OGC® DOCUMENT: 17-069R3

External identifier of this OGC® document: http://www.opengis.net/doc/IS/ogcapi-features-1/1.0

Open Geospatial Consortium

API - FEATURES - PART 1: CORE

STANDARD
Implementation

APPROVED

Version: 1.0
Submission Date: 2019-07-11
Approval Date: 2019-09-09
Publication Date: 2019-10-14
Editor: Clemens Portele, Panagiotis (Peter) A. Vretanos, Charles Heazel

License Agreement

Use of this document is subject to the license agreement at https://www.ogc.org/license

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://ogc.standardstracker.org/

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 RECOMMENDATIONS

ABSTRACT

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

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

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

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

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

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

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

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

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

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

1:Core" and the title in ISO is "Geographic Information — Geospatial API for Features — Part 1: Core."

For simplicity, this document consistently uses:

- "OGC API" to refer to the family of standards for geospatial Web APIs that in ISO is published as "Geospatial API;"
- "OGC API — Features" to refer to the multipart standard for features that in ISO is published as ISO 19168 / "Geographic Information — Geospatial API for Features;" and
- "OGC API — Features — Part 1: Core" to refer to this document that in ISO is published as ISO 19168-1 / "Geographic Information — Geospatial API for Features — Part 1: Core."

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

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

RESOURCE	PATH	HTTP METHOD	DOCUMENT REFERENCE
Landing page	/	GET	Clause 7.2
Conformance declaration	/conformance	GET	Clause 7.4
Feature collections	/collections	GET	Clause 7.13
Feature collection	/collections/{collectionId}	GET	Clause 7.14
Features	/collections/{collectionId}/items	GET	Clause 7.15
Feature	/collections/{collectionId}/items/{featureId}	GET	Clause 7.16

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

In the first approach, clients are implemented with knowledge about this standard and its resource types. The clients navigate the resources based on this knowledge and based on the responses provided by the API. The API definition may be used to determine details, e.g., on filter parameters, but this may not be necessary depending on the needs of the client. These are clients that are in general able to use multiple APIs as long as they implement OGC API Features.

The other approach targets developers that are not familiar with the OGC API standards, but want to interact with spatial data provided by an API that happens to implement OGC API Features. In this case the developer will study and use the API definition — typically an OpenAPI document — to understand the API and implement the code to interact with the API. This

assumes familiarity with the API definition language and the related tooling, but it should not be necessary to study the OGC API standards.

KEYWORDS

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

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

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

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

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

- **Research:** facilitates research into weaknesses and can be leveraged to systematically identify theoretical gaps within CWE.
- **Architectural:** organizes weaknesses according to common architectural security tactics. It is intended to assist architects in identifying potential mistakes that can be made when designing software.
- **Development:** organizes weaknesses around concepts that are frequently used or encountered in software development.

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

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

- Pathname Traversal and Equivalence Errors,
- Channel and Path Errors, and
- Web Problems.

Many of the vulnerabilities described in the CWE are introduced through the HTTP protocol. API designers and developers should be familiar with how the HTTP 1.1 addresses these vulnerabilities. This information can be found in section 15 of IETF RFC 2616.

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

IV.A. Multiple Access Routes

APIs deliver a representation of a resource. OGC APIs can deliver multiple representations (formats) of the same resource. An attacker may find that information which is prohibited in one

representation can be accessed through another. API designers must take care that the access controls on their resources are implemented consistently across all representations. That does not mean that they have to be the same. For example, consider the following.

- HTML vs. GeoTIFF – The HTML representation may consist of a text description of the resource accompanied by a thumbnail image. This has less information than the GeoTIFF representation and may be subject to more liberal access policies.
- Data Centric Security – techniques to embed access controls into the representation itself. A GeoTIFF with Data Centric Security would have more liberal access policies than a GeoTIFF without.

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

IV.B. Multiple Servers

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

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

IV.C. Path Manipulation on GET

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

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

IV.D. Path Manipulation on PUT and POST

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

Many of the resources exposed though an OGC API include hyperlinks to other resources. API clients follow these hyperlinks to access new resources or alternate representations of a

resource. Once a client authenticates to an API, they tend to trust the data returned by that API. However, a resource posted by an attacker could contain hyperlinks which contain an attack. For example, the link to an alternate representation could require the client to re-authenticate prior to passing them on to the original destination. The client sees the representation they asked for and the attacker collects the clients' authentication credentials.

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

V SUBMITTING ORGANIZATIONS

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

- CubeWerx Inc.
- Heazeltech LLC
- Hexagon
- interactive instruments GmbH
- Ordnance Survey
- Planet Labs
- US Army Geospatial Center (AGC)

VI SUBMITTERS

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

NAME	AFFILIATION
Clemens Portele *(editor)*	interactive instruments GmbH
Panagiotis (Peter) A. Vretanos *(editor)*	CubeWerx Inc.
Charles Heazel *(editor)*	Heazeltech LLC
Michael Gordon	Ordnance Survey
Jeff Harrison	US Army Geospatial Center (AGC)
Chris Holmes	Planet Labs
Frédéric Houbie	Hexagon

1 SCOPE

1 SCOPE

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

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

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

2 CONFORMANCE

2 CONFORMANCE

This standard defines six requirements / conformance classes.

The standardization targets of all conformance classes are “Web APIs.”

The main requirements class is:

- Core.

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

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

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

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

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

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

- OpenAPI specification 3.0.

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

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

Additional capabilities such as support for transactions, complex data structures, rich queries, other coordinate reference systems, subscription/notification, returning aggregated results,

etc., may be specified in future parts of the OGC API Features series or as vendor-specific extensions.

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

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 3.0.2**, 2018 https://github.com/OAI/OpenAPI-Specification/blob/master/versions/3.0.2.md

Linda van den Brink, Clemens Portele, Panagiotis (Peter) A. Vretanos: OGC 10-100r3, *Geography Markup Language (GML) simple features profile (with Corrigendum)*. Open Geospatial Consortium (2011). https://portal.ogc.org/files/?artifact_id=42729.

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

W3C: W3C html5, *HTML5*. World Wide Web Consortium http://www.w3.org/TR/html5/.

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

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

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

Klyne, Newman: IETF RFC 3339, *Date and Time on the Internet: Timestamps*. IETF (2002). http://tools.ietf.org/rfc/rfc3339.txt.

Nottingham: IETF RFC 8288, *Web Linking*. IETF (2017). http://tools.ietf.org/rfc/rfc8288.txt.

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

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:** W3C vocab-dcat]

4.2. distribution

represents an accessible form of a **dataset**

Example a downloadable file, an RSS feed or an API.

[**SOURCE:** W3C vocab-dcat]

4.3. feature

abstraction of real world phenomena

Note 1 to entry: For those unfamiliar with the term 'feature', the explanations on Spatial Things, Features and Geometry in the W3C/OGC Spatial Data on the Web Best Practice document provide more detail.

[**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.

4.5. Web API

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

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

[**SOURCE:** DWBP]

5 CONVENTIONS

5 CONVENTIONS

5.1. Identifiers

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

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

5.2. Link relations

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

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

- **alternate:** Refers to a substitute for this context.
- **collection:** The target IRI points to a resource which represents the collection resource for the context IRI.
- **describedBy:** Refers to a resource providing information about the link's context.
- **item:** The target IRI points to a resource that is a member of the collection represented by the context IRI.
- **next:** Indicates that the link's context is a part of a series, and that the next in the series is the link target.
- **license:** Refers to a license associated with this context.
- **prev:** Indicates that the link's context is a part of a series, and that the previous in the series is the link target.
    - This relation is only used in examples.
- **self:** Conveys an identifier for the link's context.
- **service-desc:** Identifies service description for the context that is primarily intended for consumption by machines.
    - API definitions are considered service descriptions.

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

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

- **items:** Refers to a resource that is comprised of members of the collection represented by the link's context.
- **conformance:** Refers to a resource that identifies the specifications that the link's context conforms to.
- **data:** Indicates that the link's context is a distribution of a dataset that is an API and refers to the root resource of the dataset in the API.

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

5.3. Use of HTTPS

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

5.4. HTTP URIs

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

5.5. API definition

5.5.1. General remarks

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

This 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 OGC API family of standards — and will support additional operations, parameters, etc. that are specific to the API or the software tool used to implement the API.

5.5.2. Role of OpenAPI

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

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

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

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

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

5.5.3. References to OpenAPI components in normative statements

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

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

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

- The range of values of a parameter or property may be extended (additional values) or constrained (if a subset of all possible values are applicable to the server). An example for a constrained range of values is to explicitly specify the supported values of a string parameter or property using an enum.
- The default value of a parameter may be changed or added unless a requirement explicitly prohibits this.
- Additional properties may be added to the schema definition of a Response Object.
- Informative text may be changed or added, like comments or description properties.

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

5.5.4. Paths in OpenAPI definitions

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

Example — URL of the OpenAPI definition: If the OpenAPI Server Object looks like this:

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

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

5.5.5. Reusable OpenAPI components

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

6 OVERVIEW

6 OVERVIEW

6.1. Design considerations

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

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

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

- Clear separation between core requirements and more advanced capabilities. This document specifies the core requirements that are relevant for almost everyone who wants to share or use feature data on a fine-grained level. Additional capabilities that several communities are using today will be specified as extensions in additional parts of the OGC API Features series.
- Technologies that change more frequently are decoupled and specified in separate modules ("requirements classes" in OGC terminology). This enables, for example, the use/re-use of new encodings for spatial data or API descriptions.
- Modularization is not just about features are resources, but about providing building blocks for fine-grained access to spatial data that can be used in Web APIs in general. In other words, a server supporting OGC API Features is not intended to implement just a standalone Features API. A corollary of this is that the same Web API may also implement other standards of the OGC API family that support additional resource types; for example, tile resources could provide access to the same features, but organized in a spatial partitioning system; or map resources could process the features and render them as as map images.

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

In the first approach, clients are implemented with knowledge about this standard and its resource types. The clients navigate the resources based on this knowledge and based on the responses provided by the API. The API definition may be used to determine details, e.g., on

filter parameters, but this may not be necessary depending on the needs of the client. These are clients that are in general able to use multiple APIs as long as they implement OGC API Features.

The other approach targets developers that are not familiar with the OGC API standards, but want to interact with spatial data provided by an API that happens to implement OGC API Features. In this case the developer will study and use the API definition — typically an OpenAPI document — to understand the API and implement the code to interact with the API. This assumes familiarity with the API definition language and the related tooling, but it should not be necessary to study the OGC API standards.

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 OGC API Features recommends supporting GeoJSON for encoding feature data, if the feature data can be represented in GeoJSON for the intended use.

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

- When solids are used for geometries (e.g., in a 3D city model),
- Geometries that include non-linear curve interpolations that cannot be simplified (e.g., use of arcs in authoritative geometries),
- Geometries that have to be represented in a coordinate reference system that is not based on WGS 84 longitude/latitude (e.g., an authoritative national reference system), or
- Features that have more than one geometric property.

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

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

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

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

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

6.3. Examples

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

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

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

7 REQUIREMENTS CLASS "CORE"

7 REQUIREMENTS CLASS "CORE"

7.1. Overview

REQUIREMENTS CLASS 1	
OBLIGATION	requirement
TARGET TYPE	Web API
PREREQUISITES	RFC 2616 (HTTP/1.1) RFC 2818 (HTTP over TLS) RFC 3339 (Date and Time on the Internet: Timestamps) RFC 8288 (Web Linking)

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

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

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

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

The `Landing page` provides links to:

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

The `API definition` describes the capabilities of the server that can be used by clients to connect to the server or by development tools to support the implementation of servers and

clients. Accessing the `API definition` using HTTP GET returns a description of the API. The API definition can be hosted on the API server(s) or a separate server.

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

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

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

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

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

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

- A local identifier for the collection that is unique for the dataset;
- A list of coordinate reference systems (CRS) in which geometries may be returned by the server: the first CRS is the default coordinate reference system (in the Core, the default is always WGS 84 with axis order longitude/latitude);
- An optional title and description for the collection;
- An optional extent that can be used to provide an indication of the spatial and temporal extent of the collection — typically derived from the data;
- An optional indicator about the type of the items in the collection (the default value, if the indicator is not provided, is 'feature').

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

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

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

Accessing the `Features` using HTTP GET returns a document consisting of features in the collection. The features included in the response are determined by the server based on the query parameters of the request. To support access to larger collections without overloading the

client, the API supports paged access with links to the next page, if more features are selected than the page size.

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

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

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

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

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

7.2. API landing page

7.2.1. Operation

REQUIREMENT 1	
NORMATIVE STATEMENT	Requirement 1-1 The server SHALL support the HTTP GET operation at the path `/`.

7.2.2. Response

REQUIREMENT 2	
NORMATIVE STATEMENTS	Requirement 2-1 A successful execution of the operation SHALL be reported as a response with a HTTP status code `200`. Requirement 2-2 The content of that response SHALL be based upon the OpenAPI 3.0 schema landingPage.yaml and include at least links to the following resources: • the API definition (relation type 'service-desc' or 'service-doc')

REQUIREMENT 2

- `/conformance` (relation type 'conformance')
- `/collections` (relation type 'data')

Schema for the landing page

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

Listing

Example — Landing page response document

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

7.2.3. Error situations

See HTTP status codes for general guidance.

7.3. API definition

7.3.1. Operation

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

REQUIREMENT 3	
NORMATIVE STATEMENT	Requirement 3-1 The URIs of all API definitions referenced from the landing page SHALL support the HTTP GET method.

PERMISSION 1	
NORMATIVE STATEMENT	Permission 1-1 The API definition is metadata about the API and strictly not part of the API itself, but it MAY be hosted as a sub-resource to the base path of the API, for example, at path `/api`. There is no need to include the path of the API definition in the API definition itself.

Note that multiple API definition formats can be supported.

7.3.2. Response

REQUIREMENT 4	
NORMATIVE STATEMENT	Requirement 4-1 A GET request to the URI of an API definition linked from the landing page (link relations `service-desc` or `service-doc`) with an `Accept` header with the value of the link property `type` SHALL return a document consistent with the requested media type.

RECOMMENDATION 1	
NORMATIVE STATEMENT	Recommendation 1-1 If the API definition document uses the OpenAPI Specification 3.0, the document SHOULD conform to the OpenAPI Specification 3.0 requirements class.

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

The idea is that any OGC API Features implementation can be used by developers that are familiar with the API definition language(s) supported by the server. For example, if an OpenAPI

definition is used, it should be possible to create a working client using the OpenAPI definition. The developer may need to learn a little bit about geometry data types, etc., but it should not be required to read this standard to access the data via the API.

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

7.3.3. Error situations

See HTTP status codes for general guidance.

7.4. Declaration of conformance classes

7.4.1. Operation

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

REQUIREMENT 5	
NORMATIVE STATEMENT	Requirement 5-1 The server SHALL support the HTTP GET operation at the path `/conformance`.

7.4.2. Response

REQUIREMENT 6	
NORMATIVE STATEMENTS	Requirement 6-1 A successful execution of the operation SHALL be reported as a response with a HTTP status code `200`. Requirement 6-2 The content of that response SHALL be based upon the OpenAPI 3.0 schema confClasses.yaml and list all OGC API conformance classes that the server conforms to.

Schema for the list of conformance classes

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

```
    items:
      type: string
```

Listing

Example — Conformance declaration response document: This example response in JSON is for a server that supports OpenAPI 3.0 for the API definition and HTML and GeoJSON as encodings for features.

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

7.4.3. Error situations

See HTTP status codes for general guidance.

7.5. HTTP 1.1

REQUIREMENT 7	
NORMATIVE STATEMENTS	Requirement 7-1 The server SHALL conform to HTTP 1.1. Requirement 7-2 If the server supports HTTPS, the server SHALL also conform to HTTP over TLS.

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

RECOMMENDATION 2	
NORMATIVE STATEMENT	Recommendation 2-1 The server SHOULD support the HTTP 1.1 method HEAD for all resources that support the method GET.

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

Servers implementing CORS will implement the method OPTIONS, too.

7.5.1. HTTP status codes

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

The **Status Codes** listed in Table 2 are of particular relevance to implementors of this standard. Status codes 200, 400, and 404 are called out in API requirements. Therefore, support for these status codes is mandatory for all compliant implementations. The remainder of the status codes in Table 2 are not mandatory, but are important for the implementation of a well functioning API. Support for these status codes is strongly encouraged for both client and server implementations.

Table 2 — Typical HTTP status codes

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 authorized to perform the requested operation on the resource.
`404`	The requested resource does not exist on the server. For example, a path parameter had an incorrect value.
`405`	The request method is not supported. For example, a POST request was submitted, but the resource only supports GET requests.
`406`	The `Accept` header submitted in the request did not support any of the media types supported by the server for the requested resource.
`500`	An internal error occurred in the server.

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

PERMISSION 2	
NORMATIVE STATEMENT	Permission 2-1 Servers MAY support other capabilities of the HTTP protocol and, therefore, MAY return other status codes than those listed in Table 2.

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

7.6. Unknown or invalid query parameters

REQUIREMENT 8	
NORMATIVE STATEMENT	Requirement 8-1 The server SHALL respond with a response with the status code `400`, if the request URI includes a query parameter that is not specified in the API definition.

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

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

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

Listing — OpenAPI schema for additional "free-form" query parameters

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

```
{
  "my_first_parameter": "some value",
  "my_other_parameter": 42
}
```

Listing

In the request URI this would be expressed as `&my_first_parameter=some%20value&my_other_parameter=42`.

REQUIREMENT 9	
NORMATIVE STATEMENT	Requirement 9-1 The server SHALL respond with a response with the status code `400`, if the request URI includes a query parameter that has an invalid value.

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

7.7. Web caching

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

RECOMMENDATION 3	
NORMATIVE STATEMENT	Recommendation 3-1 The service SHOULD support entity tags and the associated headers as specified by HTTP/1.1.

7.8. Support for cross-origin requests

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

RECOMMENDATION 4	
NORMATIVE STATEMENT	Recommendation 4-1 If the server is intended to be accessed from the browser, cross-origin requests SHOULD be supported. Note that support can also be added in a proxy layer on top of the server.

Two common mechanisms to support cross-origin requests are:

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

7.9. Encodings

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

RECOMMENDATION 5	
NORMATIVE STATEMENT	Recommendation 5-1 To support browsing the dataset and its features with a web browser and to enable search engines to crawl and index the dataset, implementations SHOULD consider to support an HTML encoding.

RECOMMENDATION 6	
NORMATIVE STATEMENT	Recommendation 6-1 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.

7.10. String internationalization

If the server supports representing resources in multiple languages, the usual HTTP content negotiation mechanisms apply. The client states its language preferences in the `Accept-Language` header of a request and the server responds with responses that have linguistic text in the language that best matches the requested languages and the capabilities of the server.

RECOMMENDATION 7	
NORMATIVE STATEMENT	Recommendation 7-1 For encodings that support string internationalization, the server SHOULD include information about the language for each string value that includes linguistic text.

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

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

7.11. Coordinate reference systems

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

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

REQUIREMENT 10	
NORMATIVE STATEMENT	Requirement 10-1 Unless the client explicitly requests a different coordinate reference system, all spatial geometries SHALL be in the coordinate reference system http://www.opengis.net/def/crs/OGC/1.3/CRS84 (WGS 84 longitude/latitude) for geometries without height information and http://www.opengis.net/def/crs/OGC/0/CRS84h (WGS 84 longitude/latitude plus ellipsoidal height) for geometries with height information.

Implementations compliant with the Core are not required to support publishing feature geometries in coordinate reference systems other than http://www.opengis.net/def/crs/OGC/1.3/CRS84 (for coordinates without height) or http://www.opengis.net/def/crs/OGC/0/CRS84h (for coordinates with height); i.e., the (optional) third coordinate number is always the height.

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

7.12. Link headers

RECOMMENDATION 8	
NORMATIVE STATEMENT	Recommendation 8-1 Links included in payload of responses SHOULD also be included as `Link` headers in the HTTP response according to RFC 8288, Clause 3. This recommendation does not apply, if there are a large number of links included in a response or a link is not known when the HTTP headers of the response are created.

7.13. Feature collections

7.13.1. Operation

REQUIREMENT 11	
NORMATIVE STATEMENT	Requirement 11-1 The server SHALL support the HTTP GET operation at the path `/collections`.

7.13.2. Response

REQUIREMENT 12	
NORMATIVE STATEMENTS	Requirement 12-1 A successful execution of the operation SHALL be reported as a response with a HTTP status code `200`. Requirement 12-2 The content of that response SHALL be based upon the OpenAPI 3.0 schema collections.yaml.

Schema for the collections resource

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

Listing

REQUIREMENT 13	
NORMATIVE STATEMENTS	Requirement 13-1 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`). Requirement 13-2 All links SHALL include the `rel` and `type` link parameters.

RECOMMENDATION 9	
NORMATIVE STATEMENTS	Recommendation 9-1 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`). Recommendation 9-2 The `type` link parameter SHOULD be provided for each link. This applies to resources that describe to the whole dataset. Recommendation 9-3 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. Recommendation 9-4 Examples for descriptions are: XML Schema, Schematron, JSON Schema, RDF Schema, OWL, SHACL, a feature catalogue, etc.

RECOMMENDATION 10	
NORMATIVE STATEMENTS	Recommendation 10-1 For each feature collection in this distribution of the dataset, the `links` property of the collection SHOULD include an item for each supported encoding with a link to the collection resource (relation: `license`).

RECOMMENDATION 10	
	Recommendation 10-2 Alternatively, if all data shared via the API is available under the same license, the link MAY instead be added to the top-level `links` property of the response. Recommendation 10-3 Multiple links to the license in different content types MAY be provided. At least a link to content type `text/html` or `text/plain` SHOULD be provided.

REQUIREMENT 14	
NORMATIVE STATEMENT	Requirement 14-1 For each feature collection provided by the server, an item SHALL be provided in the property `collections`.

PERMISSION 3	
NORMATIVE STATEMENT	Permission 3-1 To support servers with many collections, servers MAY limit the number of items in the property `collections`.

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

REQUIREMENT 15	
NORMATIVE STATEMENTS	Requirement 15-1 For each feature collection included in the response, the `links` property of the collection SHALL include an item for each supported encoding with a link to the features resource (relation: `items`). Requirement 15-2 All links SHALL include the `rel` and `type` properties.

REQUIREMENT 16	
NORMATIVE STATEMENTS	Requirement 16-1 For each feature collection, the `extent` property, if provided, SHALL provide bounding boxes that include all spatial geometries and time intervals that include all temporal geometries in this collection. The temporal extent may use `null` values to indicate an open time interval. Requirement 16-2 If a feature has multiple properties with spatial or temporal information, it is the decision of the server whether only a single spatial or temporal geometry property is used to determine the extent or all relevant geometries.

RECOMMENDATION 11	
NORMATIVE STATEMENT	Recommendation 11-1 While the spatial and temporal extents support multiple bounding boxes (`bbox` array) and time intervals (`interval` array) for advanced use cases, implementations

RECOMMENDATION 11

SHOULD provide only a single bounding box or time interval unless the use of multiple values is important for the use of the dataset and agents using the API are known to be support multiple bounding boxes or time intervals.

PERMISSION 4

NORMATIVE STATEMENTS

Permission 4-1 The Core only specifies requirements for spatial and temporal extents. However, the `extent` object MAY be extended with additional members to represent other extents, for example, thermal or pressure ranges.

Permission 4-2 The Core only supports spatial extents in WGS 84 longitude/latitude and temporal extents in the Gregorian calendar (these are the only enum values in extent.yaml).

Permission 4-3 Extension to the Core MAY add additional reference systems to the `extent` object.

Schema for a feature collection

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

```
              In the Core only a single bounding box is supported. Extensions
may support
              additional areas. If multiple areas are provided, the union of the
bounding
              boxes describes the spatial extent.
            type: array
            minItems: 1
            items:
              description: >-
                Each bounding box is provided as four or six numbers, depending
on
                whether the coordinate reference system includes a vertical axis
                (height or depth):

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

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

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

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

                If a feature has multiple spatial geometry properties, it is the
decision of the
                server whether only a single spatial geometry property is used
to determine
                the extent or all relevant geometries.
              type: array
              minItems: 4
              maxItems: 6
              items:
                type: number
              example:
                - -180
                - -90
                - 180
                - 90
          crs:
            description: >-
              Coordinate reference system of the coordinates in the spatial
extent
              (property `bbox`). The default reference system is WGS 84
longitude/latitude.
              In the Core this is the only supported coordinate reference system.
```

```
              Extensions may support additional coordinate reference systems and
add
              additional enum values.
            type: string
            enum:
              - 'http://www.opengis.net/def/crs/OGC/1.3/CRS84'
            default: 'http://www.opengis.net/def/crs/OGC/1.3/CRS84'
      temporal:
        description: >-
          The temporal extent of the features in the collection.
        type: object
        properties:
          interval:
            description: >-
              One or more time intervals that describe the temporal extent of
the dataset.
              The value `null` is supported and indicates an open time intervall.
              In the Core only a single time interval is supported. Extensions
may support
              multiple intervals. If multiple intervals are provided, the union
of the
              intervals describes the temporal extent.
            type: array
            minItems: 1
            items:
              description: >-
                Begin and end times of the time interval. The timestamps
                are in the coordinate reference system specified in `trs`. By
default
                this is the Gregorian calendar.
              type: array
              minItems: 2
              maxItems: 2
              items:
                type: string
                format: date-time
                nullable: true
              example:
                - '2011-11-11T12:22:11Z'
                - null
          trs:
            description: >-
              Coordinate reference system of the coordinates in the temporal
extent
              (property `interval`). The default reference system is the
Gregorian calendar.
              In the Core this is the only supported temporal reference system.
              Extensions may support additional temporal reference systems and
add
              additional enum values.
            type: string
            enum:
              - 'http://www.opengis.net/def/uom/ISO-8601/0/Gregorian'
            default: 'http://www.opengis.net/def/uom/ISO-8601/0/Gregorian'
  itemType:
    description: indicator about the type of the items in the collection (the
default value is 'feature').
    type: string
    default: feature
  crs:
    description: the list of coordinate reference systems supported by the
service
    type: array
```

```
    items:
      type: string
    default:
      - http://www.opengis.net/def/crs/OGC/1.3/CRS84
```

Listing

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

Example — Feature collections response document: This feature collections example response in JSON is for a dataset with a single collection "buildings". It includes links to the features resource in all formats that are supported by the service (link relation type: "items"). Representations of the resource in other formats are referenced using link relation type "alternate".

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

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

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


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

```
          "title": "CC0-1.0" },
        { "href": "https://creativecommons.org/publicdomain/zero/1.0/rdf",
          "rel": "license", "type": "application/rdf+xml",
          "title": "CC0-1.0" }
      ]
    }
  ]
}
```


7.13.3. Error situations

See HTTP status codes for general guidance.

7.14. Feature collection

7.14.1. Operation

REQUIREMENT 17	
NORMATIVE STATEMENTS	Requirement 17-1 The server SHALL support the HTTP GET operation at the path `/collections/{collectionId}`. Requirement 17-2 The parameter `collectionId` is each `id` property in the feature collections response (JSONPath: `$.collections[*].id`).

7.14.2. Response

REQUIREMENT 18	
NORMATIVE STATEMENTS	Requirement 18-1 A successful execution of the operation SHALL be reported as a response with a HTTP status code `200`. Requirement 18-2 The content of that response SHALL be consistent with the content for this feature collection in the `/collections` response. That is, the values for `id`, `title`, `description` and `extent` SHALL be identical.

7.14.3. Error situations

See HTTP status codes for general guidance.

If the parameter `collectionId` does not exist on the server, the status code of the response will be `404` (see Table 2).

7.15. Features

7.15.1. Operation

REQUIREMENT 19	
NORMATIVE STATEMENTS	Requirement 19-1 For every feature collection identified in the feature collections response (path `/collections`), the server SHALL support the HTTP GET operation at the path `/collections/{collectionId}/items`. Requirement 19-2 The parameter `collectionId` is each `id` property in the feature collections response (JSONPath: `$.collections[*].id`).

7.15.2. Parameter limit

REQUIREMENT 20	

NORMATIVE STATEMENT	Requirement 20-1 The operation SHALL support a parameter `limit` with the following characteristics (using an OpenAPI Specification 3.0 fragment):
	`name: limit`
	`in: query`
	`required: false`
	`schema:`
	`  type: integer`
	`  minimum: 1`
	`  maximum: 10000`
	`  default: 10`
	`style: form`
	`explode: false`

PERMISSION 5	
NORMATIVE STATEMENT	Permission 5-1 The values for `minimum`, `maximum` and `default` in requirement `/req/core/fc-limit-definition` are only examples and MAY be changed.

REQUIREMENT 21	
NORMATIVE STATEMENTS	Requirement 21-1 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. Requirement 21-2 Only items are counted that are on the first level of the collection. Any nested objects contained within the explicitly requested items SHALL not be counted.

PERMISSION 6	
NORMATIVE STATEMENT	Permission 6-1 The server MAY return less features than requested (but not more).

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

7.15.3. Parameter bbox

REQUIREMENT 22	

NORMATIVE STATEMENT	Requirement 22-1 The operation SHALL support a parameter `bbox` with the following characteristics (using an OpenAPI Specification 3.0 fragment):
	`name: bbox`
	`in: query`
	`required: false`
	`schema:`
	`  type: array`
	`  minItems: 4`
	`  maxItems: 6`
	`  items:`
	`  type: number`
	`style: form`
	`explode: false`

REQUIREMENT 23	
NORMATIVE STATEMENTS	Requirement 23-1 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. Requirement 23-2 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. Requirement 23-3 The `bbox` parameter SHALL match all features in the collection that are not associated with a spatial geometry, too. Requirement 23-4 The bounding box is provided as four or six numbers, depending on whether the coordinate reference system includes a vertical axis (height or depth):

REQUIREMENT 23

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

Requirement 23-5 The bounding box SHALL consist of four numbers and the coordinate reference system of the values SHALL be interpreted as WGS 84 longitude/latitude (http://www.opengis.net/def/crs/OGC/1.3/CRS84) unless a different coordinate reference system is specified in a parameter `bbox-crs`.
Requirement 23-6 The coordinate values SHALL be within the extent specified for the coordinate reference system.

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

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

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

Example — The bounding box of the New Zealand Exclusive Economic Zone: The bounding box of the New Zealand Exclusive Economic Zone in WGS 84 (from 160.6°E to 170°W and from 55.95°S to 25.89°S) would be represented in JSON as `[ 160.6, -55.95, -170, -25.89 ]` and in a query as `bbox=160.6,-55.95,-170,-25.89`.

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

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

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

7.15.4. Parameter datetime

REQUIREMENT 24	

NORMATIVE STATEMENT	Requirement 24-1 The operation SHALL support a parameter `datetime` with the following characteristics (using an OpenAPI Specification 3.0 fragment):
	`name: datetime`
	`in: query`
	`required: false`
	`schema:`
	`  type: string`
	`style: form`
	`explode: false`

REQUIREMENT 25	

NORMATIVE STATEMENTS	Requirement 25-1 Only features that have a temporal geometry that intersects the temporal information in the `datetime` parameter SHALL be part of the result set, if the parameter is provided.
	Requirement 25-2 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.
	Requirement 25-3 The `datetime` parameter SHALL match all features in the collection that are not associated with a temporal geometry, too.
	Requirement 25-4 Temporal geometries are either a date-time value or a time interval. The parameter value SHALL conform to the following syntax (using ABNF):
	`interval-closed = date-time "/" date-time`
	`interval-open-start = [".."] "/" date-time`
	`interval-open-end = date-time "/" [".."]`
	`interval = interval-closed / interval-open-start / interval-open-end`
	`datetime = date-time / interval`
	Requirement 25-5 The syntax of `date-time` is specified by RFC 3339, 5.6.
	Requirement 25-6 Open ranges in time intervals at the start or end are supported using a double-dot (`..`) or an empty string for the start/end.

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

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

Example — A date-time: February 12, 2018, 23:20:52 UTC:
`datetime=2018-02-12T23%3A20%3A52Z`

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

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

Example — Intervals: February 12, 2018, 00:00:00 UTC to March 18, 2018, 12:31:12 UTC:
`datetime=2018-02-12T00%3A00%3A00Z%2F2018-03-18T12%3A31%3A12Z`

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

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

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

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

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

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

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

7.15.5. Parameters for filtering on feature properties

RECOMMENDATION 12	

NORMATIVE STATEMENT	Recommendation 12-1 If features in the feature collection include a feature property that has a simple value (for example, a string or integer) that is expected to be useful for applications using the service to filter the features of the collection based on this property, a parameter with the name of the feature property and with the following characteristics (using an OpenAPI Specification 3.0 fragment) SHOULD be supported:
	`in: query`
	`required: false`
	`style: form`
	`explode: false`
	The `schema` property SHOULD be the same as the definition of the feature property in the response schema.

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

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

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

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

7.15.6. Combinations of filter parameters

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

7.15.7. Response

REQUIREMENT 26	
NORMATIVE STATEMENTS	Requirement 26-1 A successful execution of the operation SHALL be reported as a response with a HTTP status code `200`. Requirement 26-2 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.

REQUIREMENT 27	
NORMATIVE STATEMENT	Requirement 27-1 A `200`-response SHALL include the following links: • a link to this response document (relation: `self`), • a link to the response document in every other media type supported by the service (relation: `alternate`).

RECOMMENDATION 13	
NORMATIVE STATEMENT	Recommendation 13-1 A `200`-response SHOULD include a link to the next "page" (relation: `next`), if more features have been selected than returned in the response.

RECOMMENDATION 14	
NORMATIVE STATEMENT	Recommendation 14-1 Dereferencing a `next` link SHOULD return additional features from the set of selected features that have not yet been returned.

RECOMMENDATION 15	
NORMATIVE STATEMENT	Recommendation 15-1 The number of features in a response to a `next` link SHOULD follow the same rules as for the response to the original query and again include a `next` link, if there are more features in the selection that have not yet been returned.

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

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

Another implementation approach is to use an implementation-specific parameter that specifies the index within the result set from which the server begins presenting results in the response,

like the `startIndex` parameter that was used in WFS 2.0 (and which may be added again in additional parts of the OGC API Features series).

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

NOTE 1: Additional conformance classes for safe paging or an index parameter may be added in extensions to this specification.

PERMISSION 7	
NORMATIVE STATEMENT	Permission 7-1 A response to a `next` link MAY include a `prev` link to the resource that included the `next` link.

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

REQUIREMENT 28	
NORMATIVE STATEMENT	Requirement 28-1 All links SHALL include the `rel` and `type` link parameters.

REQUIREMENT 29	
NORMATIVE STATEMENT	Requirement 29-1 If a property `timeStamp` is included in the response, the value SHALL be set to the time stamp when the response was generated.

REQUIREMENT 30	
NORMATIVE STATEMENTS	Requirement 30-1 If a property `numberMatched` is included in the response, the value SHALL be identical to the number of features in the feature collections that match the selection parameters like `bbox`, `datetime` or additional filter parameters. Requirement 30-2 A server MAY omit this information in a response, if the information about the number of matching features is not known or difficult to compute.

REQUIREMENT 31	
NORMATIVE STATEMENTS	Requirement 31-1 If a property `numberReturned` is included in the response, the value SHALL be identical to the number of features in the response. Requirement 31-2 A server MAY omit this information in a response, if the information about the number of features in the response is not known or difficult to compute.

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

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 `offset` to implement `next` links — and the optional `prev` links):

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

Following the `next` link could return:

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

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

```
Link: <link:++http://data.example.org/collections/buildings/items.json?limit=50>;
++[] rel="self"; type="application/geo+json"
Link: <link:++http://data.example.org/collections/buildings/items.html?limit=50>;
++[] rel="alternate"; type="text/html"
Link: <link:++http://data.example.org/collections/buildings/items.json?limit=
50&#x26;offset=50&#x3e;;++[] rel="next"; type="application/geo+json"
```

Following the `next` link could return:

```
Link: <link:++http://data.example.org/collections/buildings/items.json?limit=
50&#x26;offset=50&#x3e;;++[] rel="self"; type="application/geo+json"
Link: <link:++http://data.example.org/collections/buildings/items.html?limit=
50&#x26;offset=50&#x3e;;++[] rel="alternate"; type="text/html"
Link: <link:++http://data.example.org/collections/buildings/items.json?limit=
50&#x26;offset=0&#x3e;;++[] rel="prev"; type="application/geo+json"
Link: <link:++http://data.example.org/collections/buildings/items.json?limit=
50&#x26;offset=100&#x3e;;++[] rel="next"; type="application/geo+json"
```

7.15.8. Error situations

See HTTP status codes for general guidance.

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

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

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

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

7.16. Feature

7.16.1. Operation

REQUIREMENT 32	
NORMATIVE STATEMENTS	Requirement 32-1 For every feature in a feature collection (path `/collections/{collectionId}`), the server SHALL support the HTTP GET operation at the path `/collections/{collectionId}/items/{featureId}`. Requirement 32-2 The parameter `collectionId` is each `id` property in the feature collections response (JSONPath: `$.collections[*].id`). `featureId` is a local identifier of the feature.

PERMISSION 8	
NORMATIVE STATEMENT	Permission 8-1 The Core requirements class only requires that the feature URI is unique. Implementations MAY apply stricter rules and, for example, use unique `id` values per dataset or collection.

7.16.2. Response

REQUIREMENT 33	
NORMATIVE STATEMENT	Requirement 33-1 A successful execution of the operation SHALL be reported as a response with a HTTP status code `200`.

REQUIREMENT 34	
NORMATIVE STATEMENTS	Requirement 34-1 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

REQUIREMENT 34

- a link to the feature collection that contains this feature (relation: `collection`).

Requirement 34-2 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: <link:++http://data.example.org/collections/buildings/items/123.json>;++[]
rel="self"; type="application/geo+json"
Link: <link:++http://data.example.org/collections/buildings/items/123.html>;++[]
rel="alternate"; type="text/html"
Link: <link:++http://data.example.org/collections/buildings.json>;++[] rel=
"collection"; type="application/json"
Link: <link:++http://data.example.org/collections/buildings.html>;++[] rel=
"collection"; type="text/html"
```

7.16.3. Error situations

See HTTP status codes for general guidance.

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

8

REQUIREMENTS CLASSES FOR ENCODINGS

8 REQUIREMENTS CLASSES FOR ENCODINGS

8.1. Overview

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

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

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

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

8.2. Requirements Class “HTML”

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

- The data is not discoverable using the most common mechanism for discovering information, 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.

REQUIREMENTS CLASS 2	
OBLIGATION	requirement
TARGET TYPE	Web API
PREREQUISITES	Conformance Class “Core” HTML5 Schema.org

REQUIREMENT 35	
NORMATIVE STATEMENT	Requirement 35-1 Every `200`-response of an operation of the server SHALL support the media type `text/html`.

REQUIREMENT 36	
NORMATIVE STATEMENT	Requirement 36-1 Every `200`-response of the server with the media type `text/html` SHALL be a HTML 5 document that includes the following information in the HTML body: • all information identified in the schemas of the Response Object in the HTML `<body>`, and • all links in HTML `<a>` elements in the HTML `<body>`.

RECOMMENDATION 16	
NORMATIVE STATEMENT	Recommendation 16-1 A `200`-response with the media type `text/html`, SHOULD include Schema.org annotations.

8.3. Requirements Class “GeoJSON”

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

REQUIREMENTS CLASS 3	
OBLIGATION	requirement

REQUIREMENTS CLASS 3	
TARGET TYPE	Web API
PREREQUISITES	Conformance Class “Core” GeoJSON

REQUIREMENT 37	
NORMATIVE STATEMENT	Requirement 37-1 `200`-responses of the server SHALL support the following media types: • `application/geo+json` for resources that include feature content, and • `application/json` for all other resources.

REQUIREMENT 38	
	Requirement 38-1 Every `200`-response with the media type `application/geo+json` SHALL be • a GeoJSON FeatureCollection Object for features, and • a GeoJSON Feature Object for a single feature.
NORMATIVE STATEMENTS	Requirement 38-2 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`. Requirement 38-3 The schema of all responses with the media type `application/json` SHALL conform with the JSON Schema specified for the resource in the Core requirements class.

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

Example — 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&offset=
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 — 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",
    "rel" : "collection",
    "type" : "application/json",
    "title" : "the collection document"
  } ],
  "id" : "123",
  "geometry" : {
    "type" : "Polygon",
```

```
    "coordinates" : [ ... ]
  },
  "properties" : {
    "function" : "residential",
    "floors" : "2",
    "lastUpdate" : "2015-08-01T12:34:56Z"
  }
}
```

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

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

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

REQUIREMENTS CLASS 4	
OBLIGATION	requirement
TARGET TYPE	Web API
PREREQUISITES	Conformance Class “Core” Geography Markup Language (GML), Simple Features Profile, Level 0

REQUIREMENT 39	
NORMATIVE STATEMENT	Requirement 39-1 `200`-responses of the server SHALL support the following media types: • `application/gml+xml; version=3.2; profile=http://www.opengis.net/def/profile/ogc/2.0/gml-sf0` for resources that include feature content, • `application/xml` for all other resources.

REQUIREMENT 40	
NORMATIVE STATEMENTS	Requirement 40-1 Table 3 specifies the XML document root element that the server SHALL return in a `200`-response for each resource.

REQUIREMENT 40	
	Requirement 40-2 Every representation of a feature SHALL conform to the GML Simple Features Profile, Level 0 and be substitutable for `gml:AbstractFeature`. Requirement 40-3 The schema of all responses with a root element in the `core` namespace SHALL validate against the OGC API Features Core XML Schema.

REQUIREMENT 41	
NORMATIVE STATEMENTS	Requirement 41-1 If a property `timeStamp` is included in the response, its value SHALL be reported using the HTTP header named `Date` (see RFC 2616, 4.5). Requirement 41-2 If a property `numberMatched` is included in the response, its value SHALL be reported using an HTTP header named `OGC-NumberMatched`. Requirement 41-3 If a property `numberReturned` is included in the response, its value SHALL be reported using an HTTP header named `OGC-NumberReturned`. Requirement 41-4 If links are included in the response, each link SHALL be reported using an HTTP header named `Link` (see RFC 8288, Clause 3).

Table 3 — Media types and XML elements for each resource

RESOURCE	PATH	XML ROOT ELEMENT
Landing page	`/`	`core:LandingPage`
Conformance declaration	`/conformance`	`core:ConformsTo`
Feature collections	`/collections`	`core:Collections`
Feature collection	`/collections/{collectionId}`	`core:Collections`, with just one entry for the collection `collectionId`
Features	`/collections/{collectionId}/items`	`sf:FeatureCollection`
Feature	`/collections/{collectionId}/items/{featureId}`	substitutable for `gml:AbstractFeature`

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

- `core`: http://www.opengis.net/ogcapi-features-1/1.0
- `sf`: http://www.opengis.net/ogcapi-features-1/1.0/sf
- `gml`: http://www.opengis.net/gml/3.2
- `atom`: http://www.w3.org/2005/Atom
- `xlink`: http://www.w3.org/1999/xlink

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

See Clause 6.3 for links to example responses in XML.

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

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

REQUIREMENTS CLASS 5	
OBLIGATION	requirement
TARGET TYPE	Web API
PREREQUISITES	Conformance Class "Core" Geography Markup Language (GML), Simple Features Profile, Level 2

REQUIREMENT 42	
NORMATIVE STATEMENT	Requirement 42-1 `200`-responses of the server SHALL support the following media types: • `application/gml+xml; version=3.2; profile=http://www.opengis.net/def/profile/ogc/2.0/gml-sf2` for resources that include feature content, • `application/xml` for all other resources.

REQUIREMENT 43	
NORMATIVE STATEMENT	Requirement 43-1 The requirement `/req/gmlsf0/content` applies, too, with the following changes: • All references to media type `application/gml+xml; version=3.2; profile=http://www.opengis.net/def/profile/ogc/2.0/gml-sf0` are replaced by `application/gml+xml; version=3.2; profile=http://www.opengis.net/def/profile/ogc/2.0/gml-sf2`. • All references to "GML Simple Features Profile, Level 0" are replaced by "GML Simple Features Profile, Level 2".

REQUIREMENT 44	
NORMATIVE STATEMENT	Requirement 44-1 The requirement `/req/gmlsf0/content` applies.

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.

REQUIREMENTS CLASS 6	
OBLIGATION	requirement
TARGET TYPE	Web API
PREREQUISITES	Conformance Class "Core" OpenAPI Specification 3.0.2

REQUIREMENT 45	
NORMATIVE STATEMENT	Requirement 45-1 An OpenAPI definition in JSON using the media type `application/vnd.oai.openapi+json;version=3.0` and a HTML version of the API definition using the media type `text/html` SHALL be available.

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

REQUIREMENT 46	
NORMATIVE STATEMENT	Requirement 46-1 The JSON representation SHALL conform to the OpenAPI Specification, version 3.0.

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

REQUIREMENT 47	
NORMATIVE STATEMENT	Requirement 47-1 The server SHALL implement all capabilities specified in the OpenAPI definition.

9.2. Complete definition

REQUIREMENT 48	
NORMATIVE STATEMENTS	Requirement 48-1 The OpenAPI definition SHALL specify for each operation all HTTP Status Codes and Response Objects that the server uses in responses. Requirement 48-2 This includes the successful execution of an operation as well as all error situations that originate from the server.

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

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

9.3. Exceptions

REQUIREMENT 49	
NORMATIVE STATEMENT	Requirement 49-1 For error situations that originate from the server, the API definition SHALL cover all applicable HTTP Status Codes.

Example — An exception response object definition

```
description: An error occurred.
content:
  application/json:
    schema:
      $ref: http://schemas.opengis.net/ogcapi/features/part1/1.0/openapi/schemas/
exception.yaml
  text/html:
    schema:
      type: string
```

9.4. Security

REQUIREMENT 50	
NORMATIVE STATEMENT	Requirement 50-1 For cases, where the operations of the server are access-controlled, the security scheme(s) SHALL be documented in the OpenAPI definition.

The OpenAPI specification currently supports the following security schemes:

- HTTP authentication,
- an API key (either as a header or as a query parameter),
- OAuth2's common flows (implicit, password, application and access code) as defined in RFC6749, and
- OpenID Connect Discovery.

9.5. Features

RECOMMENDATION 17	
NORMATIVE STATEMENT	Recommendation 17-1 The schema for the Response Objects of the HTTP GET operation for features SHOULD include key feature properties of the features in that feature collection. This is particularly helpful if filter parameters are defined for the collection (see recommendation `/rec/core/fc-filters`).

10

MEDIA TYPES

10 MEDIA TYPES

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

- `application/geo+json` for feature collections and features, and
- `application/json` for all other resources.

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

- `application/gml+xml;version=3.2` for any GML 3.2 feature collections and features,
- `application/gml+xml;version=3.2;profile=http://www.opengis.net/def/profile/ogc/2.0/gml-sf0` for GML 3.2 feature collections and features conforming to the GML Simple Feature Level 0 profile,
- `application/gml+xml;version=3.2;profile=http://www.opengis.net/def/profile/ogc/2.0/gml-sf2` for GML 3.2 feature collections and features conforming to the GML Simple Feature Level 2 profile, and
- `application/xml` for all other resources.

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

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

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

A

ANNEX A (NORMATIVE) ABSTRACT TEST SUITE

A ANNEX A (NORMATIVE) ABSTRACT TEST SUITE

A.1. Introduction

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

A.2. Conformance Class Core

CONFORMANCE CLASS A.1	
SUBJECT	Web API

A.2.1. General Tests

A.2.1.1. HTTP

ABSTRACT TEST A.1	
NORMATIVE STATEMENTS	Requirement A.1-1 /ats/core/http Requirement A.1-2 /req/core/http Requirement A.1-3 1. All compliance tests shall be configured to use the HTTP 1.1 protocol exclusively.

ABSTRACT TEST A.1	
	2. For APIs which support HTTPS, all compliance tests shall be configured to use HTTP over TLS (RFC 2818) with their HTTP 1.1 protocol.

A.2.1.2. CRS 84

ABSTRACT TEST A.2	
NORMATIVE STATEMENTS	Requirement A.2-1 Validate that all spatial geometries provided through the API are in the CRS84 spatial reference system unless otherwise requested by the client. Requirement A.2-2 /req/core/crs84 Requirement A.2-3 1. Do not specify a coordinate reference system in any request. All spatial data should be in the CRS84 reference system. 2. Validate retrieved spatial data using the CRS84 reference system.

A.2.2. Landing Page {root}/

ABSTRACT TEST A.3	
NORMATIVE STATEMENTS	Requirement A.3-1 Validate that a landing page can be retrieved from the expected location. Requirement A.3-2 /req/core/root-op Requirement A.3-3 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 /ats/core/root-success.

ABSTRACT TEST A.4	
NORMATIVE STATEMENTS	Requirement A.4-1 Validate that the landing page complies with the require structure and contents. Requirement A.4-2 /req/core/root-success Requirement A.4-3 Validate the landing page for all supported media types using the resources and tests identified in Schema and Tests for Landing Pages For formats that require manual inspection, perform the following: 1. Validate that the landing page includes a “service-desc” and/or “service-doc” link to an API Definition

ABSTRACT TEST A.4	
	2. Validate that the landing page includes a “conformance” link to the conformance class declaration 3. Validate that the landing page includes a “data” link to the Feature contents.

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

Table A.1 — Schema and Tests for Landing Pages

FORMAT	SCHEMA DOCUMENT	TEST ID
HTML	landingPage.yaml	/ats/html/content
GeoJSON	landingPage.yaml	/ats/geojson/content
GMLSF0	core.xsd, element `core:LandingPage`	/ats/gmlsf0/content
GMLSF2	core.xsd, element `core:LandingPage`	/ats/gmlsf2/content

A.2.3. API Definition Path {root}/api (link)

ABSTRACT TEST A.5	
NORMATIVE STATEMENTS	Requirement A.5-1 Validate that the API Definition document can be retrieved from the expected location. Requirement A.5-2 /req/core/api-definition-op Requirement A.5-3 Validate that the API Definition document can be retrieved from the expected location. Requirement A.5-4 1. Construct a path for each API Definition link on the landing page 2. Issue a HTTP GET request on each path 3. Validate that a document was returned with a status code 200 4. Validate the contents of the returned document using test /ats/core/api-definition-success.

ABSTRACT TEST A.6	
NORMATIVE STATEMENTS	Requirement A.6-1 Validate that the API Definition complies with the required structure and contents. Requirement A.6-2 /req/core/api-definition-success Requirement A.6-3 Validate the API Definition document against an appropriate schema document.

A.2.4. Conformance Path {root}/conformance

ABSTRACT TEST A.7	
NORMATIVE STATEMENTS	Requirement A.7-1 Validate that a Conformance Declaration can be retrieved from the expected location. Requirement A.7-2 /req/core/conformance-op Requirement A.7-3 1. Construct a path for each "conformance" link on the landing page as well as for the {root}/conformance path. 2. Issue an HTTP GET request on each path 3. Validate that a document was returned with a status code 200 4. Validate the contents of the returned document using test /ats/core/conformance-success.

ABSTRACT TEST A.8	
NORMATIVE STATEMENTS	Requirement A.8-1 Validate that the Conformance Declaration response complies with the required structure and contents. Requirement A.8-2 /req/core/conformance-success Requirement A.8-3 1. Validate the response document against OpenAPI 3.0 schema confClasses.yaml 2. Validate that the document includes the conformance class "http://www.opengis.net/spec/ogcapi-features-1/1.0/conf/core" 3. Validate that the document list all OGC API conformance classes that the API implements.

A.2.5. Feature Collections {root}/collections

ABSTRACT TEST A.9	
NORMATIVE STATEMENTS	Requirement A.9-1 Validate that information about the Collections can be retrieved from the expected location.

ABSTRACT TEST A.9	
	Requirement A.9-2 /req/core/fc-md-op Requirement A.9-3 1. Issue an HTTP GET request to the URL {root}/collections 2. Validate that a document was returned with a status code 200 3. Validate the contents of the returned document using test /ats/core/fc-md-success.

ABSTRACT TEST A.10	
NORMATIVE STATEMENTS	Requirement A.10-1 Validate that the Collections content complies with the required structure and contents. Requirement A.10-2 /req/core/fc-md-success, /req/core/crs84 Requirement A.10-3 1. Validate that all response documents comply with /ats/core/fc-md-links 2. Validate that all response documents comply with /ats/core/fc-md-items 3. In case the response includes a "crs" property, validate that the first value is either "http://www.opengis.net/def/crs/OGC/1.3/CRS84" or "http://www.opengis.net/def/crs/OGC/0/CRS84h" 4. Validate the collections content for all supported media types using the resources and tests identified in Schema and Tests for Collections content

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

Table A.2 — Schema and Tests for Collections content

FORMAT	SCHEMA DOCUMENT	TEST ID
HTML	collections.yaml	/ats/html/content
GeoJSON	collections.yaml	/ats/geojson/content
GMLSF0	core.xsd, element `core:Collections`	/ats/gmlsf0/content
GMLSF2	core.xsd, element `core:Collections`	/ats/gmlsf2/content

A.2.6. Feature Collection {root}/collections/{collectionId}

ABSTRACT TEST A.11	
NORMATIVE STATEMENTS	Requirement A.11-1 Validate that the Collection content can be retrieved from the expected location. Requirement A.11-2 /req/core/sfc-md-op Requirement A.11-3 For every Feature Collection described in the Collections content, issue an HTTP GET request to the URL `/collections/{collectionId}` where `{collectionId}` is the `id` property for the collection. . Validate that a Collection was returned with a status code 200 . Validate the contents of the returned document using test /ats/core/sfc-md-success.

ABSTRACT TEST A.12	
NORMATIVE STATEMENTS	Requirement A.12-1 Validate that the Collection content complies with the required structure and contents. Requirement A.12-2 /req/core/sfc-md-success Requirement A.12-3 Verify that the content of the response is consistent with the content for this Feature Collection in the `/collections` response. That is, the values for `id`, `title`, `description` and `extent` are identical.

A.2.7. Features {root}/collections/{collectionId}/items

ABSTRACT TEST A.13	
NORMATIVE STATEMENTS	Requirement A.13-1 Validate that features can be identified and extracted from a Collection using query parameters. Requirement A.13-2 /req/core/fc-op Requirement A.13-3 1. For every feature collection identified in Collections, issue an HTTP GET request to the URL `/collections/{collectionId}/items` where `{collectionId}` is the `id` property for a Collection described in the Collections content. 2. Validate that a document was returned with a status code 200. 3. Validate the contents of the returned document using test /ats/core/fc-response. Repeat these tests using the following parameter tests: **Bounding Box:** • Parameter /ats/core/fc-bbox-definition • Response /ats/core/fc-bbox-response **Limit:** • Parameter /ats/core/fc-limit-definition • Response /ats/core/fc-limit-response

ABSTRACT TEST A.13

DateTime:

- Parameter /ats/core/fc-time-definition
- Response /ats/core/fc-time-response

Error conditions:

- Query Invalid /ats/core/query-param-invalid
- Query Unknown /ats/core/query-param-unknown

Execute requests with combinations of the “bbox” and “datetime” query parameters and verify that only features are returned that match both selection criteria.

ABSTRACT TEST A.14

NORMATIVE STATEMENTS

Requirement A.14-1 Validate that the bounding box query parameters are constructed correctly.

Requirement A.14-2 /req/core/fc-bbox-definition

Requirement A.14-3 Verify that the `bbox` query parameter complies with the following definition (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
```

Use a bounding box with four numbers in all requests:

- Lower left corner, WGS 84 longitude
- Lower left corner, WGS 84 latitude
- Upper right corner, WGS 84 longitude
- Upper right corner, WGS 84 latitude

ABSTRACT TEST A.15

NORMATIVE STATEMENTS

Requirement A.15-1 Validate that the bounding box query parameters are processed corrrectly.

Requirement A.15-2 /req/core/fc-bbox-response

Requirement A.15-3

1. Verify that only features that have a spatial geometry that intersects the bounding box are returned as part of the result set.

ABSTRACT TEST A.15

2. Verify that the `bbox` parameter matched all features in the collection that were not associated with a spatial geometry (this is only applicable for datasets that include features without a spatial geometry).

3. Verify that the coordinate reference system of the geometries is WGS 84 longitude/latitude ("http://www.opengis.net/def/crs/OGC/1.3/CRS84" or "http://www.opengis.net/def/crs/OGC/0/CRS84h") since no parameter `bbox-crs` was specified in the request.

ABSTRACT TEST A.16

NORMATIVE STATEMENTS

Requirement A.16-1 Validate that the bounding box query parameters are constructed corrrectly.

Requirement A.16-2 /req/core/fc-limit-definition

Requirement A.16-3 Verify that the `limit` query parameter complies with the following definition (using an OpenAPI Specification 3.0 fragment):

```
name: limit
in: query
required: false
schema:
  type: integer
style: form
explode: false
```

Note that the API can define values for "minimum", "maximum" and "default".

ABSTRACT TEST A.17

NORMATIVE STATEMENTS

Requirement A.17-1 Validate that the limit query parameters are processed correctly.

Requirement A.17-2 /req/core/fc-limit-response

Requirement A.17-3

1. Count the Features which are on the first level of the collection. Any nested objects contained within the explicitly requested items are not be counted.

2. Verify that this count is not greater than the value specified by the `limit` parameter.

3. If the API definition specifies a maximum value for `limit` parameter, verify that the count does not exceed this maximum value.

ABSTRACT TEST A.18

NORMATIVE STATEMENTS

Requirement A.18-1 Validate that the `dateTime` query parameters are constructed correctly.

Requirement A.18-2 /req/core/fc-time-definition

Requirement A.18-3 Verify that the `datetime` query parameter complies with the following definition (using an OpenAPI Specification 3.0 fragment):

```
name: datetime
in: query
required: false
schema:
  type: string
```

ABSTRACT TEST A.18

```
                style: form
                explode: false
```

ABSTRACT TEST A.19

NORMATIVE STATEMENTS	Requirement A.19-1 Validate that the dataTime query parameters are processed correctly. Requirement A.19-2 /req/core/fc-time-response Requirement A.19-3 1. Verify that only features that have a temporal geometry that intersects the temporal information in the `datetime` parameter were included in the result set 2. Verify that all features in the collection that are not associated with a temporal geometry are included in the result set 3. Validate that the dateime parameter complies with the syntax described in /req/core/fc-time-response.

ABSTRACT TEST A.20

NORMATIVE STATEMENTS	Requirement A.20-1 Validate that the API correctly deals with invalid query parameters. Requirement A.20-2 /req/core/query-param-invalid Requirement A.20-3 1. Enter an HTTP request with an invalid query parameter. 2. Verify that the API returns the status code `400`.

ABSTRACT TEST A.21

NORMATIVE STATEMENTS	Requirement A.21-1 Validate that the API correctly deals with unknown query parameters. Requirement A.21-2 /req/core/query-param-unknown Requirement A.21-3 1. Enter an HTTP request with an query parameter that is not specified in the API definition. 2. Verify that the API returns the status code `400`.

ABSTRACT TEST A.22

NORMATIVE STATEMENTS	Requirement A.22-1 Validate that the Feature Collections complies with the require structure and contents. Requirement A.22-2 /req/core/fc-response Requirement A.22-3

ABSTRACT TEST A.22	
	1. Validate that the `type` property is present and has a value of `FeatureCollection` 2. Validate the `features` property is present and that it is populated with an array of `feature` items. 3. Validate that only Features which match the selection criteria are included in the Feature Collection. 4. If the `links` property is present, validate that all entries comply with /ats/core/fc-links 5. If the `timeStamp` property is present, validate that it complies with /ats/core/fc-timeStamp 6. If the `numberMatched` property is present, validate that it complies with /ats/core/fc-numberMatched 7. If the `numberReturned` property is present, validate that it complies with /ats/core/fc-numberReturned 8. Validate the collections content for all supported media types using the resources and tests identified in Schema and Tests for Feature Collections

The collections metadata may be retrieved in a number of different formats. The following table identifies the applicable schema document for each format and the test to be used to validate the against that schema. All supported formats should be exercised.

Table A.3 — Schema and Tests for Feature Collections

FORMAT	SCHEMA DOCUMENT	TEST ID
HTML	featureCollectionGeoJSON.yaml	/ats/html/content
GeoJSON	featureCollectionGeoJSON.yaml	/ats/geojson/content
GMLSF0	core-sf.xsd, element `sf:FeatureCollection`	/ats/gmlsf0/content
GMLSF2	core-sf.xsd, element `sf:FeatureCollection`	/ats/gmlsf2/content

Supporting Tests:

ABSTRACT TEST A.23	
NORMATIVE STATEMENTS	Requirement A.23-1 Validate that the required links are included in the Collections document. Requirement A.23-2 /req/core/fc-links, /req/core/fc-rel-type Requirement A.23-3 Verify that the response document includes:

ABSTRACT TEST A.23	
	1. a link to this response document (relation: `self`), 2. a link to the response document in every other media type supported by the server (relation: `alternate`). Verify that all links include the `rel` and `type` link parameters.

ABSTRACT TEST A.24	
NORMATIVE STATEMENTS	Requirement A.24-1 Validate the timeStamp parameter returned with a Features response Requirement A.24-2 /req/core/fc-timeStamp, /req/core/fc-rel-type Requirement A.24-3 Validate that the `timeStamp` value is set to the time when the response was generated.

ABSTRACT TEST A.25	
NORMATIVE STATEMENTS	Requirement A.25-1 Validate the numberMatched parameter returned with a Features response Requirement A.25-2 /req/core/fc-numberMatched Requirement A.25-3 Validate that the value of the `numberMatched` parameter is identical to the number of features in the feature collections that match the selection parameters like `bbox`, `datetime` or additional filter parameters.

ABSTRACT TEST A.26	
NORMATIVE STATEMENTS	Requirement A.26-1 Validate the numberReturned parameter returned with a Features response Requirement A.26-2 /req/core/fc-numberReturned Requirement A.26-3 Validate that the `numberReturned` value is identical to the number of features in the response.

A.2.8. Feature

ABSTRACT TEST A.27	
NORMATIVE STATEMENTS	Requirement A.27-1 Validate that a feature can be retrieved from the expected location. Requirement A.27-2 /req/core/f-op Requirement A.27-3

ABSTRACT TEST A.27	
	1. For a sufficiently large subset of all features in a feature collection (path `/collections/{collectionId}`), issue an HTTP GET request to the URL `/collections/{collectionId}/items/{featureId}` where `{collectionId}` is the `id` property for the collection and `{featureId}` is the `id` property of the feature. 2. Validate that a feature was returned with a status code 200 3. Validate the contents of the returned feature using test /ats/core/f-success.

ABSTRACT TEST A.28	
NORMATIVE STATEMENTS	Requirement A.28-1 Validate that the Feature complies with the required structure and contents. Requirement A.28-2 /req/core/f-success Requirement A.28-3 1. Validate that the Feature includes all required link properties using /ats/core/f-links 2. Validate the Feature for all supported media types using the resources and tests identified in Schema and Tests for Features

The Features may be retrieved in a number of different formats. The following table identifies the applicable schema document for each format and the test to be used to validate the against that schema. All supported formats should be exercised.

Table A.4 — Schema and Tests for Features

FORMAT	SCHEMA DOCUMENT	TEST ID
HTML	featureGeoJSON.yaml	/ats/html/content
GeoJSON	featureGeoJSON.yaml	/ats/geojson/content
GMLSF0	gml.xsd, element substituable for `gml:AbstractFeature`	/ats/gmlsf0/content
GMLSF2	gml.xsd, element substituable for `gml:AbstractFeature`	/ats/gmlsf2/content

Note that in the case of GMLSF0/GMLSF2 it is not sufficient to validate against gml.xsd as the feature will be defined in a GML application schema. Determine the XML Schema Document for the namespace of the feature to validate the XML document.

Supporting Tests:

ABSTRACT TEST A.29	
NORMATIVE STATEMENTS	Requirement A.29-1 Validate that the required links are included in a Feature. Requirement A.29-2 /req/core/f-links Requirement A.29-3 1. Verify that the returned Feature includes: 2. a link to this response document (relation: `self`), 3. a link to the response document in every other media type supported by the server (relation: `alternate`). 4. a link to the feature collection that contains this feature (relation: `collection`). Verify that all links include the `rel` and `type` link parameters.

A.3. Conformance Class GeoJSON

CONFORMANCE CLASS A.2	
SUBJECT	Web API

A.3.1. GeoJSON Definition

ABSTRACT TEST A.30	
NORMATIVE STATEMENTS	Requirement A.30-1 Verify support for JSON and GeoJSON Requirement A.30-2 /req/geojson/definition Requirement A.30-3
DESCRIPTION	1. A resource is requested with response media type of `application/geo+json` 2. All 200-responses SHALL support the following media types: • `application/geo+json` for resources that include feature content, and • `application/json` for all other resources.

A.3.2. GeoJSON Content

ABSTRACT TEST A.31	
NORMATIVE STATEMENTS	Requirement A.31-1 Verify the content of a GeoJSON document given an input document and schema. Requirement A.31-2 /req/geojson/content Requirement A.31-3 1. Validate that the document is a GeoJSON document. 2. Validate the document against the schema using an JSON Schema validator.

A.4. Conformance Class GML Simple Features Level 0

CONFORMANCE CLASS A.3	
SUBJECT	Web API

A.4.1. GML Simple Features 0 Definition

ABSTRACT TEST A.32	
NORMATIVE STATEMENTS	Requirement A.32-1 Verify support for GML Simple Features level 0 Requirement A.32-2 /req/gmlsf0/definition Requirement A.32-3 Verify that every `200`-response of an operation of the API where XML was requested is of media type `application/gml+xml;profile=http://www.opengis.net/def/profile/ogc/2.0/gml-sf0` (resources: Features and Feature) or `application/xml` (all other resources)

A.4.2. GML Simple Features 0 Content

ABSTRACT TEST A.33	
NORMATIVE STATEMENTS	Requirement A.33-1 Verify the content of an GML Simple Features 0 document given an input document and schema. Requirement A.33-2 /req/gmlsf0/content Requirement A.33-3 1. For the resources “Features” and “Feature”, validate that the document is a GML Simple Features level 0 document. 2. Verify that the document has the expected root element.

ABSTRACT TEST A.33	
	3. Validate the document against the schema using an XML schema validator.

A.5. Conformance Class GML Simple Features Level 2

CONFORMANCE CLASS A.4	
SUBJECT	Web API

A.5.1. GML Simple Features 2 Definition

ABSTRACT TEST A.34	
NORMATIVE STATEMENTS	Requirement A.34-1 Verify support for GML Simple Features level 2 Requirement A.34-2 /req/gmlsf2/definition Requirement A.34-3 Verify that every `200`-response of an operation of the API where XML was requested is of media type `application/gml+xml;profile=http://www.opengis.net/def/profile/ogc/2.0/gml-sf2` (resources: Features and Feature) or `application/xml` (all other resources)

A.5.2. GML Simple Features 2 Content

ABSTRACT TEST A.35	
NORMATIVE STATEMENTS	Requirement A.35-1 Verify the content of an GML Simple Features level 2 document given an input document and schema. Requirement A.35-2 /req/gmlsf2/content Requirement A.35-3 1. For the resources “Features” and “Feature”, validate that the document is a GML Simple Features level 2 document. 2. Verify that the document has the expected root element. 3. Validate the document against the schema using an XML schema validator.

A.6. Conformance Class HTML

CONFORMANCE CLASS A.5	
SUBJECT	Web API

A.6.1. HTML Definition

ABSTRACT TEST A.36	
NORMATIVE STATEMENTS	Requirement A.36-1 Verify support for HTML Requirement A.36-2 /req/html/definition Requirement A.36-3 Verify that every `200`-response of every operation of the API where HTML was requested is of media type `text/html`

A.6.2. HTML Content

ABSTRACT TEST A.37	
NORMATIVE STATEMENTS	Requirement A.37-1 Verify the content of an HTML document given an input document and schema. Requirement A.37-2 /req/html/content Requirement A.37-3 1. Validate that the document is an HTML 5 document 2. Manually inspect the document against the schema.

A.7. Conformance Class OpenAPI 3.0

CONFORMANCE CLASS A.6	
SUBJECT	Web API

ABSTRACT TEST A.38

NORMATIVE STATEMENTS	Requirement A.38-1 Verify the completeness of an OpenAPI document. Requirement A.38-2 /req/oas30/completeness Requirement A.38-3 Verify that for each operation, the OpenAPI document describes all HTTP Status Codes and Response Objects that the API uses in responses.

ABSTRACT TEST A.39

NORMATIVE STATEMENTS	Requirement A.39-1 Verify that the OpenAPI document fully describes potential exception codes. Requirement A.39-2 /req/oas30/exceptions-codes Requirement A.39-3 Verify that for each operation, the OpenAPI document describes all HTTP Status Codes that may be generated.

ABSTRACT TEST A.40

NORMATIVE STATEMENTS	Requirement A.40-1 Verify that JSON and HTML versions of the OpenAPI document are available. Requirement A.40-2 /req/oas30/oas-definition-1 Requirement A.40-3 1. Verify that an OpenAPI definition in JSON is available using the media type `application/vnd.oai.openapi+json;version=3.0` and link relation `service-desc` 2. Verify that an HTML version of the API definition is available using the media type `text/html` and link relation `service-doc`.

ABSTRACT TEST A.41

NORMATIVE STATEMENTS	Requirement A.41-1 Verify that the OpenAPI document is valid JSON. Requirement A.41-2 /req/oas30/oas-definition-2 Requirement A.41-3 Verify that the JSON representation conforms to the Open API Specification, version 3.0.

ABSTRACT TEST A.42

NORMATIVE STATEMENTS	Requirement A.42-1 Verify that all capabilities specified in the OpenAPI definition are implemented by the API. Requirement A.42-2 /req/oas30/oas-impl Requirement A.42-3 1. Construct a path from each URL template including all server URL options and all enumerated path parameters.

ABSTRACT TEST A.42

2. For each path defined in the OpenAPI document, validate that the path performs in accordance with the API definition and the API-Features standard.

ABSTRACT TEST A.43

NORMATIVE STATEMENTS

Requirement A.43-1 Verify that any authentication protocols implemented by the API are documented in the OpenAPI document.

Requirement A.43-2 /req/oas30/security

Requirement A.43-3

1. Identify all authentication protocols supported by the API.
2. Validate that each authentication protocol is described inthe OpenAPI document by a Security Schema Object and its' use specified by a Security Requirement Object.

B

ANNEX B (INFORMATIVE) BIBLIOGRAPHY

B ANNEX B (INFORMATIVE) BIBLIOGRAPHY

[1] Panagiotis (Peter) A. Vretanos: OGC 09-025r2, *OGC® Web Feature Service 2.0 Interface Standard – With Corrigendum*. Open Geospatial Consortium (2014). http://www.opengis.net/doc/IS/wfs/2.0.2.

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

[3] ISO: ISO 19101, *Geographic information – Reference model*. International Organization for Standardization, Geneva https://www.iso.org/standard/26002.html.

[4] ISO: ISO 19142, *Geographic information – Web Feature Service*. International Organization for Standardization, Geneva https://www.iso.org/standard/42136.html.

[5] W3C: W3C dwbp, *Data on the Web Best Practices*. World Wide Web Consortium http://www.w3.org/TR/dwbp/.

[6] W3C: W3C sdw-bp, *Spatial Data on the Web Best Practices*. World Wide Web Consortium http://www.w3.org/TR/sdw-bp/.

[7] W3C: W3C vocab-dcat, *Data Catalog Vocabulary (DCAT)*. World Wide Web Consortium http://www.w3.org/TR/vocab-dcat/.

[8] Ben-Kiki, O., Evans, C., Ingy döt Net: **YAML Ain't Markup Language,** https://yaml.org/

[9] IANA: **Link Relation Types,** https://www.iana.org/assignments/link-relations/link-relations.xml

C

ANNEX C (INFORMATIVE) REVISION HISTORY

ANNEX C (INFORMATIVE) REVISION HISTORY

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
2018-04-02	3.0.0-SNAPSHOT	C. Portele	7, B	Clarify 'item' links #81, clean up OpenAPI example in Annex B

DATE	RELEASE	EDITOR	PRIMARY CLAUSES MODIFIED	DESCRIPTION
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
2019-02-14	3.0.0-SNAPSHOT	C. Portele, C. Holmes	all	Bugfixes #149 and #176, change rel=item to rel=items #175, use {collectionId}, {featureId} and `id` consistently #171
2019-05-02	3.0.0-SNAPSHOT	C. Portele	all	Temporal data support #155, extents #168, result set consistency during paging #192
2019-05-20	**1.0.0-SNAPSHOT**	C. Portele	all	Change document title to “OGC API — Features” #189, minor editorial issues #204, introduce yaml #201, HEAD/OPTIONS #115, /collections path structure #90, resource names #199, /items #164, bbox/time parameter behavior for features without spatial/temporal data #122, change language in overview #124, update XMI #209
2019-06-11	1.0.0-SNAPSHOT	C. Portele	5.6, 7.2, 7.11	Add clarification about default parameter values #215, add title/description to landing page #227, correct informative wording about coordinate reference systems
2019-06-13	1.0.0-SNAPSHOT	C. Heazel, C. Portele, P. Vretanos	0, 7, 8, 11 (new), A, C	Listing of all applicable HTTP Status Codes #45, Deviations between XML and JSON encoding of various structures #133, Add section “Security Considerations” #137, Issues with the UML model and resource descriptions #217
2019-06-22	1.0.0-SNAPSHOT	C. Portele	all	Editorial cleanup, Add anchors #225
2019-06-25	1.0.0-SNAPSHOT	C. Portele	all	Move examples in Annex B/C outside of the document #239, Bulk download #230, CRS 84 Requirement #233, Endpoint /api missing from OpenAPI specification #236
2019-06-26	**1.0.0-draft.1**	C. Portele, C. Heazel	Annex A, all	Update Abstract Test Suite #112, update for release of 17-069r1

DATE	RELEASE	EDITOR	PRIMARY CLAUSES MODIFIED	DESCRIPTION
2019-06-27	1.0.0-SNAPSHOT	C. Portele	0, 7.11	Add draft identifier for WGS 84 lon/lat/h, change to 17-069r2, update front material
2019-07-01	1.0.0-SNAPSHOT	C. Portele	A	Update Abstract Test Suite #112
2019-07-08	1.0.0-SNAPSHOT	C. Portele	7.4, all	Use conformance class URIs in the conformance declaration #244
2019-07-09	1.0.0-SNAPSHOT	C. Portele	all	Clean up document
2019-07-11	**1.0.0-draft.2**	C. Portele, P. Vretanos	8.4, 8.5, A	Update XML to conform to GML Simple Features #150, update front material for submission / version “1.0.0-draft.2”, update examples
2019-09-16	1.0.0-SNAPSHOT	C. Portele	7.13.2, 7.15.2, 7.15.3, 7.15.4	Allow changes to minimum limit value #251, allow unknown start/end in datetime #252, clarification of wording for 3D bounding boxes #259 / #260, prepare release
2019-10-07	**1.0.0**		all	Edits for publication