OGC API - Features - Part 5

Search (PROPOSAL DRAFT)

Open Geospatial Consortium

Submission Date: <yyyy-mm-dd>

Approval Date: <yyyy-mm-dd>

Publication Date: <yyyy-mm-dd>

External identifier of this OGC® document: http://www.opengis.net/doc/IS/ogcapi-features-4/1.0

Internal reference number of this OGC® document: 20-096

Version: 1.0.0-SNAPSHOT (Editor's draft)

Latest Published Draft: n/a

Category: OGC® PROPOSAL DRAFT

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

OGC API - Features - Part 5: Search (PROPOSAL DRAFT)

Warning

This document is not an OGC Standard. This document is distributed for review and comment. This document is subject to change without notice and may not be referred to as an OGC Standard.

Recipients of this document are invited to submit, with their comments, notification of any relevant patent rights of which they are aware and to provide supporting documentation.

Document type: OGC® Standard (PROPOSAL DRAFT)
Document subtype: Interface
Document stage: Draft
Document language: English

License Agreement

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

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

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

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

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

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

Table of Contents

i. Abstract

NOTE	This is a PROPOSAL DRAFT for a search extension to OGC API Features. As such it is INCOMPLETE.

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.

This extension defines the behaviour of an API that supports search capabilities beyond those defined in Part 1 and which may include query expressions that are not conveniently encoded as query parameters on a URL as defined in Part 3.

NOTE	This specification is being developed in the OGC API - Features SWG but is being written as a generic extension that is applicable to a variety of resource types including features. The feature-specific portions of this extension are issolated to the clause titled Features. It is anticipated that the bulk of this extension will eventually be moved into 'OGC API - Common' and only the feature-specific content will remain to be managed by the 'OGC API - Features' SWG.

CAUTION	This is a DRAFT version of the 5th part of the OGC API - Features standards. This draft is not complete and there are open issues that are still under discussion.

ii. Keywords

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

resource feature collection instance spatial data openapi query stored REST PUT POST DELETE join filter CQL

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.

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.

Chapter 1. Scope

This document specifies an extension that defines the behaviour of a server that supports searching for resources from one or more collections. This part supports queries that cannot be expressed, or cannot be conveniently expressed, using the filtering mechanisms available in Parts 1 or 3.

Examples of the types of queries that can be expressed using Part 5 are:

- queries with a long expression text that cannot be conveniently specified as URL parameters
- bundled queries that, in a single request, fetch resources from two or more collectons
- queries that include predicates that join two or more collections
- stored queries possibly referencing multiple resource collections
- stored queries with parameters

Specifically, this document defines:

- an endpoint, `/collections/{collectionId}/search` that can be used to execute ad-hoc queries on a single collection or discover the stored queries available for this collection
- an endpoint, `/search` that can used to execute ad-hoc queries on multiple collections or discover the available multi-collection stored queries,
- an endpoint `/collections/{collectionId}/search/{queryId}` that can used to execute, create, modify or delete stored queries for this collection,
- an endpoint, `/search/{queryId}` that can be used to execute, create, modify or delete multi-collection stored queries,
- support for parameterized stored queries,
- a query expression language, based on CQL,
- support for standing or periodically executed stored queries

The following table crosswalks each of the resource endpoints discussed in this standard with the HTTP methods GET, POST, PUT and DELETE. Each intersecting cell in the table either contains a reference to the section in this standard where that combination from resource and method is discussed or the phrase `NOT DEFINED` which is used to indicate that this specification does not describe any behaviour for that combination of resource endpoint and HTTP method.

Table 1. Supported HTTP methods by resource

Resource endpoint	HTTP METHOD	Description	Reference
/collections/{collectionId}/search	GET	Get the list of stored queries for this collection.	Discovering stored queries
	POST	Execute an ad-hoc search on this collection.	Executing ad-hoc queries
	PUT	NOT DEFINED	
	DELETE	NOT DEFINED	
/search	GET	Get the list of multi-collection stored queries.	Discovering multi-collection stored queries
	POST	Execute an ad-hoc search that references multiple collections.	Executing ad-hoc multi-collection queries
	PUT	NOT DEFINED	
	DELETE	NOT DEFINED	
/collections/{collectionId}/search/{queryId}	GET	Execute this stored query.	Executing stored queries
	POST	Execute this stored query with an application/x-www-form-urlencoded body.	Executing stored queries
	PUT	Create or replace a stored query for this collection with this identifier.	Creating stored queries, Updating stored queries
	DELETE	Delete this stored query.	Deleting stored queries

Resource endpoint	HTTP METHOD	Description	Reference
/search/{queryId}	GET	Execute this multi-collection stored query.	Executing multi-collection stored queries
	POST	Execute this multi-collection stored query with an application/x-www-form-urlencoded body.	Executing multi-collection stored queries
	PUT	Create or replace a multi-collection stored query.	Creating multi-collection stored queries, Updating multi-collection stored queries
	DELETE	Delete this stored query.	Deleting multi-collection stored queries
/collections/{collectionId}/search/{queryId}/parameters	GET	Get the list of parameters for this stored query.	Retirieving the list of stored query parameters
	POST	Define the parameters for a stored query.	Defining query parameter
	PUT	Update a stored query parameter.	Updating a query parameter
	DELETE	Delete a stored query parameter.	Delete a query parameter
/search/{queryId}/parameters	GET	Get the list of parameters for this parametertized multi-collection stored query.	Retrieving the list of multi-collection-stored-query-parameters
	POST	Define the parameters for a multi-collection stored query.	Defining query parameter
	PUT	Update a multi-collection stored query parameter.	Updating a query parameter
	DELETE	Delete a multi-collection stored query parameter.	Delete a query parameter

Chapter 2. Conformance

This standard defines three requirements / conformance classes:

- Search
- Multi-Collection Search
- Stored Query
- Multi-Collection Stored Query
- Parameterized Stored Query
- Parameterized Multi-Collection Stored Query
- Parameterized Query Expressions
- OGC JSON Encoding for Query Expressions
- Standing Query
- HTML
- GeoJSON
- Geography Markup Language (GML), Simple Features Profile, Level 0
- Geography Markup Language (GML), Simple Features Profile, Level 2

The standardization target is "Web APIs".

The URIs of the associated conformance classes are:

Table 2. Conformance class URIs

Conformance class	URI
Search	http://www.opengis.net/spec/ogcapi-features-5/1.0/req/search
Multi-Collection Search	http://www.opengis.net/spec/ogcapi-features-5/1.0/req/multi-collection-search
Stored Query	http://www.opengis.net/spec/ogcapi-features-5/1.0/req/stored-query
Multi-Collection Stored Query	http://www.opengis.net/spec/ogcapi-features-5/1.0/req/multi-collection-stored-query
Parameterized Stored Query	http://www.opengis.net/spec/ogcapi-features-5/1.0/req/parameterized-stored-query
Parameterized Multi-Collection Stored Query	http://www.opengis.net/spec/ogcapi-features-5/1.0/req/parameterized-multi-collection-stored-query
Parameterized Query Expressions	http://www.opengis.net/spec/ogcapi-features-5/1.0/req/parametrized-query-expression
OGC JSON Encoding for Query Expressions	http://www.opengis.net/spec/ogcapi-features-5/1.0/req/ogc-json-query-expression
Standing Query	http://www.opengis.net/spec/ogcapi-features-5/1.0/req/standing-query

Conformance class	URI
HTML	http://www.opengis.net/spec/ogcapi-features-5/1.0/req/html
GeoJSON	http://www.opengis.net/spec/ogcapi-features-5/1.0/req/geojson
GML Simple Features Profile, Level 0	http://www.opengis.net/spec/ogcapi-features-5/1.0/req/gmlsf0
GML Simple Features Profile, Level 2	http://www.opengis.net/spec/ogcapi-features-5/1.0/req/gmlsf2

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. == 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.

- Portele, C., Vretanos, P., Heazel, C.: OGC 17-069r2, **OGC API - Features - Part 1: Core**, http://example.com/fixme
- Urpalainen, J.: IETF RFC 5261, **An Extensible Markup Language (XML) Patch Operations Framework Utilizing XML Path Language (XPath) Selectors**, 2008 http://tools.ietf.org/rfc/rfc5261.txt
- Dusseault, L., Snell, J.: IETF RFC 5789, **PATCH Method for HTTP**, 2010 http://tools.ietf.org/rfc/rfc5789.txt
- Bryan, P., Nottingham, M.: IETF RFC 6902, **JavaScript Object Notation (JSON) Patch**, 2013 http://tools.ietf.org/rfc/rfc6902.txt
- Hoffman, P., Snell, J.: IETF RFC 7396, **JSON Merge Path**, 2015 http://tools.ietf.org/rfc/rfc7396.txt == Terms, definitions and abbreviated terms

2.1. Terms and Definitions

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.

For the purposes of this document, the following additional terms, definitions and abbreviated terms apply in addition to those defined in OGC API - Features - Part 1: Core.

2.1.1. collection items end point

the path of the end point from which the resources of items of a collection can be accesses

EXAMPLE: For features, the collection items end point is '/collections/{collectionId}/items'.

EXAMPLE: For processes, the collection items end point is '/processes'

2.1.2. resource end point

the path of the end point used to access a specific instance of a resource from a collection

EXAMPLE: For features, the resource end point is '/collections/{collectionId}/items/{featureId}'.

EXAMPLE: For processes, the resource end point is '/processes/{processId}'.

Chapter 3. Conventions and background

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

Chapter 4. Requirements Class "Search"

4.1. Overview

Requirements Class	
http://www.opengis.net/spec/ogcapi-features-5/1.0/req/search	
Target type	Web API
Dependency	RFC 2616 (HTTP/1.1)

4.2. Executing ad-hoc queries

Requirement 1	**/req/search/post-op**
A	For every resource collection identified in the resource collections response (path `/collections`), the server SHALL support the HTTP POST operation at the path `/collections/{collectionId}/search`.
B	The parameter `collectionId` is each `id` property in the resource collections response (JSONPath: `$.collections[*].id`).

Requirement 2	**/req/search/limit-definition**

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

Requirement 3	**/req/search/post-body**
A	The body of the HTTP POST request SHALL contain a representation of a query.

This specification does not mandate a specific query expression language.

Recommendation 1	**/rec/ogc-json-query-expression**
A	If a query expression can be represented as JSON for its intended use, then implemenations SHOULD considing supporting the "OGC JSON Encoding for Query Expressions" as defined by the OpenAPI 3.0 schema query.yaml.

Schema for query expressions

```
type: array
items:
   type: object
   properties:
     collections:
       type: array
       items:
          type: string
     properties:
       type: array
       items:
          type: string
     filter:
        $ref: cql.yaml
```

Requirement 4	**/req/search/response**
A	A successful execution of the operation SHALL be reported as a response with a HTTP status code `200`.
B	The response SHALL only include resources selected by the request.

Requirement 5	**/req/search/limit-response**
A	The response SHALL not contain more resource than specified by the optional `limit` parameter. If the API definition specifies a maximum value for `limit` parameter, the response SHALL not contain more resource than this maximum value.
B	Only items are counted that are on the first level of the collection. Any nested objects contained within the explicitly requested items SHALL not be counted.

Chapter 5. Requirements Class "Multi-Collection Search"

5.1. Overview

Requirements Class	
http://www.opengis.net/spec/ogcapi-features-5/1.0/req/multi-collection-search	
Target type	Web API
Dependency	RFC 2616 (HTTP/1.1)

5.2. Executing ad-hoc multi-collection queries

Requirement 6	**/req/multi-collection-search/post-op**
A	The server SHALL support the HTTP POST operation at the path `/search`.

Requirement 7	**/req/multi-collection-search/limit-definition**

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

Requirement 8	**/req/multi-collection-search/post-body**
A	The body of the HTTP POST request SHALL contain a representation of a query.

This specification does not mandate a specific query expression language.

This specification does, however, recommend: [rec_ogc-json-query-expression].

Requirement 9	**/req/multi-collection-search/response**
A	A successful execution of the operation SHALL be reported as a response with a HTTP status code `200`.
B	The response SHALL only include resources selected by the request.

Requirement 10	**/req/multi-collection-search/limit-response**
A	The response SHALL not contain more resource than specified by the optional `limit` parameter. If the API definition specifies a maximum value for `limit` parameter, the response SHALL not contain more resource than this maximum value.
B	Only items are counted that are on the first level of the collection. Any nested objects contained within the explicitly requested items SHALL not be counted.

Chapter 6. Requirements Class "Stored Query"

6.1. Overview

Requirements Class	
http://www.opengis.net/spec/ogcapi-features-5/1.0/req/stored-query	
Target type	Web API

6.2. Discovering stored queries

Requirement 11	/req/stored-query/queries-op
A	For every resource collection identified in the resource collections response (path `/collections`), the server SHALL support the HTTP GET operation at the path `/collections/{collectionId}/search`.
B	The parameter `collectionId` is each `id` property in the resource collections response (JSONPath: `$.collections[*].id`).

Requirement 12	/req/stored-query/queries-success
A	A successful execution of the operation SHALL be reported as a response wit h a HTTP status code `200`.
B	The content of that response SHALL be based upon the OpenAPI 3.0 schema queries.yaml.

Schema for the list of stored queries (queries.yaml)

```
type: object
required:
  - queries
properties:
  queries:
    type: array
    items:
      $ref: query-md.yaml
  links:
    type: array
    items:
      $ref: link.yaml
```

Schema describing a stored query (query-md.yaml)

```
type: object
required:
  - id
  - mutable
properties:
  id:
    description: identifier of the stored query
    type: string
  mutable:
    description: indicates whether this query can be modified through the API
    type: boolean
    default: true
  title:
    description: human readable title for the stored query
    type: string
  description:
    description: a description of the stored query
    type: string
  collections:
    description: the list of collections, specified by id, that this query accesses
    type: array
    items:
      type: string
  query:
    description: the text of the stored query
    oneOf:
    - type: string
    - type: object
  links:
    type: array
    items:
      $ref: link.yaml
```

An implementation may choose to deploy a pre-installed set of stored queries that cannot be modified (i.e. cannot be updated or deleted). Such stored queries are called `immutable` stored queries.

6.3. Executing stored queries

Requirement 13	**/req/stored-query/get-op**
A	For every stored query identified in the stored queries response (path `/collections/{collectionId}/search`), the server SHALL support the HTTP GET operation at the path `/collection/{collectionId}/search/{queryId}`.

B	The parameter `queryId` is each `id` property in the stored queries response (JSONPath: `$.queries[*].id`).
C	The parameter `collectionId` is each `id` property in the resource collections response (JSONPath: `$.collections[*].id`).

Requirement 14	**/req/stored-query/post-op**
A	For every stored query identified in the stored queries response (path `/collections/{collectionId}/search`), the server SHALL support the HTTP POST operation at the path `/collection/{collectionId}/search/{queryId}`.
B	The parameter `queryId` is each `id` property in the stored queries response (JSONPath: `$.queries[*].id`).
C	The parameter `collectionId` is each `id` property in the resource collections response (JSONPath: `$.collections[*].id`).
D	The content type of the POST body shall be `application/x-www-form-urlencoded`.

Requirement 15	**/req/stored-query/limit-definition**

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

Requirement 16	**/req/stored-query/get-success**

A	A successful execution of the operation SHALL be reported as a response with a HTTP status code `200`.
B	The response SHALL only include resources selected by the request.

Requirement 17	**/req/stored-query/limit-response**
A	The response SHALL not contain more resource than specified by the optional `limit` parameter. If the API definition specifies a maximum value for `limit` parameter, the response SHALL not contain more resource than this maximum value.
B	Only items are counted that are on the first level of the collection. Any nested objects contained within the explicitly requested items SHALL not be counted.

6.4. Creating stored queries

Requirement 18	**/req/stored-query/put-create**
A	The server SHALL support the HTTP PUT operation at the path `/collections/{collectionId}/search/{queryId}`.
B	The parameter `queryId` SHALL be specified by the client.
C	The parameter `collectionId` is each `id` property in the resource collections response (JSONPath: `$.collections[*].id`).

Requirement 19	**/req/stored-query/put-body**
A	The body of a HTTP PUT request SHALL contain a representation of the query.

This specification does not mandate a specific query expression language.

This specification does, however, recommend: [rec_ogc-json-query-expression].

Requirement 20	**/req/stored-query/put-create-success**
A	A successful execution of the operation SHALL be reported as a response with a HTTP status code '201'.

6.5. Updating stored queries

Requirement 21	**/req/stored-query/put-update**
Condition	The value of the `mutable` parameter (JSONPath: `$.queries[*].mutable`) SHALL be `true`.
A	The server SHALL support the HTTP PUT operation at the path `/collections/{collectionId}/search/{queryId}`.
B	The parameter `queryId` is each `id` property in the stored queries response (JSONPath: `$.queries[*].id`).
C	The parameter `collectionId` is each `id` property in the resource collections response (JSONPath: `$.collections[*].id`).

See [req_stored-query_put-body].

Requirement 22	**/req/stored-query/put-update-success**
A	A successful execution of the operation SHALL be reported as a response with a HTTP status code '204'.

6.6. Deleting stored queries

Requirement 23	**/req/stored-query/delete-op**
Condition	The value of the `mutable` property (JSONPath: `$.queries[*].mutable`) SHALL be `true`.
A	For every stored query in the stored queries response (path '/collections/{collectionId}/search'), the server SHALL support the HTTP DELETE operation at the path '/collections/{collectionId}/search/{queryId}'.
B	The parameter `queryId` is each `id` property in the stored queries response (JSONPath: `$.queries[*].id`).
C	The parameter `collectionId` is each `id` property in the resource collections response (JSONPath: `$.collections[*].id`).

Requirement 24	**/req/stored-query/delete-success**

A	A successful execution of the operation SHALL be reported as a response wit h a HTTP status code '200'.

Chapter 7. Requirements Class "Multi-Collection Stored Query"

7.1. Overview

Requirements Class	
http://www.opengis.net/spec/ogcapi-features-5/1.0/req/multi-collection-stored-query	
Target type	Web API

7.2. Discovering multi-collection stored queries

Requirement 25	**/req/multi-collection-stored-query/queries-op**
A	The server SHALL support the HTTP GET operation at the path `/search`.

Requirement 26	**/req/multi-collection-stored-query/queries-success**
A	A successful execution of the operation SHALL be reported as a response wit h a HTTP status code `200`.
B	The content of that response SHALL be based upon the OpenAPI 3.0 schema queries.yaml.
C	The parameters `collections` (JSONPAth: `$.queries[*].collections`) SHALL be required.

Schema for the list of multi-collection stored queries

```
type: object
required:
  - queries
properties:
  queries:
    type: array
    items:
      $ref: query-md.yaml
  links:
    type: array
    items:
      $ref: link.yaml
```

Schema describing a multi-collection stored query

```
type: object
required:
  - id
  - mutable
properties:
  id:
    description: identifier of the stored query
    type: string
  mutable:
    description: indicates whether this query can be modified through the API
    type: boolean
    default: true
  title:
    description: human readable title for the stored query
    type: string
  description:
    description: a description of the stored query
    type: string
  collections:
    description: the list of collections, specified by id, that this query accesses
    type: array
    items:
      type: string
  query:
    description: the text of the stored query
    oneOf:
    - type: string
    - type: object
  links:
    type: array
    items:
      $ref: link.yaml
```

7.3. Executing multi-collection stored queries

Requirement 27	**/req/multi-collection-stored-query/get-op**
A	For every stored query identified in the stored queries response (path `/search`), the server SHALL support the HTTP GET operation at the path `/search/{queryId}`.
B	The parameter `queryId` is each `id` property in the stored queries response (JSONPath: `$.queries[*].id`).

Requirement 28	/req/multi-collection-stored-query/limit-definition

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

Requirement 29	/req/multi-collection-stored-query/get-success
A	A successful execution of the operation SHALL be reported as a response with a HTTP status code `200`.
B	The response SHALL only include resources selected by the request.

Requirement 30	/req/multi-collection-stored-query/limit-response
A	The response SHALL not contain more resource than specified by the optional `limit` parameter. If the API definition specifies a maximum value for `limit` parameter, the response SHALL not contain more resource than this maximum value.
B	Only items are counted that are on the first level of the collection. Any nested objects contained within the explicitly requested items SHALL not be counted.

7.4. Creating multi-collection stored queries

Requirement 31	/req/multi-collection-stored-query/put-create
A	The server SHALL support the HTTP PUT operation at the path `/search/{queryId}`.

B	The parameter `queryId` SHALL be specified by the client.
C	The parameter `collectionId` is each `id` property in the resource collections response (JSONPath: `$.collections[*].id`).

Requirement 32	**/req/multi-collection-stored-query/put-body**
A	The body of a HTTP PUT request SHALL contain a representation of the query.

This specification does not mandate a specific query expression language.

This specification does, however, recommend: [rec_ogc-json-query-expression].

Requirement 33	**/req/multi-collection-stored-query/put-create-success**
A	A successful execution of the operation SHALL be reported as a response with a HTTP status code '201'.

7.5. Updating multi-collection stored queries

Requirement 34	**/req/multi-collection-stored-query/put-update**
Condition	The value of the `mutable` parameter (JSONPath: `$.queries[*].mutable`) SHALL be `true`.
A	The server SHALL support the HTTP PUT operation at the path `/search/{queryId}`.
B	The parameter `queryId` is each `id` property in the stored queries response (JSONPath: `$.queries[*].id`).

See: [req_multi-collection-stored-query_put-body].

Requirement 35	**/req/multi-collection-stored-query/put-update-success**
A	A successful execution of the operation SHALL be reported as a response with a HTTP status code '204'.

7.6. Deleting multi-collection stored queries

Requirement 36	**/req/multi-collection-stored-query/delete-op**
Condition	The value of the `mutable` property (JSONPath: `$.queries[*].mutable`) SHALL be `true`.
A	For every stored query in the stored queries response (path '/search'), the server SHALL support the HTTP DELETE operation at the path '/search/{queryId}'.
B	The parameter `queryId` is each `id` property in the stored queries response (JSONPath: `$.queries[*].id`).

Requirement 37	**/req/multi-collection-stored-query/delete-success**
A	A successful execution of the operation SHALL be reported as a response wit h a HTTP status code '200'.

Chapter 8. Requirements Class "Parameterized Stored Query"

8.1. Overview

Requirements Class	
http://www.opengis.net/spec/ogcapi-features-5/1.0/req/parameterized-stored-query	
Target type	Web API

NOTE	Not sure we need a separate conformance class for parameterization since stored queries without parameters seem useless to me. Probably need more developer feedback for this one.

8.2. Parameter discovery

Requirement 38	/req/parametrized-stored-query/parameters-op
A	For each stored query identified in the stored queries response (path: `/collections/{collectionId}/search`), the server SHALL support the HTTP GET operation at the path `/collection/{collectionId}/search/{queryId}/parameters`.
B	The parameter `queryId` is each `id` property (JSONPath: `$.queries[*].id`) in the stored queries response (path: `/collections/{collectionId}/search`).
C	The parameter `collectionId` is each `id` property (JSONPath: `$.collections[*].id`) in the resource collections response (path: `/collections`).

Requirement 39	/req/parameterized-stored-query/parameters-success
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 OpenAPI 3.0 schema parameters.yaml.
C	Each parameter identifier, `id`, SHALL also be valid URL query parameter name as defined in Uniform Resource Identifier (URI): Generic Syntax.

Schema for the list of parameters (parameters.yaml)

```
type: object
required:
- parameters
properties:
   parameters:
      type: array
      items:
         $ref: parameter.yaml
```

Schema for a parameter description(parameter.yaml)

```
type: object
required:
   - id
   - type
properties:
   id:
      description: the token that may be used as a query parameter
      type: string
   title:
      description: a human readable title for the queryable
      type: string
   description:
      description: a human-readable narrative describing the queryable
      type: string
   language:
      description: the language used for the title and description
      type: string
   type:
      description: the data type of the queryable
      type: string
      enum:
      - string
      - number
      - integer
      - boolean
      - spatial
      - temporal
   spatial-types:
      description: the list of allowed spatial geometry types, if known
      type: string
      enum:
      - point
      - multi-point
      - curve
      - multi-curve
      - surface
      - multi-surface
```

```
      - solid
      - multi-solid
      - aggregate
      - any
   temporal-types:
      description: the list of allowed temporal geometry types, if known
      type: string
      enum:
      - instant
      - interval
   links:
      description: additional links, e.g. to documentation or to the schema of the
values
      type: array
      items:
         $ref: 'https://raw.githubusercontent.com/opengeospatial/ogcapi-
features/master/core/openapi/ogcapi-features-1.yaml#/components/schemas/link'
```

8.3. Defining query parameters

Requirement 40	**/req/parametrized-stored-query/parameter-define**
A	For each stored query identified in the stored queries response (path: `/collections/{collectionId}/search`), the server SHALL support the HTTP POST operation at the path `/collection/{collectionId}/search/{queryId}/parameters`.
B	The parameter `queryId` is each `id` property (JSONPath: `$.queries[*].id`) in the stored queries response (path: `/collections/{collectionId}/search`).
C	The parameter `collectionId` is each `id` property (JSONPath: `$.collections[*].id`) in the resource collections response (path: `/collections`).

Requirement 41	**/req/parameterized-stored-query/parameter-define-body**
A	The body of the HTTP POST request SHALL contain a document that validates against the OpenAPI 3.0 schema document `parameters.yaml`.

Requirement 42	**/req/parameterized-stored-query/parameter-define-success**

A	A successful execution of the operation SHALL be reported as a response with a HTTP status code '201'.
B	For each defined stored query parameter, the response SHALL include a 'Location' header with the URL of the newly created parameter.

8.4. Updating a query parameter

Requirement 43	**/req/parametrized-stored-query/parameter-update**
A	For each stored query parameter identified in the parameters response (path: `/collections/{collectionId}/search/{queryId}/parameters`), the server SHALL support the HTTP PUT operation at the path `/collection/{collectionId}/search/{queryId}/parameters/{parameterId}`.
B	The parameter `parameterId` is each `id` property (JSONPath: `$.parameters[*].id`) in the parameters response (path: `/collections/{collectionId}/search/{queryid}/parameters`).
C	The parameter `queryId` is each `id` property (JSONPath: `$.queries[*].id`) in the stored queries response (path: `/collections/{collectionId}/search`).
D	The parameter `collectionId` is each `id` property (JSONPath: `$.collections[*].id`) in the resource collections response (path: `/collections`).

Requirement 44	**/req/parameterized-stored-query/parameter-update-body**
A	The body of the HTTP PUT request SHALL contain a document that validates against the OpenAPI 3.0 schema document `parameter.yaml`.

Requirement 45	**/req/parameterized-stored-query/parameter-update-success**
A	A successful execution of the operation SHALL be reported as a response with a HTTP status code '200'.

8.5. Deleting a query parameter

Requirement 46	**/req/parametrized-stored-query/parameter-delete**
A	For each stored query parameter identified in the parameters response (path: `/collections/{collectionId}/search/{queryId}/parameters`), the server SHALL support the HTTP DELETE operation at the path `/collection/{collectionId}/search/{queryId}/parameters/{parameterId}`.
B	The parameter `parameterId` is each `id` property (JSONPath: `$.parameters[*].id`) in the parameters response (path: `/collections/{collectionId}/search/{queryid}/parameters`).
C	The parameter `queryId` is each `id` property (JSONPath: `$.queries[*].id`) in the stored queries response (path: `/collections/{collectionId}/search`).
D	The parameter `collectionId` is each `id` property (JSONPath: `$.collections[*].id`) in the resource collections response (path: `/collections`).

Requirement 47	**/req/parameterized-stored-query/parameter-delete-success**
A	A successful execution of the operation SHALL be reported as a response with a HTTP status code '200'.

8.6. Executing a stored query with parameters

See Executing stored queries to understand how to execute an un-parameterized stored query.

See URL query parameters to understand how to append query parameters to the stored query path.

Example 1. Invoking a parameterized stored query.

This example invoked the parameterized stored query with identifier `MyQuery01` that takes two parameters `myParam01` and `myParam02`.

```
.../collections/MyCollection01/search/MyQuery01?myParam01=X&myParam02=Y
```

Chapter 9. Requirements Class "Parameterized Multi-Collection Stored Query"

9.1. Overview

Requirements Class	
http://www.opengis.net/spec/ogcapi-features-5/1.0/req/parameterized-multi-collection-stored-query	
Target type	Web API

NOTE Not sure we need a separate conformance class for parameterization since stored queries without parameters seem useless to me. Probably need more developer feedback for this one.

9.2. Parameter discovery

Requirement 48	**/req/parametrized-multi-collection-stored-query/parameters-op**
A	For each stored query identified in the stored queries response (path: `/search`), the server SHALL support the HTTP GET operation at the path `/search/{queryId}/parameters`.
B	The parameter `queryId` is each `id` property (JSONPath: `$.queries[*].id`) in the stored queries response (path: `/search`).
C	The parameter `collectionId` is each `id` property (JSONPath: `$.collections[*].id`) in the resource collections response (path: `/collections`).

Requirement 49	**/req/parameterized-multi-collection-stored-query/parameters-success**
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 OpenAPI 3.0 schema parameters.yaml.

C	Each parameter identifier, `id`, SHALL also be valid URL query parameter name as defined in Uniform Resource Identifier (URI): Generic Syntax.

See parameters.yaml.

See parameter.yaml.

9.3. Defining query parameters

Requirement 50	**/req/parametrized-multi-collection-stored-query/parameter-define**
A	For each stored query identified in the stored queries response (path: `/search`), the server SHALL support the HTTP POST operation at the path `/search/{queryId}/parameters`.
B	The parameter `queryId` is each `id` property (JSONPath: `$.queries[*].id`) in the stored queries response (path: `/search`).
C	The parameter `collectionId` is each `id` property (JSONPath: `$.collections[*].id`) in the resource collections response (path: `/collections`).

Requirement 51	**/req/parameterized-multi-collection-stored-query/parameter-define-body**
A	The body of the HTTP POST request SHALL contain a document that validates against the OpenAPI 3.0 schema document `parameters.yaml`.

Requirement 52	**/req/parameterized-multi-collection-stored-query/parameter-define-success**
A	A successful execution of the operation SHALL be reported as a response with a HTTP status code '201'.
B	For each defined stored query parameter, the response SHALL include a 'Location' header with the URL of the newly created parameter.

9.4. Update a query parameter

Requirement 53	/req/parametrized-multi-collection-stored-query/parameter-update
A	For each stored query parameter identified in the parameters response (path: `/search/{queryId}/parameters`), the server SHALL support the HTTP PUT operation at the path `/search/{queryId}/parameters/{parameterId}`.
B	The parameter `parameterId` is each `id` property (JSONPath: `$.parameters[*].id`) in the parameters response (path: `/search/{queryid}/parameters`).
C	The parameter `queryId` is each `id` property (JSONPath: `$.queries[*].id`) in the stored queries response (path: `/search`).
D	The parameter `collectionId` is each `id` property (JSONPath: `$.collections[*].id`) in the resource collections response (path: `/collections`).

Requirement 54	/req/parameterized-multi-collection-stored-query/parameter-update-body
A	The body of the HTTP PUT request SHALL contain a document that validates against the OpenAPI 3.0 schema document `parameter.yaml`.

Requirement 55	/req/parameterized-multi-collection-stored-query/parameter-update-success
A	A successful execution of the operation SHALL be reported as a response with a HTTP status code '200'.

9.5. Delete a query parameter

Requirement 56	/req/parametrized-multi-collection-stored-query/parameter-delete

A	For each stored query parameter identified in the parameters response (path: `/search/{queryId}/parameters`), the server SHALL support the HTTP DELETE operation at the path `/search/{queryId}/parameters/{parameterId}`.
B	The parameter `parameterId` is each `id` property (JSONPath: `$.parameters[*].id`) in the parameters response (path: `/search/{queryid}/parameters`).
C	The parameter `queryId` is each `id` property (JSONPath: `$.queries[*].id`) in the stored queries response (path: `/search`).
D	The parameter `collectionId` is each `id` property (JSONPath: `$.collections[*].id`) in the resource collections response (path: `/collections`).

Requirement 57	**/req/parameterized-multi-collection-stored-query/parameter-delete-success**
A	A successful execution of the operation SHALL be reported as a response with a HTTP status code '200'.

9.6. Executing a multi-collection stored query with parameters

See Executing stored queries to understand how to execute an un-parameterized stored query.

See URL query parameters to understand how to append query parameters to the stored query path.

Example 2. Invoking a parameterized multi-collection stored query.

This example invoked the parameterized multi-collection stored query with idenfitier `MyQuery02` that takes two parameters `myParam01` and `myParam02`.

```
.../search/MyQuery02?myParam01=X&myParam02=Y
```

Chapter 10. Requirements Class "Parameterized Query Expression"

10.1. Overview

Requirements Class	
http://www.opengis.net/spec/ogcapi-features-5/1.0/req/parameterized-query-expression	
Target type	Web API

10.2. Substitution variables

Requirement 58	/req/parametrized-query-expression/substitution-variable
A	For each parameter identifier (JSONPath: `$.parameters[*].id`) in the parameters response (path: `/collections/{collectionId}/search/{queryId}/parameters`) there SHALL be a correponding substitution variable of the form: `"$" "{" substitutionVariableName "}"` in the text of the query expression used to create the stored query.
B	The name of the substitution variable (denoted using the token `substitutionVariableName` above) SHALL be the same as the `id` of the stored query parameter (JSONPath: `$.parameters[*].id`).

NOTE Need to include substitution rules for the server.

Example 3. A parameterized query expression

In this example, the ${geometry} substitution variable is used to parameterized this stored query allowing it to be executed with different geometries each time.

```
{
   "and": [
      {
         "like": [
            {"property": "eo_instruments"},
            "OLI%"
         ]
      },
      {
         "intersects": [
            {"property": "footprint"},
            ${geometry}
         ]
      }
   ]
}
```

10.3. URL query parameters

Requirement 59	**/req/parametrized-query-expression/url-query-parameter**

A	For every stored query identified in the stored queries response (path: `/collections/{collectionId}/search`) and for each defined query parameter identified in the query parameters response (path: `/collections/{collectionId}/search/{queryId}/parameters`), a URL query parameter with the following characteristics (using an OpenAPI specification 3.0 fragment) SHALL be supported by the operation:
	`name: {parameterId}`
	`in: query`
	`required: true`
	`style: form`
	`explode: false`

B	The parameter `parameterId` is each `id` property (JSONPath: `$.parameters[*].id`) in the query parameters response (path: `/collections/{collectionId}/search/{queryId}/parameters`).

C	The parameter `queryId` is each `id` property (JSONPath: `$.queries[*].id`) in the stored queries response (path: `/collections/{collectionId}/search`).
D	The parameter `collectionId` is each `id` property (JSONPath: `$.collections[*].id`) in the resource collections response (path: `/collections`).

Example 4. Invoking a prameterized stored query

Assuming that a parameterized stored query named `FindInstrument` is created using the query expression from the example A parameterized query expression, then the following is an example invocation:

```
   /collections/landsat/search/FindInstrument?geometry=POLYGON((43.5845 -79.5442,
43.6079 -79.4893, 43.5677 -79.4632, 43.6129 -79.3925, 43.6223 -79.3238, 43.6576
-79.3163, 43.7945 -79.1178, 43.8144 -79.1542, 43.8555 -79.1714, 43.7509 -79.6390,
43.5845 -79.5442))
```

Chapter 11. Requirements Class "OGC JSON Encoding for Query Expressions"

11.1. Overview

Requirements Class	
http://www.opengis.net/spec/ogcapi-features-5/1.0/req/ogc-json-query-expression	
Target type	Web API

11.2. Query Expressions

Requirement 60	/req/ogc-json-query-expression/schema

A	The OGC JSON Encoding for Query Experssions SHALL be defined by the following OpenAPI 3.0 schema fragment:
	`type: array`
	`items:`
	`  type: object`
	`  properties:`
	`  collections:`
	`  type: array`
	`  items:`
	`  type: string`
	`  properties:`
	`  type: array`
	`  items:`
	`  type: string`
	`  filter:`
	`  $ref: cql.yaml`

Requirement 61	/req/ogc-json-query-expression/crs
Condition	The server SHALL support the CRS conformance class.

A	The OGC JSON Encoding for Query Experssions SHALL be extended to include a `filter-crs` parameter as defined in the following OpenAPI 3.0 fragment: *query.yaml* ``` type: array items: type: object properties: collections: type: array items: type: string properties: type: array items: type: string filter-crs: type: string format: uri filter: $ref: cql.yaml ```

NOTE Need to define what join output looks like ... i.e. tuples.

11.3. Examples

Example 5. Query a single collection.

This example queries a collection, radarsat2, for images that intersect a specific area and acquired from any instrument that start with "OLI".

```
CLIENT                                                                SERVER
  |                                                                     |
  |   POST /collections/radarsat2/search?limit=500   HTTP/1.1           |
  |   Host: www.someserver.com/                                         |
  |   Accept: application/geo+json                                      |
  |   Content-Type: application/ogcqry+json                             |
  |                                                                     |
  |   {                                                                 |
  |      "and": [                                                       |
  |         {                                                           |
  |            "like": [                                                |
  |               {"property": "eo_instruments"},                       |
  |               "OLI%"                                                |
  |            ]                                                        |
  |         },                                                          |
  |         {                                                           |
  |            "intersects": [                                          |
  |               {"property": "footprint"},                            |
  |               {                                                     |
  |                  "type": "Polygon",                                 |
  |                  "coordinates": [                                   |
  |                      [                                              |
  |                         [43.5845,-79.5442],                         |
  |                         [43.6079,-79.4893],                         |
  |                         [43.5677,-79.4632],                         |
  |                         [43.6129,-79.3925],                         |
  |                         [43.6223,-79.3238],                         |
  |                         [43.6576,-79.3163],                         |
  |                         [43.7945,-79.1178],                         |
  |                         [43.8144,-79.1542],                         |
  |                         [43.8555,-79.1714],                         |
  |                         [43.7509,-79.6390],                         |
  |                         [43.5845,-79.5442]                          |
  |                      ]                                              |
  |                  ]                                                  |
  |               }                                                     |
  |            ]                                                        |
  |         }                                                           |
  |      ]                                                              |
  |   }                                                                 |
  |-------------------------------------------------------------------->|
  |                                                                     |
  |   Content-Type: application/json                                    |
  |   {                                                                 |
  |      "type": "FeatureCollection"                                    |
```

```
|      "features": [                                                  |
|        {                                                            |
|           "id": "radarsat2.1010",                                   |
|           "type": "Feature",                                        |
|           "geometry": { ... },                                      |
|           "properties": { ... }                                     |
|        },                                                           |
|        .                                                            |
|        .                                                            |
|        .                                                            |
|        {                                                            |
|           "id": "radarsat2.4763",                                   |
|           "type": "Feature",                                        |
|           "geometry": { ... },                                      |
|           "properties": { ... }                                     |
|        }                                                            |
|        .                                                            |
|        .                                                            |
|        .                                                            |
|      ]                                                              |
|   }                                                                 |
|<--------------------------------------------------------------------|
```

Example 6. Example of a bundled query.

This example bundles two queries; one of "parks" and some on "lakes". The response is a GeoJSON feature collection containing the features

Example 7. Example of a spatial join query.

Find all the lakes in Algonquin Park. The query object contains a "collections" key which is the list of collections being queries. The "filter" key is simply CQL.

The response is an array of "tuples". That is, pairs (since we are querying two collections) of features that satisfy the query predicates. Since the "parks" feature in each tuple would always be the same (i.e. Algonquin Park) we use a JSON reference in every tuple after the first one to reference the Algoquin Park feature rather than duplicating it over and over.

The features themselves are encoded as GeoJSON.

```
CLIENT                                                              SERVER
  |                                                                   |
  |   POST /search   HTTP/1.1                                         |
  |   Host: www.someserver.com/                                       |
  |   Accept: application/json                                        |
  |   Content-Type: application/ogcqry+json                           |
  |                                                                   |
  |   [                                                               |
  |      {                                                            |
  |         "collections": ["parks","lakes"]                          |
  |         "filter": {                                               |
  |            "and": [                                               |
  |               {"eq": [{"property": "parks.name"},"Algonquin Park"]} |
  |               {"contains": [{"property": "parks.geometry"},       |
  |                             {"property": "lakes.geometry"}]}      |
  |            ]                                                      |
  |         }                                                         |
  |      }                                                            |
  |   ]                                                               |
  |------------------------------------------------------------------>|
  |                                                                   |
  |   Content-Type: application/json                                  |
  |   {                                                               |
  |      "tuples": [                                                  |
  |         [                                                         |
  |            {                                                      |
  |               "id": "park.001",                                   |
  |               "type": "Feature",                                  |
  |               "geometry": { ... },                                |
  |               "properties": { ... }                               |
  |            },                                                     |
  |            {                                                      |
  |               "id": "lake.001",                                   |
  |               "type": "Feature",                                  |
  |               "geometry": { ... },                                |
  |               "properties": { ... }                               |
  |            }                                                      |
  |         ],                                                        |
```

```
|         [                                                                |
|            { "$ref": "#/tuples[0]/[0]" },                                |
|            {                                                             |
|               "id": "lake.001",                                          |
|               "type": "Feature",                                         |
|               "geometry": { ... },                                       |
|               "properties": { ... }                                      |
|            }                                                             |
|         ],                                                               |
|         .                                                                |
|         .                                                                |
|         .                                                                |
|      ]                                                                   |
|   }                                                                      |
|<-------------------------------------------------------------------------|
```

Chapter 12. Requirements Class "Standing Query"

12.1. Overview

Requirements Class	
http://www.opengis.net/spec/ogcapi-features-5/1.0/req/standing-query	
Target type	Web API

T.B.D.

Standing queries are stored queries that are run perodically the results of which are sent to a response handler (e.g. email, webhook, etc.).

Example 8. A standing query example.

This is an example of a standing query invocation. In this case the parameters `responseHandler` and `period` are requirements of the standing query. The parameter `since` is a parameter os the `NewBuildings` stored query.

```
.../collections/Buildings/search/NewBuildings?responseHandler=someone@somemail.com
&period=P1W&since=2020-01-01
```

Chapter 13. OpenAPI 3.0

See OGC API - Features - Part 1: Core, Clause 9.

Chapter 14. Media Types

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

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

- application/ogc-query+json
- application/parameters+json

Chapter 15. Security Considerations

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

Annex A: Abstract Test Suite (Normative)

A.1. Introduction

NOTE	This needs to be updated.

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

Annex B: Revision History

Date	Release	Editor	Primary clauses modified	Description
2019-xx-xx	1.0.0-SNAPSHOT	J. Doe	all	initial version

Annex C: Bibliography

- Heazel, Ch.: **Guide to OGC API - Features**, https://example.org/fixme
- Open API Initiative: **OpenAPI Specification 3.0.2**, https://github.com/OAI/OpenAPI-Specification/blob/master/versions/3.0.2.md