m/items?f=
application%2Fvnd.geo%2Bjson"/>
      <atom:link rel="item"
            title="Roads"
            type="application/gml+xml;version=3.2;profile=http://www.opengis.
net/def/profile/ogc/2.0/gml-sf0"
            href="http://www.acme.com/3.0/wfs/collections/roadl_1m/items?f=ap
plication%2Fgml%2Bxml%3Bversion%3D3.2%3Bprofile%3Dhttp%3A%2F%2Fwww.opengis.net
%2Fdef%2Fprofile%2Fogc%2F2.0%2Fgml-sf0"/>
      <atom:link rel="alternate"
            title="Roads"
            type="text/html"
            href="http://www.acme.com/3.0/wfs/collections/roadl_1m/items?f=text
%2Fhtml"/>
      <atom:link rel="describedBy"
            title="Schema for Roads"
            type="application/xml"
            href="http://www.acme.com/3.0/wfs/collections/roadl_1m/schema"/>
      <Extent>
         <Spatial crs="http://www.opengis.net/def/crs/OGC/1.3/CRS84">
           <LowerCorner>-179.999420166016 -54.88802337646479</LowerCorner>
           <UpperCorner>179.9999 74.740592956543</UpperCorner>
         </Spatial>
         <Temporal trs="http://www.opengis.net/def/uom/ISO-8601/0/Gregorian">
            <begin>2017-01-01T00:00:00Z</begin>
            <end>2017-12-31T23:59:59Z</end>
         </Temporal>
      </Extent>
      <DefaultCRS>http://www.opengis.net/def/crs/OGC/1.3/CRS84</DefaultCRS>
   </Collection>
   <Collection>
      <Name>inwatera_1m</Name>
      <Title>Inland Water Areas</Title>
      <atom:link rel="item"
            title="Inland Water Areas"
            type="application/geo+json"
            href="http://www.acme.com/3.0/wfs/collections/inwatera_1m/items?f=
application%2Fvnd.geo%2Bjson"/>
      <atom:link rel="item"
            title="Inland Water Areas"
            type="application/gml+xml;version=3.2;profile=http://www.opengis.
net/def/profile/ogc/2.0/gml-sf0"
            href="http://www.acme.com/3.0/wfs/collections/inwatera_1m/items?f=a
pplication%2Fgml%2Bxml%3Bversion%3D3.2%3Bprofile%3Dhttp%3A%2F%2Fwww.opengis.net
%2Fdef%2Fprofile%2Fogc%2F2.0%2Fgml-sf0"/>
      <atom:link rel="alternate"
            title="Inland Water Areas"
            type="text/html"
            href="http://www.acme.com/3.0/wfs/collections/inwatera_1m/items?f=
text%2Fhtml"/>
      <atom:link rel="describedBy"
            title="Schema for Inland Water Areas"
```

```
            type="application/xml"
            href="http://www.acme.com/3.0/wfs/collections/inwatera_1m/schema"/>
      <Extent>
         <Spatial crs="http://www.opengis.net/def/crs/OGC/1.3/CRS84">
            <LowerCorner>-179.999420166016 -70.91725158691409</LowerCorner>
            <UpperCorner>179.9999 83.57595062255859</UpperCorner>
         </Spatial>
         <Temporal trs="http://www.opengis.net/def/uom/ISO-8601/0/Gregorian">
            <begin>2017-01-01T00:00:00Z</begin>
            <end>2017-12-31T23:59:59Z</end>
         </Temporal>
      </Extent>
      <DefaultCRS>http://www.opengis.net/def/crs/OGC/1.3/CRS84</DefaultCRS>
   </Collection>
</Collections>
```

Figure C.3

C.5. Feature collection metadata

Only the information for the selected feature collection (roads) is included in the response.

```
<?xml version="1.0" encoding="UTF-8"?>
<Collections
   service="WFS"
   version="3.0.0"
   xmlns="http://www.opengis.net/wfs/3.0"
   xmlns:atom="http://www.w3.org/2005/Atom"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://www.opengis.net/wfs/3.0 ../wfs.xsd ">
   <Collection>
      <Name>roadl_1m</Name>
      <Title>Roads</Title>
      <atom:link rel="item"
            title="Roads"
            type="application/geo+json"
            href="http://www.acme.com/3.0/wfs/collections/roadl_1m/items?f=
application%2Fvnd.geo%2Bjson"/>
      <atom:link rel="item"
            title="Roads"
            type="application/gml+xml;version=3.2;profile=http://www.opengis.
net/def/profile/ogc/2.0/gml-sf0"
            href="http://www.acme.com/3.0/wfs/collections/roadl_1m/items?f=ap
plication%2Fgml%2Bxml%3Bversion%3D3.2%3Bprofile%3Dhttp%3A%2F%2Fwww.opengis.net
%2Fdef%2Fprofile%2Fogc%2F2.0%2Fgml-sf0"/>
      <atom:link rel="alternate"
            title="Roads"
            type="text/html"
            href="http://www.acme.com/3.0/wfs/collections/roadl_1m/items?f=text
%2Fhtml"/>
      <atom:link rel="describedBy"
            title="Schema for Roads"
            type="application/xml"
            href="http://www.acme.com/3.0/wfs/collections/roadl_1m/schema"/>
      <Extent>
         <Spatial crs="http://www.opengis.net/def/crs/OGC/1.3/CRS84">
           <LowerCorner>-179.999420166016 -54.88802337646479</LowerCorner>
           <UpperCorner>179.9999 74.740592956543</UpperCorner>
```

```
         </Spatial>
         <Temporal trs="http://www.opengis.net/def/uom/ISO-8601/0/Gregorian">
            <begin>2017-01-01T00:00:00Z</begin>
            <end>2017-12-31T23:59:59Z</end>
         </Temporal>
      </Extent>
      <DefaultCRS>http://www.opengis.net/def/crs/OGC/1.3/CRS84</DefaultCRS>
   </Collection>
</Collections>
```

Figure C.4

C.6. A feature collection

This response contains 2 features of the airport facilities collection and has a link to the next features.

```
<?xml version="1.0" encoding="UTF-8"?>
<wfs:FeatureCollection
   timeStamp="2018-04-02T15:14:20-04:00"
   numberReturned="2"
   numberMatched="9335"
   xmlns="http://www.acme.com/namespaces/ns1"
   xmlns:wfs="http://www.opengis.net/wfs/3.0"
   xmlns:gml="http://www.opengis.net/gml/3.2"
   xmlns:atom="http://www.w3.org/2005/Atom"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://www.acme.com/namespaces/ns1
       http://www.acme.com/3.0/wfs/collections/aerofacp_1m/schema
       http://www.opengis.net/wfs/3.0 ../wfs.xsd
       http://www.w3.org/2005/Atom
       http://schemas.opengis.net/kml/2.3/atom-author-link.xsd
       http://www.opengis.net/gml/3.2
       http://schemas.opengis.net/gml/3.2.1/gml.xsd">
   <wfs:boundedBy>
      <wfs:Spatial crs="http://www.opengis.net/def/crs/OGC/1.3/CRS84">
         <wfs:LowerCorner>-179.878326416016 -54.9311103820801</wfs:LowerCorner>
         <wfs:UpperCorner>179.339859008789 79.52944183349609</wfs:UpperCorner>
      </wfs:Spatial>
      <wfs:Temporal trs="http://www.opengis.net/def/uom/ISO-8601/0/Gregorian">
         <wfs:begin>2017-01-01T00:00:00Z</wfs:begin>
         <wfs:end>2017-12-31T23:59:59Z</wfs:end>
      </wfs:Temporal>
   </wfs:boundedBy>

   <!-- Link to the next set of features in this result set. -->
   <atom:link rel="next" title="The next set of features in this result set"
      type="application/gml+xml;version=3.2;profile=http://www.opengis.net/def/
profile/ogc/2.0/gml-sf0"
      href="http://www.acme.com/3.0/wfs/qid8cff81438e943620dd813fd56c"/>

   <!-- Other hypermedia controls -->
   <atom:link rel="service" title="The service that offers these features"
      type="application/json" href="http://www.acme.com/3.0/wfs"/>
   <atom:link rel="collection"
      title="The collection of which these feature are members"
      type="application/json"
      href="http://www.acme.com/3.0/wfs/collections/aerofacp_1m"/>
```

```
   <atom:link rel="describedBy" title="The schema for these feature"
      type="application/xml"
      href="http://www.acme.com/3.0/wfs/collections/aerofacp_1m/schema"/>

   <!-- Features in the result set -->
   <wfs:member>
      <atom:link rel="self" title="This feature"
         type="application/gml+xml;version=3.2;profile=http://www.opengis.net/
def/profile/ogc/2.0/gml-sf0"
         href="http://www.acme.com/3.0/wfs/collections/aerofacp_1m/items/1?f=a
pplication%2Fgml%2Bxml%3Bversion%3D3.2%3Bprofile%3Dhttp%3A%2F%2Fwww.opengis.net
%2Fdef%2Fprofile%2Fogc%2F2.0%2Fgml-sf0"/>
      <atom:link rel="alternate" title="This feature as GeoJSON"
         type="application/geo+json"
         href="http://www.acme.com/3.0/wfs/collections/aerofacp_1m/items/1?f=
application%2Fgeo%2Bjson"/>
      <atom:link rel="alternate" title="This feature as HTML" type="text/html"
         href="http://www.acme.com/3.0/wfs/collections/aerofacp_1m/items/1?f=
application%2Ftext%2Bhtml"/>
      <aerofacp_1m gml:id="F1">
         <geometry>
            <gml:Point gml:id="geom1" srsName="http://www.opengis.net/def/crs/
OGC/1.3/CRS84">
               <gml:pos>-176.466049194336 -43.81286239624023</gml:pos>
            </gml:Point>
         </geometry>
         <id>1455</id>
         <f_code>GB005</f_code>
         <iko>NZCI</iko>
         <nam>CHATHAM ISLANDS</nam>
         <na3>NZ58133</na3>
         <use>999</use>
         <zv3>13</zv3>
         <tile_id>434</tile_id>
         <end_id>1</end_id>
      </aerofacp_1m>
   </wfs:member>
   <wfs:member>
      <atom:link rel="self" title="This feature"
         type="application/gml+xml;version=3.2;profile=http://www.opengis.net/
def/profile/ogc/2.0/gml-sf0"
         href="http://www.acme.com/3.0/wfs/collections/aerofacp_1m/items/2?f=a
pplication%2Fgml%2Bxml%3Bversion%3D3.2%3Bprofile%3Dhttp%3A%2F%2Fwww.opengis.net
%2Fdef%2Fprofile%2Fogc%2F2.0%2Fgml-sf0"/>
      <atom:link rel="alternate" title="This feature as GeoJSON"
         type="application/geo+json"
         href="http://www.acme.com/3.0/wfs/collections/aerofacp_1m/items/2?f=
application%2Fgeo%2Bjson"/>
      <atom:link rel="alternate" title="This feature as HTML" type="text/html"
         href="http://www.acme.com/3.0/wfs/collections/aerofacp_1m/items/2?f=
application%2Ftext%2Bhtml"/>
      <aerofacp_1m gml:id="F2">
         <geometry>
            <gml:Point gml:id="geom2" srsName="http://www.opengis.net/def/crs/
OGC/1.3/CRS84">
               <gml:pos>-149.5207672119141 -23.3629035949707</gml:pos>
            </gml:Point>
         </geometry>
         <id>4421</id>
         <f_code>GB005</f_code>
         <iko>NTAT</iko>
         <nam>TUBUAI</nam>
         <na3>FP67494</na3>
```

```
            <use>49</use>
            <zv3>3</zv3>
            <tile_id>397</tile_id>
            <end_id>1</end_id>
         </aerofacp_1m>
      </wfs:member>
   </wfs:FeatureCollection>
```

Figure C.5

D

ANNEX D (INFORMATIVE) REVISION HISTORY

ANNEX D (INFORMATIVE) REVISION HISTORY

Table D.1

DATE	RELEASE	EDITOR	PRIMARY CLAUSES MODIFIED	DESCRIPTION
2017-10-09	3.0.0-SNAPSHOT	C. Portele	all	initial version
2017-10-11	3.0.0-SNAPSHOT	C. Portele	all	changes discussed in SWG/PT call on 2017-10-09
2017-12-13	3.0.0-SNAPSHOT	C. Portele	all	address issues #2, #5, #6, #7, #8, #14, #15, #19
2018-01-22	3.0.0-SNAPSHOT	C. Portele	7	add description of the UML diagram
2018-02-01	3.0.0-SNAPSHOT	C. Portele	2, 3, 5, 7	add links to recent issues on GitHub; address issues #31, #32
2018-02-11	3.0.0-SNAPSHOT	C. Portele	2, 6, 7, 8	address issue #25
2018-02-27	3.0.0-SNAPSHOT	C. Portele	all	address issues #3, #9, #12, #22, #23, #24, #44; add links to issues #41, #45, #46, #47
2018-03-04	3.0.0-SNAPSHOT	T. Schaub	7, B	JSON schema fixes #54, #55
2018-03-12	3.0.0-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	3.0.0-SNAPSHOT	J. Amara	7	Uniqueness of feature id #83
2018-03-21	3.0.0-SNAPSHOT	I. Rinne	7	Clarified the requirement /req/core/crs84 #92
2018-03-28	3.0.0-SNAPSHOT	C. Portele	3, 4, 7	Temporal support #57, bbox no longer restricted to CRS84 #60, clarify 'collection' #86, clarify feature id constraints #84

DATE	RELEASE	EDITOR	PRIMARY CLAUSES MODIFIED	DESCRIPTION
2018-04-02	3.0.0-SNAPSHOT	C. Portele	7, B	Clarify 'item' links #81, clean up OpenAPI example in Annex B
2018-04-03	3.0.0-SNAPSHOT	C. Portele	4 to 9	Clean-up asciidoc #100
2018-04-04	3.0.0-SNAPSHOT	P. Vretanos, C. Portele	8.4, 8.5, C	Clarify XML encoding #58
2018-04-05	3.0.0-SNAPSHOT	C. Heazel	A	Initial version of the Abstract Test Suite #112
2018-04-05	3.0.0-SNAPSHOT	C. Portele	C	Fix axis order in example #113
2018-04-07	3.0.0-SNAPSHOT	C. Portele	7, 9, 10	Add HTTP status code guidance #105, add warning about OpenAPI media type #117
2018-04-07	3.0.0-SNAPSHOT	C. Reed, C. Portele	all	Edits after review #119
2018-04-07	**3.0.0-draft.1**	C. Portele	iv, v	First draft release

BIBLIOGRAPHY

BIBLIOGRAPHY

1. W3C: Data Catalog Vocabulary, W3C Recommendation 16 January 2014, https://www.w3.org/TR/vocab-dcat/

2. W3C: Data on the Web Best Practices, W3C Recommendation 31 January 2017, https://www.w3.org/TR/dwbp/

3. W3C/OGC: Spatial Data on the Web Best Practices, W3C Working Group Note 28 September 2017, https://www.w3.org/TR/sdw-bp/

4. IANA: Link Relation Types, https://www.iana.org/assignments/link-relations/link-relations.xml