

**OGC® DOCUMENT: 17-069R3**

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

**OGC®**  
Making location count.

# 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

**Notice:** This document is an OGC Member approved international standard. This document is available on a royalty free, non-discriminatory basis. Recipients of this document are invited to submit, with their comments, notification of any relevant patent rights of which they are aware and to provide supporting documentation.

### **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**

Copyright © 2024 Open Geospatial Consortium

To obtain additional rights of use, visit <https://www.ogc.org/legal>

### **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

I. ABSTRACT .....	viii
II. KEYWORDS .....	x
III. PREFACE .....	xi
IV. SECURITY CONSIDERATIONS .....	xii
IV.A. Multiple Access Routes .....	xii
IV.B. Multiple Servers .....	xiii
IV.C. Path Manipulation on GET .....	xiii
IV.D. Path Manipulation on PUT and POST .....	xiii
V. SUBMITTING ORGANIZATIONS .....	xv
VI. SUBMITTERS .....	xv
1. SCOPE .....	2
2. CONFORMANCE .....	4
3. NORMATIVE REFERENCES .....	7
4. TERMS AND DEFINITIONS .....	9
5. CONVENTIONS .....	12
5.1. Identifiers .....	12
5.2. Link relations .....	12
5.3. Use of HTTPS .....	13
5.4. HTTP URIs .....	13
5.5. API definition .....	14
6. OVERVIEW .....	17
6.1. Design considerations .....	17
6.2. Encodings .....	18
6.3. Examples .....	19
7. REQUIREMENTS CLASS “CORE” .....	21
7.1. Overview .....	21
7.2. API landing page .....	23
7.3. API definition .....	24

7.4. Declaration of conformance classes .....	26
7.5. HTTP 1.1 .....	27
7.6. Unknown or invalid query parameters .....	29
7.7. Web caching .....	30
7.8. Support for cross-origin requests .....	30
7.9. Encodings .....	30
7.10. String internationalization .....	31
7.11. Coordinate reference systems .....	32
7.12. Link headers .....	32
7.13. Feature collections .....	33
7.14. Feature collection .....	39
7.15. Features .....	39
7.16. Feature .....	47
 8. REQUIREMENTS CLASSES FOR ENCODINGS .....	 50
8.1. Overview .....	50
8.2. Requirements Class "HTML" .....	50
8.3. Requirements Class "GeoJSON" .....	51
8.4. Requirements Class "Geography Markup Language (GML), Simple Features Profile, Level 0" .....	53
8.5. Requirements Class "Geography Markup Language (GML), Simple Features Profile, Level 2" .....	55
 9. REQUIREMENTS CLASS "OPENAPI 3.0" .....	 58
9.1. Basic requirements .....	58
9.2. Complete definition .....	59
9.3. Exceptions .....	59
9.4. Security .....	59
9.5. Features .....	60
 10. MEDIA TYPES .....	 62
 ANNEX A (NORMATIVE) ABSTRACT TEST SUITE .....	 64
A.1. Introduction .....	64
A.2. Conformance Class Core .....	64
A.3. Conformance Class GeoJSON .....	71
A.4. Conformance Class GML Simple Features Level 0 .....	72
A.5. Conformance Class GML Simple Features Level 2 .....	73
A.6. Conformance Class HTML .....	74
A.7. Conformance Class OpenAPI 3.0 .....	74
 ANNEX B (INFORMATIVE) BIBLIOGRAPHY .....	 77
 ANNEX C (INFORMATIVE) REVISION HISTORY .....	 79

# LIST OF TABLES

Table 1 – Overview of resources, applicable HTTP methods and links to the document sections .....	ix
Table 2 – Typical HTTP status codes .....	28
Table 3 – Media types and XML elements for each resource .....	54
Table A.1 – Schema and Tests for Landing Pages .....	65
Table A.2 – Schema and Tests for Collections content .....	67
Table A.3 – Schema and Tests for Feature Collections .....	69
Table A.4 – Schema and Tests for Features .....	71

# LIST OF RECOMMENDATIONS

REQUIREMENTS CLASS 1 .....	21
REQUIREMENTS CLASS 2 .....	51
REQUIREMENTS CLASS 3 .....	51
REQUIREMENTS CLASS 4 .....	54
REQUIREMENTS CLASS 5 .....	55
REQUIREMENTS CLASS 6 .....	58
REQUIREMENT 1 .....	23
REQUIREMENT 2 .....	23
REQUIREMENT 3 .....	24
REQUIREMENT 4 .....	25
REQUIREMENT 5 .....	26
REQUIREMENT 6 .....	26
REQUIREMENT 7 .....	27
REQUIREMENT 8 .....	29
REQUIREMENT 9 .....	29
REQUIREMENT 10 .....	32
REQUIREMENT 11 .....	33
REQUIREMENT 12 .....	33
REQUIREMENT 13 .....	33
REQUIREMENT 14 .....	34
REQUIREMENT 15 .....	34

REQUIREMENT 16 .....	34
REQUIREMENT 17 .....	39
REQUIREMENT 18 .....	39
REQUIREMENT 19 .....	40
REQUIREMENT 20 .....	40
REQUIREMENT 21 .....	40
REQUIREMENT 22 .....	40
REQUIREMENT 23 .....	41
REQUIREMENT 24 .....	41
REQUIREMENT 25 .....	41
REQUIREMENT 26 .....	43
REQUIREMENT 27 .....	44
REQUIREMENT 28 .....	45
REQUIREMENT 29 .....	45
REQUIREMENT 30 .....	45
REQUIREMENT 31 .....	46
REQUIREMENT 32 .....	47
REQUIREMENT 33 .....	47
REQUIREMENT 34 .....	48
REQUIREMENT 35 .....	51
REQUIREMENT 36 .....	51
REQUIREMENT 37 .....	52
REQUIREMENT 38 .....	52
REQUIREMENT 39 .....	54
REQUIREMENT 40 .....	54
REQUIREMENT 41 .....	54
REQUIREMENT 42 .....	56
REQUIREMENT 43 .....	56
REQUIREMENT 44 .....	56
REQUIREMENT 45 .....	58
REQUIREMENT 46 .....	58
REQUIREMENT 47 .....	58
REQUIREMENT 48 .....	59

REQUIREMENT 49 .....	59
REQUIREMENT 50 .....	60
RECOMMENDATION 1 .....	25
RECOMMENDATION 2 .....	27
RECOMMENDATION 3 .....	30
RECOMMENDATION 4 .....	30
RECOMMENDATION 5 .....	30
RECOMMENDATION 6 .....	31
RECOMMENDATION 7 .....	31
RECOMMENDATION 8 .....	32
RECOMMENDATION 9 .....	33
RECOMMENDATION 10 .....	34
RECOMMENDATION 11 .....	34
RECOMMENDATION 12 .....	43
RECOMMENDATION 13 .....	44
RECOMMENDATION 14 .....	44
RECOMMENDATION 15 .....	45
RECOMMENDATION 16 .....	51
RECOMMENDATION 17 .....	60
PERMISSION 1 .....	25
PERMISSION 2 .....	28
PERMISSION 3 .....	34
PERMISSION 4 .....	35
PERMISSION 5 .....	40
PERMISSION 6 .....	40
PERMISSION 7 .....	45
PERMISSION 8 .....	47
CONFORMANCE CLASS A.1 .....	64
CONFORMANCE CLASS A.2 .....	71
CONFORMANCE CLASS A.3 .....	72
CONFORMANCE CLASS A.4 .....	73
CONFORMANCE CLASS A.5 .....	74
CONFORMANCE CLASS A.6 .....	74



# 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.

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 through 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 through an OGC API include hyperlinks to other resources. API clients follow these hyperlinks to access new resources or alternate representations of a

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

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



## 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)



## SUBMITTERS

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

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

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

---

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

---

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).

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 <https://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>.

The background features a dark blue field with several thin, light yellow lines intersecting at various points. Three of these intersection points are marked with small yellow dots. One dot is located in the upper right quadrant, another is further to the right and slightly lower, and the third is in the lower left quadrant. The lines create a network of triangles and other geometric shapes across the page.

4

# TERMS AND DEFINITIONS

---

## 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

---

## 5.1. 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.

**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

---

## 6.1. 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 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”

---

## 7.1. 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

### 7.2.2. Response

#### REQUIREMENT 2

##### NORMATIVE STATEMENTS

Requirement 2-1

Requirement 2-2

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

```

### 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

## PERMISSION 1

### NORMATIVE STATEMENT

Permission 1-1

Note that multiple API definition formats can be supported.

## 7.3.2. Response

## REQUIREMENT 4

### NORMATIVE STATEMENT

Requirement 4-1

## RECOMMENDATION 1

### NORMATIVE STATEMENT

Recommendation 1-1

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

### 7.4.2. Response

#### REQUIREMENT 6

NORMATIVE STATEMENTS

Requirement 6-1

Requirement 6-2

Schema for the list of conformance classes

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

**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  
Requirement 7-2

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

### RECOMMENDATION 2

#### NORMATIVE STATEMENT

Recommendation 2-1

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

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

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
```

#### 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
}
```

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

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

## 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

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

## RECOMMENDATION 6

### NORMATIVE STATEMENT

### Recommendation 6-1

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 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

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

## 7.13. Feature collections

### 7.13.1. Operation

#### REQUIREMENT 11

##### NORMATIVE STATEMENT

Requirement 11-1

### 7.13.2. Response

#### REQUIREMENT 12

##### NORMATIVE STATEMENTS

Requirement 12-1

Requirement 12-2

#### Schema for the collections resource

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

#### REQUIREMENT 13

##### NORMATIVE STATEMENTS

Requirement 13-1

Requirement 13-2

#### RECOMMENDATION 9

##### NORMATIVE STATEMENTS

Recommendation 9-1

Recommendation 9-2



## RECOMMENDATION 9

Recommendation 9-3  
Recommendation 9-4

## RECOMMENDATION 10

### NORMATIVE STATEMENTS

Recommendation 10-1  
Recommendation 10-2  
Recommendation 10-3

## REQUIREMENT 14

### NORMATIVE STATEMENT

Requirement 14-1

## PERMISSION 3

### NORMATIVE STATEMENT

Permission 3-1

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  
Requirement 15-2

## REQUIREMENT 16

### NORMATIVE STATEMENTS

Requirement 16-1  
Requirement 16-2

## RECOMMENDATION 11

### NORMATIVE STATEMENT

Recommendation 11-1

## PERMISSION 4

### NORMATIVE STATEMENTS

Permission 4-1

Permission 4-2

Permission 4-3

### 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:** >-

the dataset. One or more time intervals that describe the temporal extent of  
 intervall. The value `null` is supported and indicates an open time  
 may support In the Core only a single time interval is supported. Extensions  
 union of the multiple intervals. If multiple intervals are provided, 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

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

**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
	Requirement 17-2

### 7.14.2. Response

REQUIREMENT 18	
NORMATIVE STATEMENTS	Requirement 18-1
	Requirement 18-2

### 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  
Requirement 19-2

## 7.15.2. Parameter limit

## REQUIREMENT 20

### NORMATIVE STATEMENT

Requirement 20-1

## PERMISSION 5

### NORMATIVE STATEMENT

Permission 5-1

## REQUIREMENT 21

### NORMATIVE STATEMENTS

Requirement 21-1  
Requirement 21-2

## PERMISSION 6

### NORMATIVE STATEMENT

Permission 6-1

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

## REQUIREMENT 23

### NORMATIVE STATEMENTS

Requirement 23-1  
Requirement 23-2  
Requirement 23-3  
Requirement 23-4  
Requirement 23-5  
Requirement 23-6

“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).

**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

## REQUIREMENT 25

### NORMATIVE STATEMENTS

Requirement 25-1  
Requirement 25-2



## REQUIREMENT 25

Requirement 25-3

Requirement 25-4

Requirement 25-5

Requirement 25-6

“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.

**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.

**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

#### 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'
```

#### 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

## REQUIREMENT 26

### Requirement 26-2

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

## RECOMMENDATION 13

### NORMATIVE STATEMENT

### Recommendation 13-1

## RECOMMENDATION 14

### NORMATIVE STATEMENT

### Recommendation 14-1

## RECOMMENDATION 15

### NORMATIVE STATEMENT

#### Recommendation 15-1

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

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

## REQUIREMENT 29

### NORMATIVE STATEMENT

#### Requirement 29-1

## REQUIREMENT 30

### NORMATIVE STATEMENTS

#### Requirement 30-1

## REQUIREMENT 30

Requirement 30-2

## REQUIREMENT 31

NORMATIVE STATEMENTS

Requirement 31-1

Requirement 31-2

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

**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
	Requirement 32-2

PERMISSION 8	
NORMATIVE STATEMENT	Permission 8-1

### 7.16.2. Response

REQUIREMENT 33	
NORMATIVE STATEMENT	Requirement 33-1

## REQUIREMENT 34

### NORMATIVE STATEMENTS

Requirement 34-1

Requirement 34-2

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

**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

---



## 8.1. 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

REQUIREMENT 36	
NORMATIVE STATEMENT	Requirement 36-1

RECOMMENDATION 16	
NORMATIVE STATEMENT	Recommendation 16-1

### 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
TARGET TYPE	Web API
PREREQUISITES	Conformance Class “Core” GeoJSON

## REQUIREMENT 37

### NORMATIVE STATEMENT

Requirement 37-1

## REQUIREMENT 38

### NORMATIVE STATEMENTS

Requirement 38-1

Requirement 38-2

Requirement 38-3

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.

**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"
    }
  }
]
}

```

**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

REQUIREMENT 40	
NORMATIVE STATEMENTS	Requirement 40-1
	Requirement 40-2
	Requirement 40-3

REQUIREMENT 41	
NORMATIVE STATEMENTS	Requirement 41-1
	Requirement 41-2
	Requirement 41-3
	Requirement 41-4

**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

RESOURCE	PATH	XML ROOT ELEMENT
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

## REQUIREMENT 43

NORMATIVE STATEMENT

Requirement 43-1

## REQUIREMENT 44

NORMATIVE STATEMENT

Requirement 44-1



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
---------------------	------------------

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
---------------------	------------------

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
---------------------	------------------

## 9.2. Complete definition

---

### REQUIREMENT 48

#### NORMATIVE STATEMENTS

Requirement 48-1

Requirement 48-2

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

#### 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

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



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.



# ANNEX A (NORMATIVE) ABSTRACT TEST SUITE

---



# 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 Programming Interface (Web API). Therefore, an API may expose resources in addition to those defined by the standard. A test engine must be able to traverse the API, identify and validate test points, and ignore resource paths which are not to be tested.

#### A.2. Conformance Class Core

---

CONFORMANCE CLASS 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
	Requirement A.1-2
	Requirement A.1-3

### A.2.1.2. CRS 84

#### ABSTRACT TEST A.2

NORMATIVE STATEMENTS	Requirement A.2-1
	Requirement A.2-2
	Requirement A.2-3

### A.2.2. Landing Page {root}/

#### ABSTRACT TEST A.3

NORMATIVE STATEMENTS	Requirement A.3-1
	Requirement A.3-2
	Requirement A.3-3

#### ABSTRACT TEST A.4

NORMATIVE STATEMENTS	Requirement A.4-1
	Requirement A.4-2
	Requirement A.4-3

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	<a href="#">landingPage.yaml</a>	/ats/html/content
GeoJSON	<a href="#">landingPage.yaml</a>	/ats/geojson/content
GMLSF0	<a href="#">core.xsd</a> , element core:LandingPage	/ats/gmlsf0/content
GMLSF2	<a href="#">core.xsd</a> , 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  
Requirement A.5-2  
Requirement A.5-3  
Requirement A.5-4

#### ABSTRACT TEST A.6

##### NORMATIVE STATEMENTS

Requirement A.6-1  
Requirement A.6-2  
Requirement A.6-3

### A.2.4. Conformance Path {root}/conformance

#### ABSTRACT TEST A.7

##### NORMATIVE STATEMENTS

Requirement A.7-1  
Requirement A.7-2  
Requirement A.7-3

#### ABSTRACT TEST A.8

##### NORMATIVE STATEMENTS

Requirement A.8-1  
Requirement A.8-2  
Requirement A.8-3

### A.2.5. Feature Collections {root}/collections

#### ABSTRACT TEST A.9

##### NORMATIVE STATEMENTS

Requirement A.9-1  
Requirement A.9-2  
Requirement A.9-3

## ABSTRACT TEST A.10

NORMATIVE STATEMENTS	Requirement A.10-1
	Requirement A.10-2
	Requirement A.10-3

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	<a href="#">collections.yaml</a>	/ats/html/content
GeoJSON	<a href="#">collections.yaml</a>	/ats/geojson/content
GMLSF0	<a href="#">core.xsd</a> , element core: Collections	/ats/gmlsf0/content
GMLSF2	<a href="#">core.xsd</a> , element core: Collections	/ats/gmlsf2/content

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

## ABSTRACT TEST A.11

NORMATIVE STATEMENTS	Requirement A.11-1
	Requirement A.11-2
	Requirement A.11-3

## ABSTRACT TEST A.12

NORMATIVE STATEMENTS	Requirement A.12-1
	Requirement A.12-2
	Requirement A.12-3

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

## ABSTRACT TEST A.13

### NORMATIVE STATEMENTS

Requirement A.13-1  
Requirement A.13-2  
Requirement A.13-3

## ABSTRACT TEST A.14

### NORMATIVE STATEMENTS

Requirement A.14-1  
Requirement A.14-2  
Requirement A.14-3

## ABSTRACT TEST A.15

### NORMATIVE STATEMENTS

Requirement A.15-1  
Requirement A.15-2  
Requirement A.15-3

## ABSTRACT TEST A.16

### NORMATIVE STATEMENTS

Requirement A.16-1  
Requirement A.16-2  
Requirement A.16-3

## ABSTRACT TEST A.17

### NORMATIVE STATEMENTS

Requirement A.17-1  
Requirement A.17-2  
Requirement A.17-3

## ABSTRACT TEST A.18

### NORMATIVE STATEMENTS

Requirement A.18-1  
Requirement A.18-2  
Requirement A.18-3

## ABSTRACT TEST A.19

### NORMATIVE STATEMENTS

Requirement A.19-1  
Requirement A.19-2

## ABSTRACT TEST A.19

Requirement A.19-3

## ABSTRACT TEST A.20

NORMATIVE STATEMENTS

Requirement A.20-1

Requirement A.20-2

Requirement A.20-3

## ABSTRACT TEST A.21

NORMATIVE STATEMENTS

Requirement A.21-1

Requirement A.21-2

Requirement A.21-3

## ABSTRACT TEST A.22

NORMATIVE STATEMENTS

Requirement A.22-1

Requirement A.22-2

Requirement A.22-3

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	<a href="#">featureCollectionGeoJSON.yaml</a>	/ats/html/content
GeoJSON	<a href="#">featureCollectionGeoJSON.yaml</a>	/ats/geojson/content
GMLSF0	<a href="#">core-sf.xsd</a> , element <code>sf:FeatureCollection</code>	/ats/gmlsf0/content
GMLSF2	<a href="#">core-sf.xsd</a> , element <code>sf:FeatureCollection</code>	/ats/gmlsf2/content

**Supporting Tests:**

### ABSTRACT TEST A.23

#### NORMATIVE STATEMENTS

Requirement A.23-1  
Requirement A.23-2  
Requirement A.23-3

### ABSTRACT TEST A.24

#### NORMATIVE STATEMENTS

Requirement A.24-1  
Requirement A.24-2  
Requirement A.24-3

### ABSTRACT TEST A.25

#### NORMATIVE STATEMENTS

Requirement A.25-1  
Requirement A.25-2  
Requirement A.25-3

### ABSTRACT TEST A.26

#### NORMATIVE STATEMENTS

Requirement A.26-1  
Requirement A.26-2  
Requirement A.26-3

## A.2.8. Feature

### ABSTRACT TEST A.27

#### NORMATIVE STATEMENTS

Requirement A.27-1  
Requirement A.27-2  
Requirement A.27-3

### ABSTRACT TEST A.28

#### NORMATIVE STATEMENTS

Requirement A.28-1  
Requirement A.28-2  
Requirement A.28-3

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	<a href="#">featureGeoJSON.yaml</a>	/ats/html/content
GeoJSON	<a href="#">featureGeoJSON.yaml</a>	/ats/geojson/content
GMLSF0	<a href="#">gml.xsd</a> , element substitutable for <code>gml:AbstractFeature</code>	/ats/gmlsf0/content
GMLSF2	<a href="#">gml.xsd</a> , element substitutable for <code>gml:AbstractFeature</code>	/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
	Requirement A.29-2
	Requirement A.29-3

## 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

Requirement A.30-2

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

Requirement A.31-2

Requirement A.31-3

## 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

Requirement A.32-2

Requirement A.32-3

## A.4.2. GML Simple Features 0 Content

### ABSTRACT TEST A.33

#### NORMATIVE STATEMENTS

Requirement A.33-1

Requirement A.33-2

Requirement A.33-3

## 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

Requirement A.34-2

Requirement A.34-3

## A.5.2. GML Simple Features 2 Content

### ABSTRACT TEST A.35

#### NORMATIVE STATEMENTS

Requirement A.35-1

Requirement A.35-2

Requirement A.35-3



## 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

Requirement A.36-2

Requirement A.36-3

### A.6.2. HTML Content

#### ABSTRACT TEST A.37

NORMATIVE STATEMENTS

Requirement A.37-1

Requirement A.37-2

Requirement A.37-3

## A.7. Conformance Class OpenAPI 3.0

---

### CONFORMANCE CLASS A.6

SUBJECT

Web API

#### ABSTRACT TEST A.38

NORMATIVE STATEMENTS

Requirement A.38-1

Requirement A.38-2

## ABSTRACT TEST A.38

Requirement A.38-3

## ABSTRACT TEST A.39

### NORMATIVE STATEMENTS

Requirement A.39-1

Requirement A.39-2

Requirement A.39-3

## ABSTRACT TEST A.40

### NORMATIVE STATEMENTS

Requirement A.40-1

Requirement A.40-2

Requirement A.40-3

## ABSTRACT TEST A.41

### NORMATIVE STATEMENTS

Requirement A.41-1

Requirement A.41-2

Requirement A.41-3

## ABSTRACT TEST A.42

### NORMATIVE STATEMENTS

Requirement A.42-1

Requirement A.42-2

Requirement A.42-3

## ABSTRACT TEST A.43

### NORMATIVE STATEMENTS

Requirement A.43-1

Requirement A.43-2

Requirement A.43-3



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 <https://www.w3.org/TR/dwbp/>.
- [6] W3C: W3C sdw-bp, *Spatial Data on the Web Best Practices*. World Wide Web Consortium <https://www.w3.org/TR/sdw-bp/>.
- [7] W3C: W3C vocab-dcat, *Data Catalog Vocabulary (DCAT)*. World Wide Web Consortium <https://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>



# 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 <a href="#">#2</a> , <a href="#">#5</a> , <a href="#">#6</a> , <a href="#">#7</a> , <a href="#">#8</a> , <a href="#">#14</a> , <a href="#">#15</a> , <a href="#">#19</a>
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 <a href="#">#31</a> , <a href="#">#32</a>
2018-02-11	3.0.0-SNAPSHOT	C. Portele	2, 6, 7, 8	address issue <a href="#">#25</a>
2018-02-27	3.0.0-SNAPSHOT	C. Portele	all	address issues <a href="#">#3</a> , <a href="#">#9</a> , <a href="#">#12</a> , <a href="#">#22</a> , <a href="#">#23</a> , <a href="#">#24</a> , <a href="#">#44</a> ; add links to issues <a href="#">#41</a> , <a href="#">#45</a> , <a href="#">#46</a> , <a href="#">#47</a>
2018-03-04	3.0.0-SNAPSHOT	T. Schaub	7, B	JSON schema fixes <a href="#">#54</a> , <a href="#">#55</a>
2018-03-12	3.0.0-SNAPSHOT (for ISO NWIP)	C. Portele	all	Updates after the WFS 3.0 Hackathon <a href="#">#59</a> , <a href="#">#61</a> , <a href="#">#62</a> , <a href="#">#63</a> , <a href="#">#64</a> , <a href="#">#69</a> , <a href="#">#72</a> , <a href="#">#77</a> , <a href="#">#78</a> ; resolve <a href="#">#4</a> ; editorial edits
2018-03-15	3.0.0-SNAPSHOT	J. Amara	7	Uniqueness of feature id <a href="#">#83</a>

DATE	RELEASE	EDITOR	PRIMARY CLAUSES MODIFIED	DESCRIPTION
2018-03-21	3.0.0-SNAPSHOT	I. Rinne	7	Clarified the requirement /req/core/crs84 <a href="#">#92</a>
2018-03-28	3.0.0-SNAPSHOT	C. Portele	3, 4, 7	Temporal support <a href="#">#57</a> , bbox no longer restricted to CRS84 <a href="#">#60</a> , clarify 'collection' <a href="#">#86</a> , clarify feature id constraints <a href="#">#84</a>
2018-04-02	3.0.0-SNAPSHOT	C. Portele	7, B	Clarify 'item' links <a href="#">#81</a> , clean up Open API example in Annex B
2018-04-03	3.0.0-SNAPSHOT	C. Portele	4 to 9	Clean-up asciidoc <a href="#">#100</a>
2018-04-04	3.0.0-SNAPSHOT	P. Vretanos, C. Portele	8.4, 8.5, C	Clarify XML encoding <a href="#">#58</a>
2018-04-05	3.0.0-SNAPSHOT	C. Heazel	A	Initial version of the Abstract Test Suite <a href="#">#112</a>
2018-04-05	3.0.0-SNAPSHOT	C. Portele	C	Fix axis order in example <a href="#">#113</a>
2018-04-07	3.0.0-SNAPSHOT	C. Portele	7, 9, 10	Add HTTP status code guidance <a href="#">#105</a> , add warning about OpenAPI media type <a href="#">#117</a>
2018-04-07	3.0.0-SNAPSHOT	C. Reed, C. Portele	all	Edits after review <a href="#">#119</a>
2018-04-07	<b>3.0.0-draft.1</b>	C. Portele	iv, v	First draft release
2019-02-14	3.0.0-SNAPSHOT	C. Portele, C. Holmes	all	Bugfixes <a href="#">#149</a> and <a href="#">#176</a> , change rel= item to rel=items <a href="#">#175</a> , use {collectionId}, {featureId} and id consistently <a href="#">#171</a>
2019-05-02	3.0.0-SNAPSHOT	C. Portele	all	Temporal data support <a href="#">#155</a> , extents <a href="#">#168</a> , result

DATE	RELEASE	EDITOR	PRIMARY CLAUSES MODIFIED	DESCRIPTION
				set consistency during paging <a href="#">#192</a>
2019-05-20	1.0.0-SNAPSHOT	C. Portele	all	Change document title to “OGC API – Features” <a href="#">#189</a> , minor editorial issues <a href="#">#204</a> , introduce yaml <a href="#">#201</a> , HEAD/OPTIONS <a href="#">#115</a> , /collections path structure <a href="#">#90</a> , resource names <a href="#">#199</a> , /items <a href="#">#164</a> , bbox/time parameter behavior for features without spatial/temporal data <a href="#">#122</a> , change language in overview <a href="#">#124</a> , update XMI <a href="#">#209</a>
2019-06-11	1.0.0-SNAPSHOT	C. Portele	5.6, 7.2, 7.11	Add clarification about default parameter values <a href="#">#215</a> , add title/description to landing page <a href="#">#227</a> , 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 <a href="#">#45</a> , Deviations between XML and JSON encoding of various structures <a href="#">#133</a> , Add section “Security Considerations” <a href="#">#137</a> , Issues with the UML model and resource descriptions <a href="#">#217</a>
2019-06-22	1.0.0-SNAPSHOT	C. Portele	all	Editorial cleanup, Add anchors <a href="#">#225</a>



DATE	RELEASE	EDITOR	PRIMARY CLAUSES MODIFIED	DESCRIPTION
2019-06-25	1.0.0-SNAPSHOT	C. Portele	all	Move examples in Annex B/C outside of the document <a href="#">#239</a> , Bulk download <a href="#">#230</a> , CRS 84 Requirement <a href="#">#233</a> , Endpoint /api missing from Open API specification <a href="#">#236</a>
2019-06-26	<b>1.0.0-draft.1</b>	C. Portele, C. Heazel	Annex A, all	Update Abstract Test Suite <a href="#">#112</a> , update for release of 17-069r1
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 <a href="#">#112</a>
2019-07-08	1.0.0-SNAPSHOT	C. Portele	7.4, all	Use conformance class URIs in the conformance declaration <a href="#">#244</a>
2019-07-09	1.0.0-SNAPSHOT	C. Portele	all	Clean up document
2019-07-11	<b>1.0.0-draft.2</b>	C. Portele, P. Vretanos	8.4, 8.5, A	Update XML to conform to GML Simple Features <a href="#">#150</a> , 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 <a href="#">#251</a> , allow unknown start/end in datetime <a href="#">#252</a> , clarification of wording for 3D bounding boxes <a href="#">#259</a> / <a href="#">#260</a> , prepare release

DATE	RELEASE	EDITOR	PRIMARY CLAUSES MODIFIED	DESCRIPTION
2019-10-07	1.0.0		all	Edits for publication