OGC API - Features - Part 3

Filtering

Open Geospatial Consortium

Submission Date: 2024-03-26

Approval Date: 2024-05-23

Publication Date: 2024-07-26

External identifier of this OGC® document: http://www.opengis.net/doc/IS/ogcapi-features-3/1.0

Internal reference number of this OGC® document: 19-079r2

Version: 1.0

Latest Published Draft: n/a

Category: OGC® Implementation Specification

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

OGC API - Features - Part 3: Filtering

Document type:	OGC® Standard
Document subtype:	Interface
Document stage:	Approved
Document language:	English

License Agreement

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

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

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

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

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

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

Table of Contents

i. Abstract

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

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 part is a separate standard.

A fundamental operation performed on a collection of features is that of filtering in order to obtain a subset of the data which contains feature instances that satisfy some filtering criteria. Part three of the OGC API - Features Standard defines query parameters (`filter`, `filter-lang`, `filter-crs`) to specify filter criteria in a request to an API and the Queryables resource that declares the properties of data in a collection that can be used in filter expressions.

ii. Keywords

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

OGC, filter, expression, query, SQL, CQL2, where clause, selection clause, OGC API

iii. Preface

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.

iv. Submitting organizations

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

- CubeWerx Inc.
- Ecere Corporation
- GeoSolutions di Giannecchini Simone & C. s.a.s.
- interactive instruments GmbH
- US Army Geospatial Center (AGC)

v. Submitters

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

Name	**Affiliation**
Panagiotis (Peter) A. Vretanos *(editor)*	CubeWerx Inc.
Clemens Portele *(editor)*	interactive instruments GmbH

Andrea Aime	GeoSolutions di Giannecchini Simone & C. s.a.s.
Jeff Harrison	US Army Geospatial Center (AGC)
Jérôme Jacovella-St-Louis	Ecere Corporation

Chapter 1. Scope

This document specifies an extension to the OGC API - Features - Part 1: Core standard that defines the behavior of a server that supports enhanced filtering capabilities expressed using the Common Query Language (CQL2).

Enhanced filtering capabilities in this case means that the server supports the ability to define selection clauses using predicates beyond those supported by Part 1 (i.e., `bbox` and `datetime`).

This document defines

- Query parameters for specifying a filter in a request to a Web API;
- Support for CQL2 Text and CQL2 JSON as languages for filter expressions;
- How the set of properties that can be used to construct filter expressions ("queryables") are published by a Web API.

Chapter 2. Conformance

This standard defines the following requirements classes, grouped by their standardization target:

- Web API
  - Queryables
  - Queryables as Query Parameters
  - Filter
  - Features Filter

The Queryables requirements class defines the Queryables resource (at path `/collections/{collectionId}/queryables`) and its representation as a JSON Schema. Queryables can be used to determine the list of property names and their schemas that may be used to construct filter expressions.

The Queryables as Query Parameters requirements class adds the requirement to provide query parameters for queryables according to the recommendation in the section Parameters for filtering on feature properties in OGC API - Features - Part 1: Core.

The Filter requirements class defines a set of HTTP query parameters that may be used to specify complex filter expressions on HTTP requests. The specific set of parameters defined in this requirements class is:

- `filter` - The filter expression.
- `filter-lang` - The language used in the filter expression.
- `filter-crs` - The coordinate reference system used in the filter expression, if Part 2 is supported.

The Features Filter requirements class defines the binding between the Filter requirements class and the OGC API - Features - Part 1: Core standard.

Conformance with this standard shall be checked using all the relevant tests specified in Annex A 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.

Table 1. Conformance class URIs

Conformance class	URI
Queryables	`http://www.opengis.net/spec/ogcapi-features-3/1.0/conf/queryables`
Queryables as Query Parameters	`http://www.opengis.net/spec/ogcapi-features-3/1.0/conf/queryables-query-parameters`
Filter	`http://www.opengis.net/spec/ogcapi-features-3/1.0/conf/filter`
Features Filter	`http://www.opengis.net/spec/ogcapi-features-3/1.0/conf/features-filter`

APIs that implement the Common Query Language (CQL2) Standard should advertize all supported CQL2 conformance classes in the Conformance Declaration, too.

2.1. Roadmap

The content of this sub-clause is informative.

Because the filter parameters are not exclusively useful for features, it is anticipated that the Queryables and Filter requirements classes will eventually become parts of the OGC API - Common suite of standards thus leaving the Queryables as Query Parameters and Features Filter requirements classes as part 3 of the OGC API - Features Standard.

Chapter 3. References

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

- Open Geospatial Consortium (OGC). OGC 17-069r4: **OGC API - Features - Part 1: Core corrigendum** [online]. Edited by C. Portele, P. Vretanos, C. Heazel. 2022 [viewed 2023-02-19]. Available at https://docs.ogc.org/is/17-069r4/17-069r4.html
- Open Geospatial Consortium (OGC). OGC 18-058r1: **OGC API - Features - Part 2: Coordinate Reference Systems by Reference corrigendum** [online]. Edited by C. Portele, P. Vretanos. 2022 [viewed 2023-02-19]. Available at https://docs.ogc.org/is/18-058r1/18-058r1.html
- Open Geospatial Consortium (OGC). OGC 21-065r1: **Common Query Language (CQL2)** [online]. Edited by P. Vretanos, C. Portele. to be published. Available at https://docs.ogc.org/is/21-065r1/21-065r1.html
- Internet Engineering Task Force (IETF). draft-bhutton-json-schema-01: **JSON Schema: A Media Type for Describing JSON Documents** [online]. Edited by A. Wright, H. Andrews, B. Hutton, G. Dennis. 2022 [viewed 2024-02-28]. Available at https://json-schema.org/draft/2020-12/json-schema-core

Chapter 4. Terms, Definitions, Symbols and Abbreviated Terms

4.1. Terms and Definitions

This document used 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 standard 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.

4.1.1. collection

a body of resources that belong or are used together; an aggregate, set, or group of related resources (OGC 20-024, OGC API - Common - Part 2: Collections).

4.1.2. endpoint

a web address (URI) at which access can be gained to a service or resource

4.1.3. filter expression

predicate encoded for transmission between systems

4.1.4. predicate

set of computational operations applied to a data instance which evaluate to true or false (OGC Filter Encoding 2.0 Encoding Standard - With Corrigendum)

4.1.5. publisher

entity responsible for making a resource available (Dublin Core Metadata Initiative - DCMI Metadata Terms)

NOTE	As content of OGC API Standards, a resource is typically published at an endpoint.

4.1.6. queryable

a token that represents a property of a resource that can be used in a **filter expression**

4.1.7. resource

entity that might be identified (Dublin Core Metadata Initiative - DCMI Metadata Terms)

NOTE	The term "resource", when used in the context of an OGC API Standard, should be understood to mean a web resource unless otherwise indicated.

4.1.8. web resource

a **resource** that is identified by a HTTP URI.

4.2. Abbreviated terms

ABNF
: Augmented Backus-Naur Form

API
: Application Programming Interface

BNF
: Backus-Naur Form

CQL2
: Common Query Language

CRS
: Coordinate Reference System

HTTP
: Hypertext Transfer Protocol

HTTPS
: Hypertext Transfer Protocol Secure

IANA
: Internet Assigned Numbers Authority

JSON
: JavaScript Object Notation

OGC
: Open Geospatial Consortium

URI
: Uniform Resource Identifier

WKT
: Well-Known Text

YAML

YAML Ain’t Markup Language

Chapter 5. Conventions and background

5.1. General remarks

See OGC API - Features - Part 1: Core, Clauses 5 and 6.

5.2. Identifiers

The normative provisions in this standard are denoted by the URI `http://www.opengis.net/spec/ogcapi-features-3/1.0`.

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

5.3. Additional link relation types

The following OGC link relation types are introduced in this document (no applicable link relation type in the IANA link relation type register could be identified):

- **http://www.opengis.net/def/rel/ogc/1.0/queryables**: Refers to a resource that lists properties that can be used to query sub-resources of the link's context.

As an alternative, the Compact URI (CURIE) **[ogc-rel:queryables]** can be used, too, where a CURIE is an allowed value.

5.4. HTTP URIs

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. Not all URIs in this document are properly percent-encoded for better readability.

5.5. Dependencies to other requirements classes

The requirements classes in this extension distinguish two types of dependencies to other specifications or requirements classes:

First, there are the obligatory dependencies. Every server implementing the requirements class has to conform to the referenced specification or requirements class.

In addition, requirements classes can also have conditional dependencies. Servers implementing the requirements class do not have to conform to the referenced specification or requirements class, but if they do, they have to conform to the requirements that identify the conditional dependency as a pre-condition for the normative statement.

Chapter 6. Requirements Class "Queryables"

Requirements Class	
http://www.opengis.net/spec/ogcapi-features-3/1.0/req/queryables	
Target type	Web API

This requirements class defines the Queryables resource for discovering a list of resource properties with their types and constraints that may be used to construct filter expressions on a collection of resources, for example, a set of features.

This Standard does not assume that the content schema of a resource being queried is available for inspection. Therefore, a means needs to exist to interrogate an endpoint to determine the names and types of the properties that may be used to construct a filter expression ("queryables").

In addition, a publisher may want to support queryables that are not directly represented as resource properties in the content schema of the resource. Or the publisher may want to restrict filtering on certain properties. For example, because the backend datastore has not been configured to allow high-performance queries on those properties.

Requirement 1	/req/queryables/queryables-link
A	The Queryables resource SHALL be referenced from any filterable resource with a link with the link relation type `http://www.opengis.net/def/rel/ogc/1.0/queryables` (or, alternatively, `[ogc-rel:queryables]`, where a CURIE is explicitly allowed as a value).

It is recommended to include the link in the response via an HTTP header.

Requirement 2	/req/queryables/get-queryables-op
A	The Queryables resource SHALL support the HTTP GET operation and the media type `application/schema+json`.

The response is returned as a JSON Schema document that describes a single JSON object where each property is a queryable.

Requirement 3	/req/queryables/get-queryables-response
A	A successful execution of the operation SHALL be reported as a response with an HTTP status code of `200`.

B	For responses that use `application/schema+json` as the `Content-Type` of the response, the response SHALL have the following characteristics: • The property `$schema` is `https://json-schema.org/draft/2020-12/schema`. • The property `$id` is the URI of the resource without query parameters. • The `type` is `object` and each property is a queryable.
C	Each property SHALL include a `type` member, except for spatial properties.
D	Each spatial property SHALL not include a `type` or `$ref` member.
E	Each spatial property SHALL include a `format` member with a string value "geometry", followed by a hyphen, followed by the name of the geometry type in lower case. I.e., the values for the Simple Feature geometry types are: "geometry-point", "geometry-multipoint", "geometry-linestring", "geometry-multilinestring", "geometry-polygon", "geometry-multipolygon", and "geometry-geometrycollection". In addition, the following special values are supported: "geometry-any" as the wildcard for any geometry type, "geometry-point-or-multipoint" for a Point or MultiPoint, "geometry-linestring-or-multilinestring" for a LineString or MultiLineString, and "geometry-polygon-or-multipolygon" for a Polygon or MultiPolygon.
F	Each temporal property SHALL be a string literal with the appropriate format (e.g., `date-time` or `date` for instances, depending on the temporal granularity).
G	The `additionalProperties` member with a value of `true` or `false` is used to state the behavior with respect to properties that are not explicitly declared in the queryables schema. If `additionalProperties` is missing or has the default value `true`, any property name is valid in a filter expression and the property reference SHALL evaluate to `null`, if the property does not exist for a resource. If `additionalProperties` is set to `false`, property references that are not explicitly declared in the queryables schema SHALL result in a `400` response.

NOTE	The queryables schema does not specify a schema of any object that can be retrieved from the API. JSON Schema is used for the queryables to have a consistent approach for describing schema information. The queryables schema describes a "virtual" resource view for the purpose of filtering. JSON Schema will also be used in Part 5: Schemas of OGC API Features to describe schemas for feature content.

To support clients, providing additional detail about the meaning of the queryable properties and their value range is recommended:

Recommendation 1	**/rec/queryables/queryables-schema**
A	Each property SHOULD have a human readable title (`title`) and, where necessary for the understanding of the property, a description (`description`).
B	The `type` SHOULD be one of the following values: `string` (string or temporal properties), `number`/`integer` (numeric properties), `boolean` (boolean properties), `object` (object properties) or `array` (array properties).
C	Properties that represent a URI SHOULD be represented as a string with format `uri` or `uri-reference`.
D	Properties that represent a URI template SHOULD be represented as a string with format `uri-template`.
E	Properties that represent a UUID SHOULD be represented as a string with format `uuid`.
F	For string properties that are , `minLength`, `maxLength`, `enum` and/or `pattern` SHOULD be provided, where applicable.
G	For numeric properties, `multipleOf`, `minimum`, `exclusiveMinimum`, `maximum`, `exclusiveMaximum` SHOULD be provided, where applicable.
H	For integer properties that represent enumerated values, `enum` SHOULD be provided.
I	For array properties, the property SHOULD consist of items that are strings or numbers.
J	The JSON Schema keywords SHOULD be constrained to the those mentioned in this recommendation and requirement `/req/queryables/get-queryables-response`.

Example 1. Queryables example

```
{
  "$schema" : "https://json-schema.org/draft/2020-12/schema",
  "$id" :
"https://demo.ldproxy.net/zoomstack/collections/roads_national/queryables",
  "type" : "object",
  "title" : "National Roads",
  "description" : "Lines representing the road network. A road is defined as a
metalled way for vehicles.",
  "properties" : {
    "geom" : {
      "title": "Centerline",
      "description": "The geometry of the road.",
      "x-ogc-role" : "primary-geometry",
```

```
      "format" : "geometry-linestring"
    },
    "name" : {
      "title" : "Road Name",
      "description": "The common, human readable, name of the road.",
      "type" : "string"
    },
    "number" : {
      "title" : "Road Identifier",
      "description": "The official number of the road.",
      "type" : "string"
    },
    "type" : {
      "title" : "Type",
      "description": "The road type, one of 'Primary' or 'Motorway'.",
      "enum" : [ "Primary", "Motorway" ],
      "type" : "string"
    },
    "level" : {
      "title" : "Level",
      "enum" : [ 0, 1, 2 ],
      "type" : "integer"
    }
  },
  "additionalProperties" : false
}
```

Example 2. Filtering on complex data structures

Queryables also support simplified filtering on complex data structures. This example illustrates one potential approach to define a queryable on specific SpatioTemporal Asset Catalog (STAC) assets. The following is a STAC snippet:

```
"features": [
  {
    "type": "Feature",
    "assets": {
      "B5": {
        "eo:bands": [
          {
            "common_name": "nir",
            "name": "B5"
          }
        ],
        "href": "https://landsat-pds.s3.us-west-
2.amazonaws.com/c1/L8/060/247/LC08_L1TP_060247_20180905_20180912_01_T1/LC08_L1TP_0
60247_20180905_20180912_01_T1_B5.TIF",
        "type": "image/tiff; application=geotiff; profile=cloud-optimized"
      }
```

```
    }
  }
]
```

A common type of filter is to fetch only resources that have in their `assets` object a value that has an `eo:bands` value that contains a `common_name` value of "nir".

This could be implemented using a queryable `common_band_names` defined as an array of strings:

```
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "$id": "https://example.net/stac/collections/landsat/queryables",
  "type": "object",
  "title": "A STAC item",
  "properties": {
    "common_band_names": {
      "title": "Common band names",
      "description": "an array of common band names included in the assets",
      "type" : "array",
      "items": {
        "type": "string"
      }
    }
  }
}
```

Internally this could be mapped to the following JSON Path for a STAC item (`item`):

`item['assets'][*]['eo:bands'][\*]['common_name']`

Note that the JSON Path is just included here to clarify the mapping. An implementation would use a mapping that fits its backend datastore, but these details are hidden from the user.

A client can now use, e.g., the CQL2 array predicates to filter with the queryable. Examples using the CQL2 text encoding:

- `A_CONTAINS(common_band_names, ["nir"])` - includes at least a 'nir' band
- `A_EQUALS(common_band_names, ["nir","blue"])` - 'nir' and 'blue' exist for the feature, but no other band

Chapter 7. Requirements Class "Queryables as Query Parameters"

Requirements Class	
http://www.opengis.net/spec/ogcapi-features-3/1.0/req/queryables-query-parameters	
Target type	Web API
Dependency	OGC API - Features - Part 1: Core, Requirements Class "Core"
Dependency	Requirements Class "Queryables"

This requirements class specifies a requirement to provide query parameters for queryables according to the recommendation in the section Parameters for filtering on feature properties in OGC API - Features - Part 1: Core.

Requirement 4	/req/queryables-query-parameters/parameters
A	For every feature collection, the server SHALL support the Queryables resource at the path `/collections/{collectionId}/queryables`.
B	The parameter `collectionId` is each `id` property in the Collections resource (JSONPath: `$.collections[*].id`).
C	For every queryable of a feature collection that has a simple value (string, number, integer or boolean), the collection SHALL support a query parameter at path `/collections/{collectionId}/items` with the same schema as the schema of the queryable.
D	If the query parameter is provided in a request, the response SHALL only include resources that match the provided value for the queryable.

Example 3. Queryables as Query Parameters example

The OpenAPI 3.0 definitions of the query parameters for the Queryables in Queryables example would be:

```
type_roads_national:
  name: type
  in: query
  description: Filter the collection by property 'type' (Road Type).
  required: false
  style: form
  explode: false
  schema:
    title: Road Type
    description: The road type, one of 'Primary' or 'Motorway'.
```

```
    type: string
    enum:
    - Primary
    - Motorway
name_roads_national:
  name: name
  in: query
  description: Filter the collection by property 'name' (Road Name).
  required: false
  style: form
  explode: false
  schema:
    title: Road Name
    description: The common, human readable, name of the road.
    type: string
number_roads_national:
  name: number
  in: query
  description: Filter the collection by property 'number' (Road Identifier).
  required: false
  style: form
  explode: false
  schema:
    title: Road Identifier
    description: The official number of the road.
    type: integer
    minimum: 1
    maximum: 9999
```

Chapter 8. Requirements Class "Filter"

8.1. Overview

Requirements Class	
http://www.opengis.net/spec/ogcapi-features-3/1.0/req/filter	
Target type	Web API
Dependency	Requirements Class "Queryables"

OGC API - Features - Part 1: Core defines two filtering parameters on the resource at path `/collections/{collectionId}/items`: `bbox` and `datetime`. OGC API - Features - Part 1: Core also adds support for simple equality predicates logically joined using the `AND` operator. These capabilities offer simple resource filtering for HTTP requests.

The Filter requirements class defines additional query parameters that allow more complex filtering expressions to be specified when querying server resources.

Specifically, this clause defines the parameters, `filter`, `filter-lang` and `filter-crs`.

8.2. Parameter filter

The Filter requirements class defines a general parameter, `filter`, whose value is a filter expression to be applied when retrieving resources. This is necessary to determine which resources should be included in a result set.

Requirement 5	/req/filter/filter-param

A	The HTTP GET operation on the path that fetches resource instances (e.g. `/collections/{collectionId}/items`) SHALL support a parameter `filter` based on the following OpenAPI 3.0 fragment:
	`name: filter`
	`in: query`
	`required: false`
	`schema:`
	`  type: string`
	`style: form`
	`explode: false`

Recommendation 2	/rec/filter/queryables-schema
A	The `filter` parameter MAY specify a default value.

8.3. Parameter filter-lang

Any predicate language that can be suitably expressed as the value of an HTTP query parameter may be specified as the value of the `filter` parameter. In order to specify that specific language that is being used, this clause defines the `filter-lang` parameter.

Requirement 6	/req/filter/filter-lang-param

A	The HTTP GET operation on the path that fetches resource instances (e.g. `/collections/{collectionId}/items`) SHALL support a parameter `filter-lang` based on the following OpenAPI 3.0 fragment:
	`name: filter-lang`
	`in: query`
	`required: false`
	`schema:`
	`  type: string`
	`  enum:`
	`  - 'cql2-text'`
	`  - 'cql2-json'`
	`  default: 'cql2-text'`
	`style: form`

B	The `enum` array in the schema of `filter-lang` SHALL list the filter encodings that the server supports for the resource.
C	The `default` value in the schema of `filter-lang` SHALL identify the filter encoding that the server will assume, if a `filter` is provided, but no `filter-lang`.

The enumerated value `cql2-text` is used to indicate that the value of the `filter` parameter is the text encoding of CQL2.

The enumerated value `cql2-json` is used to indicate that the value of the `filter` parameter is the JSON encoding of CQL2.

Servers that support other filtering languages can extend this list of values as necessary although the meanings of any additional values are not described in this Standard.

8.4. Parameter filter-crs

For reasons discussed in the W3C/OGC Spatial Data on the Web Best Practices document, spatial coordinates by default are in the coordinate reference system WGS 84 longitude and latitude for 2D coordinates and WGS 84 longitude, latitude and ellipsoidal height in meters for 3D coordinates. For servers that support geometries in other coordinate reference systems, the `filter-crs` parameter defined in this clause allows clients to assert which CRS is being used to encode geometric values in a filter expression. Otherwise, the `filter-crs` parameter has no use.

Requirement 7	/req/filter/filter-crs-wgs84
A	If a HTTP GET operation on the path that fetches resource instances (e.g. `/collections/{collectionId}/items`) includes a `filter` parameter, but no `filter-crs` parameter, the server SHALL process all geometries in the filter expression using CRS84 (for coordinates without height) or CRS84h (for coordinates with ellipsoidal height) as the coordinate reference system (CRS).

Requirement 8	/req/filter/filter-crs-param
Condition	Server supports additional coordinate reference systems

A	The HTTP GET operation on the path that fetches resource instances (e.g. `/collections/{collectionId}/items`) SHALL support a parameter `filter-crs` based on the following OpenAPI 3.0 fragment:
	
	`name: filter-crs`
	`in: query`
	`required: false`
	`schema:`
	`  type: string`
	`  format: uri-reference`
	`style: form`
	`explode: false`

B	If an HTTP GET operation on the path that fetches resource instances (e.g. `/collections/{collectionId}/items`) includes the `filter` and the `filter-crs` parameter, the server SHALL process all geometries in the filter expression using the CRS identified by the URI in `filter-crs`.
C	The server SHALL return an error, if it does not support the CRS identified in `filter-crs` for the resource.

NOTE	Discovery of the list of supported coordinate reference systems for use with the `filter-crs` parameter depends on the filterable resource.

Recommendation 3	/rec/filter/filter-crs-list
A	The server SHOULD list the supported coordinate reference system URIs as enums in the schema of the `filter-crs` parameter.

8.5. Interaction with other predicates

Servers can and often will support additional filtering parameters on filterable resources. This clause defines how the `filter` parameter and other filtering parameters should interact if specified in a single request.

Requirement 9	/req/filter/mixing-expressions
A	Other filter predicates supported by the server (e.g. `bbox`, `datetime`, etc.) SHALL be logically connected with the `AND` operator when mixed in a request with the `filter` parameter.

8.6. CQL2 functions

If the server supports CQL2 and the requirements class "Functions", a resource, `/functions` is published that allows clients to discover the list of functions that a server offers.

Requirement 10	/req/filter/get-functions-operation
Condition	Server implements Common Query Language (CQL2), requirements class "Functions"
A	A server SHALL support the HTTP GET operation at the `/functions` path.

Requirement 11	/req/filter/get-functions-response-json
Condition	Server implements Common Query Language (CQL2), requirements class "Functions"
A	A successful execution of the operation SHALL be reported as a response with a HTTP status code `200`.

B	The content of that response SHALL be based upon the following OpenAPI 3.0 schema and list all functions that the server supports:

```
type: object
required:
- functions
properties:
  functions:
    type: array
    items:
      type: object
      required:
      - name
      - returns
      properties:
        name:
          type: string
        description:
          type: string
        metadataUrl:
          type: string
          format: uri-reference
        arguments:
          type: array
          items:
            type: object
            required:
            - type
            properties:
              title:
                type: string
              description:
                type: string
              type:
                type: array
                items:
                  type: string
                  enum:
                  - string
                  - number
                  - integer
                  - datetime
                  - geometry
                  - boolean
        returns:
          type: array
          items:
            type: string
            enum:
```

Example 4. Get functions JSON response example


```
{
   "functions": [
      {
         "name": "min",
         "arguments": [
            {
               "type": ["string","number","datetime"]
            },
            {
               "type": ["string","number","datetime"]
            },
         ],
         "returns": ["string","number","datetime"]
      },
      {
         "name": "max",
         "arguments": [
            {
               "type": ["string","number","datetime"]
            },
            {
               "type": ["string","number","datetime"]
            },
         ],
         "returns": ["string","number","datetime"]
      },
      {
         "name": "geometryType",
         "arguments": [
            {
               "type": ["geometry"]
            }
         ],
         "returns": ["string"]
      }
   ]
}
```


8.7. Filter expression languages

This Standard only specifies `filter-lang` values for the Common Query Language (CQL2). However, support for this filter expression language is not mandatory and other languages can be used as the value of the filter parameter, too.

8.8. Response

A filter expression defines a subset of items from a collection that should be presented in a query response.

Requirement 12	**/req/filter/response**
A	The filter expression SHALL be evaluated for each item of the collection being queried.
B	If the filter expression evaluates to `TRUE` then the item SHALL be included in the result set.
C	If the filter expression evaluates to `FALSE` then the item SHALL be excluded from the result set.

Chapter 9. Requirements Class "Features Filter"

9.1. Overview

Requirements Class	
http://www.opengis.net/spec/ogcapi-features-3/1.0/req/features-filter	
Target type	Web API
Dependency	Requirements Class "Filter"
Dependency	OGC API - Features - Part 1: Core, Requirements Class "Core"
Conditional Dependency	OGC API - Features - Part 2: Coordinate Reference Systems by Reference

This clause defines the binding between the OGC API - Features - Part 1: Core Standard and the filter parameters defined in the Filter conformance class as well as the Queryables resource defined in the Queryables conformance class.

9.2. Queryables

9.2.1. Operation

Requirement 13	/req/features-filter/get-queryables-op
A	For every feature collection, the server SHALL support the Queryables resource at the path `/collections/{collectionId}/queryables`.
B	The parameter `collectionId` is each `id` property in the Collections resource (JSONPath: `$.collections[*].id`).

9.3. Feature Collection

9.3.1. Response

Requirement 14	/req/features-filter/queryables-link
A	The Queryables resource SHALL be referenced from each Collection resource with a link with the link relation type `http://www.opengis.net/def/rel/ogc/1.0/queryables` (or, alternatively, `[ogc-rel:queryables]`).

9.4. Features

9.4.1. Operation

As per the OGC API - Features - Part 1: Core Standard, features are accessed using the HTTP GET method via the `/collections/{collectionId}/items` path (see Features). The following additional requirements bind the parameters filter, filter-lang and filter-crs to the GET operation on this path.

Requirement 15	**/req/features-filter/filter-param**
A	The HTTP GET operation on the `/collections/{collectionId}/items` path SHALL support the `filter` parameter as defined in the Parameter filter clause.

Requirement 16	**/req/features-filter/filter-lang-param**
A	The HTTP GET operation on the `/collections/{collectionId}/items` path SHALL support the `filter-lang` parameter as defined in the Parameter filter-lang clause.

Recommendation 4	**/rec/features-filter/text-encoding**
A	If a filter expression can be represented for its intended use as text, servers SHOULD support the CQL2 text encoding.

Recommendation 5	**/rec/features-filter/JSON-encoding**
A	If a filter expression can be represented for its intended use as JSON, servers SHOULD support the CQL2 JSON encoding.

Requirement 17	**/req/features-filter/filter-crs-param**
Condition	Server implements OGC API - Features - Part 2: Coordinate Reference Systems by Reference
A	The HTTP GET operation on the `/collections/{collectionId}/items` path SHALL support the `filter-crs` parameter as defined in the Parameter filter-crs clause.

Discovery of the list of supported coordinate reference systems for use with the filter-crs parameter is fully described in OGC API - Features - Part 2: Coordinate Reference Systems by Reference. Briefly, the list of supported CRSs can be found in the global list of CRS identifiers (path: `/collections/crs`) and/or the list of collection-specific CRS identifiers (path: `/collections/{collectionsId}/crs`).

9.4.2. Response

Requirement 18	**/req/features-filter/response**

A	A filter expression SHALL be evaluated for each feature of a collection.
B	All other filtering parameters specified (i.e. zero or more of bbox, datetime and property filters) SHALL be evaluated for each feature of a collection.
C	If the filter expression AND all other specified filtering parameters (i.e. zero or more of bbox, datetime and property filters) evaluate to `TRUE` then the feature SHALL be included in the result set.
D	If the filter expression OR any other specified filtering parameter (i.e. zero or more of bbox, datetime and property filters) evaluates to `FALSE` then the feature SHALL be excluded from the result set.
E	The server SHALL respond with a `400` error, if a parameter value of `filter`, `filter-lang` or `filter-crs` is invalid.

Chapter 10. Media Types

See OGC API - Features - Part 1: Core, Clause 10.

Chapter 11. Security Considerations

See OGC API - Features - Part 1: Core, Clause 11.

Annex A: Abstract Test Suite (Normative)

This test suite uses the Given-When-Then notation to specify the tests.

In order to execute tests against the Web API under test, the Web API needs to support one of the specified encodings. Since all known implementations at this time support JSON, this test suite uses the JSON encoding and adds a dependency to the Conformance Class "JSON" in OGC API - Common - Part 1: Core for the general "Filter" tests and to the Conformance Class "GeoJSON" in OGC API - Features - Part 1: Core for the feature-specific tests.

The Web API under test can require authorization. Any Executable Test Suite implementing this test suite should implement the following security schemes supported by OpenAPI 3.0: HTTP Authorization schemes "basic" and "bearer", API keys, and OAuth2 flow "authorizationCode".

A.1. Conformance Class "Queryables"

Conformance Class	
http://www.opengis.net/spec/ogcapi-features-3/1.0/conf/queryables	
Target type	Web API
Requirements class	Requirements Class "Queryables"
Dependency	OGC API - Common - Part 1: Core, Conformance Class "JSON"

NOTE	The Conformance Class "JSON" in Common Core has a dependency to the Conformance Classes "Core" and "Landing Page"; that is, testing against "JSON" will automatically test against the dependencies.

The following table lists input given to all tests in this conformance class:

- The landing page URI of the OGC Web API (`{apiURI}`);
- A list of filterable resources in the API (`{apiURI}/{pathToResource}`);
- Authentication credentials (optional);

A.1.1. Conformance Test 1

Test id:	/conf/queryables/get-conformance
Requirements:	n/a
Test purpose:	Check that the API declares support for the conformance class

Test method:	Given: • n/a When: • the request for the Conformance Declaration is executed ◦ method: `GET` ◦ path: `{apiURI}/conformance` ◦ authentication, if authentication credentials are provided Then: • assert successful execution (status code is "200", `Content-Type` header is "application/json"); • assert that `$.conformsTo` is a string array that includes the value "http://www.opengis.net/spec/ogcapi-features-3/1.0/conf/queryables".

A.1.2. Conformance Test 2

Test id:	/conf/queryables/get-queryables-uris
Requirements:	/req/queryables/queryables-link
Test purpose:	Check that a link to the Queryables resource exists for every filterable resource
Test method:	Given: • the list of filterable resources (`{apiURI}/{pathToResource}`); When: • a request is executed for every filterable resource ◦ method: `HEAD` (if `HEAD` results in a 405 response, use `GET` instead) ◦ path: `{apiURI}/{pathToResource}` ◦ header: `Accept: application/json` ◦ authentication, if authentication credentials are provided Then: • assert successful execution (status code is "200"); • assert that the response includes a `Link` header with `rel` set to `http://www.opengis.net/def/rel/ogc/1.0/queryables`; • store the `href` value as the Queryables URI for the filterable resource ({queryablesUri}).

A.1.3. Conformance Test 3

Test id:	/conf/queryables/get-queryables
Requirements:	/req/queryables/get-queryables-op; /req/queryables/get-queryables-response
Test purpose:	Check that the Queryables resource exists for every filterable resource

Test method:	Given: • test "get-queryables-uris" was successful • the list of Queryables URIs for the filterable resources (list of {queryablesUri}) When: • the request for the Queryables page is executed for every filterable resource ◦ method: `GET` ◦ path: `{queryablesUri}` ◦ header: `Accept: application/schema+json` ◦ authentication, if authentication credentials are provided Then: • assert successful execution (status code is "200", `Content-Type` header is "application/schema+json"); • assert that the value of the `$schema` member is "https://json-schema.org/draft/2020-12/schema" • assert that the value of the `$id` member is "{queryablesUri}". • assert that the value of the `type` member is "object". • assert that `$.properties` is a non-empty object; • assert that each member in `$.properties` has an object as its value and the object either includes `type` member or a `format` member whose value starts with `geometry-`; • assert that the response is a valid JSON Schema; • store the key of an arbitrary property with a `type` member as the sample queryable of the filterable resource; • store the key of an arbitrary property of the object as the spatial queryable of the filterable resource, if the value of member is an object that includes no `type` member and a `format` member with a value `geometry-{type}` where `{type}` is one of "point", "multipoint", "linestring", "multilinestring", "polygon", "multipolygon", "geometrycollection", "any", "point-or-multipoint", "linestring-or-multilinestring", or "polygon-or-multipolygon"; • store the value of the `additionalProperties` member or `true`, if it is not provided.

NOTE	Sub-requirement G of `/req/queryables/get-queryables-response` can only be checked when executing filter queries in dependent conformance classes.

A.2. Conformance Class "Queryables as Query Parameters"

Conformance Class	
http://www.opengis.net/spec/ogcapi-features-3/1.0/conf/queryables-query-parameters	
Target type	Web API
Requirements class	Requirements Class "Queryables as Query Parameters"
Dependency	Conformance Class "Queryables"

The following table lists input given to all tests in this conformance class:

- The landing page URI of the OGC Web API (`{apiURI}`);
- Authentication credentials (optional);
- A list of filterable resources in the API (`{apiURI}/{pathToResource}`), where `{pathToResource}` is `/collections/{collectionId}/items`;
- The media type of the response when accessing these resources (`{responseMediaType}`)
- The list of acceptable status codes for a successful filter execution (default: "200");
- The list of acceptable status codes for an unsuccessful filter execution (default: "400");
- A valid value for each queryable `{queryable}`;
- An invalid valid value for each queryable `{queryable}`;
- The information stored during the execution of conformance tests of conformance class "Queryables".

A.2.1. Conformance Test 4

Test id:	/conf/queryables-query-parameters/get-conformance
Requirements:	n/a
Test purpose:	Check that the API declares support for the conformance class

Test method:	Given: • n/a When: • the request for the Conformance Declaration is executed ◦ method: `GET` ◦ path: `{apiURI}/conformance` ◦ header: `Accept: application/json` ◦ authentication, if authentication credentials are provided Then: • assert successful execution (status code is "200", `Content-Type` header is "application/json"); • assert that `$.conformsTo` is a string array that includes the value "http://www.opengis.net/spec/ogcapi-features-3/1.0/conf/queryables-query-parameters".

A.2.2. Conformance Test 5

Test id:	/conf/queryables-query-parameters/query-param
Requirements:	/req/queryables-query-parameters/parameters
Test purpose:	Check that query parameters for queryables are supported

Test method:	Given: • test "get-queryables" was successful • the list of collections • the sample queryable of every collection When: • a request for every filterable resource that supports filtering is executed and every queryable (`queryable`) with a valid value for the queryable (`{valid-value}`) ◦ method: `GET` ◦ path: `{apiURI}/collections/{collectionId}/items` ◦ query parameters (before percent encoding): `{queryable}={valid-value}` ◦ header: `Accept: {responseMediaType}` ◦ authentication, if authentication credentials are provided Then: • assert successful execution (the status code is in the list of acceptable status codes for a successful execution, `Content-Type` header is `{responseMediaType}`); • assert that each returned resource matches the filter. When: • a request for every filterable resource that supports filtering is executed and every queryable (`queryable`) with an invalid value for the queryable (`{invalid-value}`) ◦ method: `GET` ◦ path: `{apiURI}/collections/{collectionId}/items` ◦ query parameters (before percent encoding): `{queryable}={invalid-value}` ◦ header: `Accept: {responseMediaType}` ◦ authentication, if authentication credentials are provided Then: • assert unsuccessful execution (the status code is in the list of acceptable status codes for an unsuccessful execution).

A.3. Conformance Class "Filter"

Conformance Class	
http://www.opengis.net/spec/ogcapi-features-3/1.0/conf/filter	
Target type	Web API
Requirements class	Requirements Class "Filter"
Dependency	Conformance Class "Queryables"

The following table lists input given to all tests in this conformance class:

- The landing page URI of the OGC Web API (`{apiURI}`);
- Authentication credentials (optional);
- The list of filterable resources in the API (`{apiURI}/{pathToResource}`);
- The media type of the response when accessing these resources (`{responseMediaType}`)
- The list of acceptable status codes for a successful filter execution (default: "200", "204");
- The list of acceptable status codes for an unsuccessful filter execution (default: "400");
- The name of the filter language to test (`{filter-lang}`; default: "cql2-text");
- A flag that indicates whether the filter language is the default filter language;
- A valid filter expression in the filter language for a queryable `{queryable}` (`{filter-valid}`; default: `{queryable} IS NULL`);
- An invalid filter expression in the filter language (`{filter-invalid}`; default: `THIS IS NOT A FILTER`);
- A valid bbox filter expression in the filter language for a spatial queryable `{spatialQueryable}` and two longitude/latitude positions in WGS 84 `{x1}`/`{y1}` and `{x2}`/`{y2}` (`{bbox-filter}`; default: `S_INTERSECTS({spatialQueryable},BBOX({x1},{y1},{x2},{y2})`);
- A flag that indicates whether the API supports custom functions in filter expressions;
- The information stored during the execution of conformance tests of conformance class "Queryables".

A.3.1. Conformance Test 6

Test id:	/conf/filter/get-conformance
Requirements:	n/a
Test purpose:	Check that the API declares support for the conformance class

Test method:	Given: • n/a When: • the request for the Conformance Declaration is executed ◦ method: `GET` ◦ path: `{apiURI}/conformance` ◦ header: `Accept: application/json` ◦ authentication, if authentication credentials are provided Then: • assert successful execution (status code is "200", `Content-Type` header is "application/json"); • assert that `$.conformsTo` is a string array that includes the value "http://www.opengis.net/spec/ogcapi-features-3/1.0/conf/filter".

A.3.2. Conformance Test 7

Test id:	/conf/filter/filter-param
Requirements:	/req/filter/filter-param, /req/filter/filter-lang-param, /req/filter/response
Test purpose:	Check that the query parameter `filter` is supported

Test method:	Given: • test "get-queryables" was successful • the list of filterable resources • the sample queryable of every filterable resource When: • a request for each resource that supports filtering is executed without a filter parameter ◦ method: `GET` ◦ path: `{apiURI}/{pathToResource}` ◦ header: `Accept: {responseMediaType}` ◦ authentication, if authentication credentials are provided Then: • assert successful execution (the status code is in the list of acceptable status codes for a successful execution, `Content-Type` header is `{responseMediaType}`); • store the result as the unfiltered result of the resource. When: • a request for each resource that supports filtering is executed with a valid filter expression ◦ method: `GET` ◦ path: `{apiURI}/{pathToResource}` ◦ query parameters (before percent encoding): `filter-lang={filter-lang}&filter={filter-valid}` where `{queryable}` in `{filter-valid}` is replaced by the sample queryable of the filterable resource ◦ header: `Accept: {responseMediaType}` ◦ authentication, if authentication credentials are provided Then: • assert successful execution (the status code is in the list of acceptable status codes for a successful execution, `Content-Type` header is `{responseMediaType}`); • assert that each returned resource matches the filter expression. When: • a request for each resource that supports filtering is executed with an invalid filter expression

A.3.3. Conformance Test 8

Test id:	/conf/filter/filter-lang-default
Requirements:	/req/filter/filter-param, /req/filter/filter-lang-param, /req/filter/response
Test purpose:	Check that the query parameter `filter-lang` default value is supported

Test method:

Given:

- test "get-queryables" was successful
- the list of filterable resources
- the queryables of every filterable resource
- the filter language `{filter-lang}` is the default filter language

When:

- a request for each resource that supports filtering is executed with a valid filter expression
  - method: `GET`
  - path: `{apiURI}/{pathToResource}`
  - query parameters (before percent encoding): `filter={filter-valid}` where `{queryable}` in `{filter-valid}` is replaced by the sample queryable of the collection
  - header: `Accept: {responseMediaType}`
  - authentication, if authentication credentials are provided

Then:

- assert successful execution (the status code is in the list of acceptable status codes for a successful execution, `Content-Type` header is `{responseMediaType}`);
- assert that each returned resource matches the filter expression.

When:

- a request for each resource that supports filtering is executed with an invalid filter expression
  - method: `GET`
  - path: `{apiURI}/{pathToResource}`
  - query parameters (before percent encoding): `filter={filter-invalid}` where `{queryable}` in `{filter-invalid}` is replaced by the sample queryable of the collection
  - header: `Accept: {responseMediaType}`
  - authentication, if authentication credentials are provided

Then:

- assert unsuccessful execution (the status code is in the list of acceptable status codes for an unsuccessful execution).

A.3.4. Conformance Test 9

Test id:	/conf/filter/expression-construction
Requirements:	/req/queryables/get-queryables-response
Test purpose:	Check that unknown queryables are rejected, if this is declared in the Queryables resource
Test method:	Given: • test "get-queryables" was successful • the list of filterable resources, reduced to those where `additionalProperties` is `false` • the sample queryable of every filterable resource in the list When: • a request for each resource is executed with a filter expression with an unsupported queryable ◦ method: `GET` ◦ path: `{apiURI}/{pathToResource}` ◦ query parameters (before percent encoding): `filter-lang={filter-lang}&filter={filter-valid}` where `{queryable}` in `{filter-valid}` is replaced by "this_is_not_a_queryable" ◦ header: `Accept: {responseMediaType}` ◦ authentication, if authentication credentials are provided Then: • assert unsuccessful execution (the status code is in the list of acceptable status codes for an unsuccessful execution).

A.3.5. Conformance Test 10

Test id:	/conf/filter/filter-crs-wgs84
Requirements:	/req/filter/filter-crs-wgs84, /req/filter/response
Test purpose:	Check that spatial predicates assume WGS84 by default

Test method: Given:

- test "get-queryables" was successful
- the list of filterable resources with a spatial queryable
- the spatial queryable of each filterable resource
- the WGS84 bbox of the resources in each filterable resource

When:

- a request for each filterable resource is executed with a filter expression with a spatial predicate
  - method: `GET`
  - path: `{apiURI}/{pathToResource}`
  - query parameters (before percent encoding): `filter-lang={filter-lang}&filter={bbox-filter}` where `{spatialQueryable}` in `{bbox-filter}` is replaced by the spatial queryable, `{x1}` is replaced by the west-bound longitude of the WGS84 bbox of the resource, `{y1}` by the south-bound latitude, `{x2}` by the east-bound longitude, and `{y2}` by the north-bound latitude
  - header: `Accept: {responseMediaType}`
  - authentication, if authentication credentials are provided

Then:

- assert successful execution (the status code is in the list of acceptable status codes for a successful execution, `Content-Type` header is `{responseMediaType}`).
- assert that result contains the same features as the unfiltered result of the filterable resource.

When:

- a request for each filterable resource with a filter expression with a spatial predicate
  - method: `GET`
  - path: `{apiURI}/{pathToResource}`
  - query parameters (before percent encoding): `filter-lang={filter-lang}&filter={bbox-filter}` where `{spatialQueryable}` in `{bbox-filter}` is replaced by the spatial queryable, `{x1}` is replaced by "1000000", `{y1}` by "1000000", `{x2}` by "2000000", and `{y2}` by "2000000"
  - header: `Accept: {responseMediaType}`
  - authentication, if authentication credentials are provided

Then:

A.3.6. Conformance Test 11

Test id:	/conf/filter/filter-crs-param
Requirements:	/req/filter/filter-crs-param, /req/filter/response
Test purpose:	Check that spatial predicates assume WGS84 by default

Test method: Given:

- test "get-queryables" was successful
- the list of filterable resources, that have a non-empty list of supported CRSs
- the spatial queryable of every collection
- the WGS84 bbox of every collection

When:

- a request for each filterable resource and every CRS for that resource with a filter expression with a spatial predicate
  - method: `GET`
  - path: `{apiURI}/{pathToResource}`
  - query parameters (before percent encoding): `filter-lang={filter-lang}&filter-crs={crsId}&filter={bbox-filter}` where `{spatialQueryable}` in `{bbox-filter}` is replaced by the spatial queryable, `{crsId}` by the URI of the CRS, `{x1}` and `{y1}` by the coordinates of the west-bound longitude and south-bound latitude of the WGS84 bbox of the collection transformed to the CRS, and `{x2}` and `{y2}` by the coordinates of the east-bound longitude and north-bound latitude of the WGS84 bbox of the collection transformed to the CRS
  - header: `Accept: {responseMediaType}`
  - authentication, if authentication credentials are provided

Then:

- assert successful execution (the status code is in the list of acceptable status codes for a successful execution, `Content-Type` header is `{responseMediaType}`);
- assert that result contains the same features as the unfiltered result of the collection.

When:

- a request for each filterable resource with a filter expression with a spatial predicate
  - method: `GET`
  - path: `{apiURI}/{pathToResource}`
  - query parameters (before percent encoding): `filter-lang={filter-lang}&filter-crs={crsId}&filter={bbox-filter}` where `{spatialQueryable}` in `{bbox-filter}` is replaced by the spatial queryable, `{crsId}` by "http://www.opengis.net/def/crs/OGC/0/does_not_exist", `{x1}` is replaced by the west-bound longitude of the WGS84 bbox of the collection, `{y1}`

A.3.7. Conformance Test 12

Test id:	/conf/filter/get-functions
Requirements:	/req/filter/get-functions-operation; /req/filter/get-functions-response-json
Test purpose:	Check that the Functions resource exists and is schema valid
Test method:	Given: • test "get-conformance" was successful • the API supports custom functions in filter expressions When: • the request for the Functions page is executed ◦ method: `GET` ◦ path: `{apiURI}/functions` ◦ header: `Accept: application/json` ◦ authentication, if authentication credentials are provided Then: • assert successful execution (status code is "200", `Content-Type` header is "application/json"); • assert that the response is valid against the OpenAPI 3.0 schema identified in the requirement.

NOTE Requirement `/req/filter/mixing-expression` can only be checked in dependent conformance classes, when additional filtering parameters are known.

A.4. Conformance Class "Features Filter"

Conformance Class	
http://www.opengis.net/spec/ogcapi-features-3/1.0/conf/features-filter	
Target type	Web API
Requirements class	Requirements Class "Features Filter"
Dependency	Conformance Class "Filter"
Dependency	OGC API - Features - Part 1: Core, Conformance Class "GeoJSON"

NOTE The Conformance Class "GeoJSON" has a dependency to the Conformance Class "Core"; that is, testing against "GeoJSON" will automatically test against the dependencies.

The following table lists input given to all tests in this conformance class:

- The landing page URI of the OGC Web API (`{apiURI}`);
- Authentication credentials (optional);
- The name of the filter language to test (`{filter-lang}`; default: "cql2-text");
- A flag that indicates whether the filter language is the default filter language;
- A valid filter expression in the filter language for a queryable `{queryable}` (`{filter-valid}`; default: `{queryable} IS NULL`);
- An invalid filter expression in the filter language (`{filter-invalid}`; default: `THIS IS NOT A FILTER`);
- A valid bbox filter expression in the filter language for a spatial queryable `{spatialQueryable}` and two longitude/latitude positions in WGS 84 `{x1}`/`{y1}` and `{x2}`/`{y2}` (`{bbox-filter}`; default: `S_INTERSECTS({spatialQueryable},ENVELOPE({x1},{y1},{x2},{y2})`);
- A flag that indicates whether the API supports custom functions in filter expressions.

A.4.1. Conformance Test 13

Test id:	/conf/features-filter/get-conformance
Requirements:	n/a
Test purpose:	Check that the API declares support for the conformance class
Test method:	Given: • n/a When: • the request for the Conformance Declaration is executed ◦ method: `GET` ◦ path: `{apiURI}/conformance` ◦ header: `Accept: application/json` ◦ authentication, if authentication credentials are provided Then: • assert successful execution (status code is "200", `Content-Type` header is "application/json"); • assert that `$.conformsTo` is a string array that includes the value "http://www.opengis.net/spec/ogcapi-features-3/1.0/conf/features-filter".

A.4.2. Conformance Test 14

Test id:	/conf/features-filter/get-collections

Requirements:	n/a
Test purpose:	Retrieve the list of collections provided by the API
Test method:	Given: • test "get-conformance" was successful When: • the request for the Collections page is executed ◦ method: `GET` ◦ path: `{apiURI}/collections` ◦ header: `Accept: application/json` ◦ authentication, if authentication credentials are provided Then: • assert successful execution (status code is "200", `Content-Type` header is "application/json"); • assert that `$.collections` is an array; • store the array as the list of collections.

A.4.3. Conformance Test 15

Test id:	/conf/features-filter/get-collection
Requirements:	/req/features-filter/queryables-link
Test purpose:	Check that a link to the Queryables resource exists for every collection

Test method:	Given: • test "get-collections" was successful • the list of collections When: • the request for the Collection page is executed for every collection in the list (`collectionId`: JSONPath `$.collections[*].id`) ◦ method: `GET` ◦ path: `{apiURI}/collections/{collectionId}` ◦ header: `Accept: application/json` ◦ authentication, if authentication credentials are provided Then: • assert successful execution (status code is "200", `Content-Type` header is "application/json"); • assert that a non-negative integer `n` exists where `$.links[{n}].rel` is "http://www.opengis.net/def/rel/ogc/1.0/queryables" and where `$.links[{n}].href` is (after normalization) the URI `{apiURI}/collections/{collectionId}/queryables`; • store `$.extent.spatial.bbox[0]` as the WGS84 bbox of the collection • store `$.crs[*]` as the list of CRS supported for the collection

A.4.4. Conformance Test 16

Test id:	/conf/features-filter/filter-on-items
Requirements:	/req/features-filter/filter-param, /req/features-filter/filter-lang-param, /req/features-filter/filter-crs-param, /req/features-filter/response
Test purpose:	Check that the API supports filters on the Features resource.

Test method:	Given: • test "get-conformance" was successful When: • the tests for the "Filter" conformance class are executed with the following input parameters: ◦ All input parameters given to this conformance class (see above); ◦ The path template to the resource that supports filtering is `{apiURI}/collections/{collectionId}/items` with a response media type `application/geo+json` (GeoJSON); ◦ The acceptable status codes for a successful filter execution are: "200"; ◦ The list of acceptable status codes for an unsuccessful filter execution are: "400". Then: • the Web API under test passes the tests.

A.4.5. Conformance Test 17

Test id:	/conf/features-filter/mixing-expression
Requirements:	/req/filter/mixing-expressions
Test purpose:	Check that a filter and a bbox parameter are evaluated with an AND

Test method: Given:

- test "get-queryables" was successful
- the list of collections
- the spatial queryable of every collection
- the WGS84 bbox of every collection

When:

- a request for the resource that supports filtering is executed for every collection in the list (`collectionId`: JSONPath `$.collections[*].id`) with a filter expression with a spatial predicate, where the collection has a WGS84 bbox and a spatial queryable
  - method: `GET`
  - path: `{pathToResource}`
  - query parameters (before percent encoding): `filter-lang={filter-lang}&filter={bbox-filter}&bbox={x1},{y1},{x2},{y2}` where `{spatialQueryable}` in `{bbox-filter}` is replaced by the spatial queryable, `{x1}` is replaced by the west-bound longitude of the WGS84 bbox of the collection, `{y1}` by the south-bound latitude, `{x2}` by the east-bound longitude, and `{y2}` by the north-bound latitude
  - header: `Accept: {responseMediaType}`
  - authentication, if authentication credentials are provided

Then:

- assert successful execution (the status code is in the list of acceptable status codes for a successful execution, `Content-Type` header is `{responseMediaType}`).
- assert that result contains the same features as the unfiltered result of the collection.

When:

- a request for the resource that supports filtering is executed for every collection in the list (`collectionId`: JSONPath `$.collections[*].id`) with a filter expression with a spatial predicate, where the collection has a WGS84 bbox and a spatial queryable
  - method: `GET`
  - path: `{pathToResource}`
  - query parameters (before percent encoding): `filter-lang={filter-lang}&filter={bbox-filter}&bbox={x3},{y3},{x4},{y4}` where `{spatialQueryable}` in `{bbox-filter}` is replaced by the spatial queryable, `{x1}` is replaced by the west-bound longitude of the WGS84

Annex B: Revision History

Date	Release	Editor	Primary clauses modified	Description
2020-04-06	1.0.0-SNAPSHOT	P. Vretanos	all	initial version
2020-12-22	1.0.0-draft.1	P. Vretanos, C. Portele	all	version for OAB review
2021-01-18	1.0.0-SNAPSHOT	P. Vretanos, C. Portele, C. Reed	all	include changes from OAB review
2021-01-27	1.0.0-draft.2	C. Portele	5.2, 10.2, 10.3	include changes from OGC-NA review, version for public review
2021-09-27	1.0.0-SNAPSHOT	C. Portele	all	move CQL2 to its own standard
2024-03-07	1.0.0-rc.1	C. Portele, P. Vretanos	all	release candidate, submission for the OGC approval process
2024-03-26	1.0.0-rc.2	C. Portele, P. Vretanos	all	release candidate • #912 Changed requirement for a Link HTTP header to a recommendation.

Annex C: Bibliography

- Internet Engineering Task Force (IETF). RFC 5234: **Augmented BNF for Syntax Specifications: ABNF** [online]. Edited by D. Crocker, P. Overell. 2008 [viewed 2020-11-22]. Available at https://www.rfc-editor.org/rfc/rfc5234.html
- Internet Engineering Task Force (IETF). draft-handrews-json-schema-validation-02: **JSON Schema Validation: A Vocabulary for Structural Validation of JSON** [online]. Edited by A. Wright, H. Andrews, B. Hutton. 2019 [viewed 2020-11-22]. Available at https://datatracker.ietf.org/doc/html/draft-handrews-json-schema-02
- Internet Assigned Numbers Authority (IANA). **Link Relation Types** [online, viewed 2020-03-16], Available at https://www.iana.org/assignments/link-relations/link-relations.xml
- Open Geospatial Consortium (OGC). **OGC Link Relation Type Register** [online, viewed 2021-06-10], Available at http://www.opengis.net/def/rel
- Open Geospatial Consortium (OGC). **OGC CURIE Register** [online, viewed 2023-04-24], Available at http://www.opengis.net/def/curie
- OpenAPI Initiative (OAI). **OpenAPI Specification 3.0** [online]. 2020 [viewed 2020-03-16]. The latest patch version at the time of publication of this standard was 3.0.3, available at http://spec.openapis.org/oas/v3.0.3
- Internet Engineering Task Force (IETF). RFC 3986: **Uniform Resource Identifier (URI): Generic Syntax** [online]. Edited by T. Berners-Lee, R. Fielding, L. Masinter. [viewed 2020-03-16]. Available at https://www.rfc-editor.org/rfc/rfc3986.html
- ISO 8601-1:2019, **Date and time — Representations for information interchange — Part 1: Basic rules**
- ISO 8601-2:2019, **Date and time — Representations for information interchange — Part 2: Extensions**
- ISO 15836-2:2019, **Information and documentation — The Dublin Core metadata element set — Part 2: DCMI Properties and classes**
- Open Geospatial Consortium (OGC) / World Wide Web Consortium (W3C): **Spatial Data on the Web Best Practices** [online]. Edited by J. Tandy, L. van den Brink, P. Barnaghi. 2017 [viewed 2020-03-16]. Available at https://www.w3.org/TR/sdw-bp/