Common Query Language (CQL2)

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/cql2/1.0

Internal reference number of this OGC® document: 21-065r2

Version: 1.0.0

Category: OGC Standard

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

Common Query Language (CQL2)

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

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. This document specifies

- A filter grammar called Common Query Language (CQL2);
- Two encodings for CQL2 - a text and a JSON encoding.

The Common Query Language (CQL2) defined in this document is a generic filter grammar that can be used to specify how resource instances in a source collection of any item type, including features, can be filtered to identify a results set. Typically, CQL2 is used in query operations to identify the subset of resources, such as features, that should be included in a response document. However, CQL2 can also be used in other operations, such as updates, to identify the subset of resources that should be affected by an operation.

Each resource instance in the source collection is evaluated against a filtering expression. The filter expression always evaluates to `true`, `false` or `null`. If the expression evaluates to `true`, the resource instance satisfies the expression and is marked as being in the result set. If the overall filter expression evaluates to `false` or `null`, the data instance is not in the result set. Thus, the net effect of evaluating a filter expression is a set of resources that satisfy the predicates in the expression.

The Common Query Language and its text encoding are not new, but this is the first time that the language is formally specified. The Common Query Language with the acronym CQL was originally created as a text encoding for use with implementations of the OGC Catalogue Service Implementation Specification. The language is based on the capabilities in the OGC Filter Encoding Standard, which was originally part of the Web Feature Service (WFS) Standard.

The Common Query Language as specified in this document is a revision of this earlier version. While the language design including the classification of operators are consistent with the earlier specification, there have been a number of changes and existing implementations of CQL will need to be updated to process filter expressions specified by this document. This document therefore uses the acronym CQL2 to refer to the current version of the Common Query Language.

NOTE	The use of CQL2 also distinguishes the Common Query Language from other existing uses of CQL for query languages, for example, for the Cassandra Query Language.

ii. Keywords

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

OGC, common query language, filter, expression, query, SQL, CQL2, where clause, selection clause

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 the Common Query Language (CQL2) to express filter expressions on spatial and temporal data.

This document defines

- A text encoding for a CQL2 filter;
- A JSON encoding for a CQL2 filter.

NOTE

The CQL2 grammar contains all the necessary language elements of a general purpose expression language, including support for spatio-temporal values. The focus of this version of the language are boolean-valued filter expressions. It is planned to potentially remove this restriction, while maintaining backward compatibility, allowing expressions that result in values of other data types, including geometries. Such expressions could be used, for example, in a styling language to specify parameter values, or to define derived properties. In this case, a particular use of CQL2 could specify the boolean-valued restriction separately, for example when used as a filter predicate expression, or as a styling rule selector. The usability of such an expression language, especially in the context of geometry types, would also greatly benefit from standardizing additional functions and/or operators e.g., to compute the geometry resulting from buffering or intersecting operations. Example files and associated schemas have been published on the OGC Schemas repository.

Chapter 2. Conformance

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

- Servers that evaluate filter expressions
  - Basic CQL2
  - Advanced Comparison Operators
  - Case-insensitive Comparisons
  - Accent-insensitive Comparisons
  - Basic Spatial Functions
  - Basic Spatial Functions with additional Spatial Literals
  - Spatial Functions
  - Temporal Functions
  - Array Functions
  - Property-Property Comparisons
  - Functions
  - Arithmetic Expressions
  - CQL2 Text encoding
  - CQL2 JSON encoding

NOTE	"Server" is used in this Standard to identify any executable software that is able to evaluate filter expressions on data.

The Basic CQL2 requirements class defines the minimal subset of the Common Query Language (CQL2) that all implementations must support. Basic CQL2 is intended to be a useful, but limited set of predicates that support fine-grained read-access to collections of resources.

The specific set of operators defined in this requirements class is:

- Logical operators:
  - and
  - or
  - not
- Comparison operators:
  - equal to
  - not equal to
  - less than
  - less than or equal to
  - greater than

    - greater than or equal to
    - is null

Basic CQL2 only requires support for property-literal comparisons. That means that servers only have to support property references as expressions on the left-hand side of an operator, and only literal values as expressions on the right-hand side.

An encoding of CQL2 may be used as the value of the filter parameters defined in the "Filter" requirements class specified in OGC API - Features - Part 3: Filtering.

The Advanced Comparison Operators requirements class specifies additional comparison operators:

- like
- between
- in

The Case-insensitive Comparison requirements class adds a standardized string function to support case-insensitive string comparisons:

- casei

The Accent-insensitive Comparison requirements class adds a standardized string function to support accent-insensitive string comparisons:

- accenti

The Basic Spatial Functions requirements class specifies minimal requirements for servers that support standardized spatial comparison functions. Only the following spatial comparison function must be supported, and only for points and bounding boxes:

- s_intersects

The Basic Spatial Functions with additional Spatial Literals requirements class is similar to the Basic Spatial Functions requirements class except it removes the restrictions on the spatial literals that may be used in the expression.

The Spatial Functions requirements class specifies requirements for servers that support a richer set of spatial comparison functions. The list of additional spatial comparison functions that must be supported is:

- s_contains
- s_crosses
- s_disjoint
- s_equals
- s_overlaps
- s_touches

- s_within

All spatial data types from the Simple Feature Access Standard must be supported.

The Temporal Functions requirements class specifies requirements for servers that support standardized temporal comparison function. The list of temporal comparison function that must be supported is:

- t_after
- t_before
- t_contains
- t_disjoint
- t_during
- t_equals
- t_finishedby
- t_finishes
- t_intersects
- t_meets
- t_metby
- t_overlappedby
- t_overlaps
- t_startedby
- t_starts

The Array Functions requirements class specifies requirements for standardized array comparison functions for sets of values. The array comparison functions that must be supported are:

- a_containedby
- a_contains
- a_equals
- a_overlaps

The Property-Property Comparisons requirements class drops the permission to restrict expressions on the left-hand side to properties and to restrict expressions on the right-hand side to literal values. This supports property-property, but also literal-literal or literal-property comparisons.

The Functions requirements class specifies requirements for supporting function calls (e.g. min, max, etc.) in a CQL2 expression. Function calls are the primary means of extending the language. Implementations should provide a capability to discover the available functions.

The Arithmetic Expressions requirements class specifies requirements for supporting the standard set of arithmetic operators (`+`, `-`, `*`, `/`, `%`, `div`, and `^`) in a CQL2 expression.

The CQL2 Text encoding requirements class defines a text encoding for CQL2. Such an encoding is suitable for use with HTTP query parameters such as the `filter` parameter defined by the "Filter" requirements class specified in OGC API - Features - Part 3: Filtering.

The CQL2 JSON encoding requirements class defines a JSON encoding for CQL2. Such as encoding is suitable for use as the body of an HTTP POST request.

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**
Basic CQL2	http://www.opengis.net/spec/cql2/1.0/conf/basic-cql2
Advanced Comparison Operators	http://www.opengis.net/spec/cql2/1.0/conf/advanced-comparison-operators
Case-insensitive Comparison	http://www.opengis.net/spec/cql2/1.0/conf/case-insensitive-comparison
Accent-insensitive Comparison	http://www.opengis.net/spec/cql2/1.0/conf/accent-insensitive-comparison
Basic Spatial Functions	http://www.opengis.net/spec/cql2/1.0/conf/basic-spatial-functions
Basic Spatial Functions with additional Spatial Literals	http://www.opengis.net/spec/cql2/1.0/conf/basic-spatial-functions-plus
Spatial Functions	http://www.opengis.net/spec/cql2/1.0/conf/spatial-functions
Temporal Functions	http://www.opengis.net/spec/cql2/1.0/conf/temporal-functions
Array Functions	http://www.opengis.net/spec/cql2/1.0/conf/array-functions
Property-Property Comparisons	http://www.opengis.net/spec/cql2/1.0/conf/property-property
Functions	http://www.opengis.net/spec/cql2/1.0/conf/functions
Arithmetic Expressions	http://www.opengis.net/spec/cql2/1.0/conf/arithmetic
CQL2 Text encoding	http://www.opengis.net/spec/cql2/1.0/conf/cql2-text
CQL2 JSON encoding	http://www.opengis.net/spec/cql2/1.0/conf/cql2-json

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 06-103r4: **OpenGIS® Implementation Standard for Geographic information - Simple feature access - Part 1: Common architecture** [online]. Edited by J. Herring. 2011 [viewed 2020-11-22]. Available at http://portal.opengeospatial.org/files/?artifact_id=25355
- Internet Engineering Task Force (IETF). RFC 7946: **The GeoJSON Format** [online]. Edited by H. Butler, M. Daly, A. Doyle, S. Gillies, S. Hagen, T. Schaub. 2016 [viewed 2020-03-16]. Available at https://www.rfc-editor.org/rfc/rfc7946.html
- Open Geospatial Consortium (OGC) / World Wide Web Consortium (W3C). **Time Ontology in OWL** [online]. Edited by S. Cox, C. Little. 2020 [viewed 2020-11-22]. Available at https://www.w3.org/TR/owl-time
- Internet Engineering Task Force (IETF). RFC 3339: **Date and Time on the Internet: Timestamps** [online]. Edited by G. Klyne, C. Newman. 2002 [viewed 2020-03-16]. Available at https://www.rfc-editor.org/rfc/rfc3339.html
- Internet Engineering Task Force (IETF). draft-handrews-json-schema-02: **JSON Schema: A Media Type for Describing JSON Documents** [online]. Edited by A. Wright, H. Andrews, B. Hutton, G. Dennis. 2019 [viewed 2020-11-22]. Available at https://datatracker.ietf.org/doc/html/draft-handrews-json-schema-02
- Unicode Consortium: **Unicode® 15.0.0** [online]. Available at https://unicode.org/versions/Unicode15.0.0/

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. boundary

set that represents the limit of an entity (ISO 19107:2019, definition 3.6)

NOTE	Boundary is most commonly used in the context of geometry, where the set is a collection of points or a collection of objects that represent those points. In other arenas, the term is used metaphorically to describe the transition between an entity and the rest of its domain of discourse.

4.1.2. 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).

4.1.3. custom function

function that is not specified in the CQL2 standard

NOTE	Custom functions require support for the Functions requirements class.

4.1.4. exterior

difference between the universe and the closure (ISO 19107:2019,definition 3.37)

NOTE	The concept of exterior is applicable to both topological and geometric complexes.

4.1.5. filter expression

predicate encoded for transmission between systems

4.1.6. function

rule that associates each element from a domain (source, or domain of the function) to a unique element in another domain (target, co-domain, or range) (ISO 19107:2003, definition 4.41)

4.1.7. interior

set of all direct positions that are on a geometric object but which are not on its boundary

NOTE	The interior of a topological object is the continuous image of the interior of any of its geometric realizations. This is not included as a definition because it follows from a theorem of topology. Another way of saying this is that any point on a geometric object is in its interior if it can be placed inside a homeomorphic image of an open set in the Euclidean space of the object's topological dimension.

4.1.8. standardized function

function specified in the CQL2 standard

4.1.9. 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)

4.1.10. queryable

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

4.1.11. resource

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

4.1.12. unicode case folding; case folding

process of making two texts which differ only in case identical for comparison purposes (W3C Character Model for the World Wide Web: String Matching)

NOTE	Case folding is meant for the purpose of case-insensitive string matching.

4.1.13. unicode normalization; normalization

process of removing alternate representations of equivalent sequences from textual data, to convert the data into a form that can be binary-compared for equivalence (Glossary of Unicode Terms)

4.2. Symbols

- ∩ intersection, operation on two or more sets

- ∩ and, logical intersection
- ∅ empty set, the set having no members
- ≠ not equal
- ⇔ if and only if, logical equivalence between statements
- ⊂ is a subset of
- **dim(x)** returns the maximum dimension (-1, 0, 1, or 2) of the geometric object x
- **I(x)** represents the interior of the geometric object x
- **B(x)** represents the boundary of the geometric object x
- **E(x)** represents the exterior of the geometric object x

4.3. Abbreviated terms

ABNF
: Augmented Backus-Naur Form

API
: Application Programming Interface

BNF
: Backus-Naur Form

CQL2
: Common Query Language

CRS
: Coordinate Reference System

DE-9IM
: Dimensionally Extended Nine-Intersection Model

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. Identifiers

The normative provisions in this standard are denoted by the URI `http://www.opengis.net/spec/cql2/1.0`.

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

5.2. Use of BNF

BNF as specified in Augmented BNF for Syntax Specifications is used to formally specify the grammar of the Common Query Language (CQL2) and its text encoding (CQL2-Text).

5.3. Use of JSON Schema

JSON Schema draft 2019-09 (JSON Schema, JSON Schema Validation) is used to formally specify the schema of the JSON encoding of CQL2 (CQL2-JSON).

5.4. Dependencies to other requirements classes

The requirements classes in this document 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 "Basic CQL2"

6.1. Overview

Requirements Class	
http://www.opengis.net/spec/cql2/1.0/req/basic-cql2	
Target type	Servers that evaluate filter expressions
Dependency	OGC Simple feature access - Part 1: Common architecture, Architecture
Dependency	W3C/OGC Time Ontology in OWL
Dependency	RFC 3339 (Date and Time on the Internet: Timestamps)
Dependency	JSON Schema: A Media Type for Describing JSON Documents
Dependency	Unicode

This clause defines the core of a query language called `Common Query Language` (CQL2) that may be used to construct filter expressions. This core is called Basic CQL2.

Subsequent clauses define additional filtering capabilities as well as several encodings of CQL2.

6.2. CQL2 filter expression

A CQL2 filter expression is an expression that defines a logically connected set of predicates that are evaluated for each item of a collection. Each predicate is an operator with operands where the number of operands depends on the operator. An operand is either a literal, a property, a standardized or custom function or an arithmetic expression.

A predicate is an expression that evaluates to the Boolean values of `TRUE` or `FALSE` or that evaluates to the value `NULL` when dealing with unknown values. Logically connected predicates are evaluated according to the following truth table:

Table 2. Truth table of evaluating CQL2 predicates

Predicate 1	**Predicate 2**	**Predicate1 AND Predicate2**	**Predicate1 OR Predicate2**
TRUE	TRUE	TRUE	TRUE
TRUE	FALSE	FALSE	TRUE
FALSE	TRUE	FALSE	TRUE
FALSE	FALSE	FALSE	FALSE
TRUE	NULL	NULL	TRUE
FALSE	NULL	FALSE	NULL
NULL	TRUE	NULL	TRUE
NULL	FALSE	FALSE	NULL

Predicate 1	Predicate 2	Predicate1 AND Predicate2	Predicate1 OR Predicate2
NULL	NULL	NULL	NULL

A collection item that satisfies ALL the requirements of a CQL2 filter expression according to the above true table evaluates to a Boolean value of `TRUE`; otherwise the CQL2 filter expression evaluates to `FALSE` or `NULL`.

If a CQL2 filter expression evaluates to `TRUE` for an item, the item is included in the result set and is thus available for further processing such as presentation in a response document.

If a CQL2 filter expression overall evaluates to `FALSE` or `NULL` for an item, the item is not included in the result set and is thus not available for further processing.

Requirement 1	/req/basic-cql2/cql2-filter
A	A server SHALL support a CQL2 filter expression composed of a logically connected series of one or more predicates as described by the BNF rule `booleanExpression` in CQL2 BNF with the exception that the rules `isLikePredicate`, `isBetweenPredicate`, `isInListPredicate`, `spatialPredicate`, `temporalPredicate`, `arrayPredicate`, `function` and `arithmeticExpression` as well as the functions `CASEI` and `ACCENTI` in the rules `characterExpression` and `patternExpression` do not have to be supported.

Literal values do not have to be supported on the left-hand side of predicates and property references do not have to be supported on the right-hand side of predicates.

Permission 1	/per/basic-cql2/cql2-filter
A	In the rule `binaryComparisonPredicate` the server MAY only support `propertyName` in the first `scalarExpression` rule as well as `characterLiteral`, `numericLiteral`, `booleanLiteral`, and `instantInstance` in the second `scalarExpression` rule.
B	In the rule `isNullPredicate` the server MAY only support `propertyName` in the `scalarExpression` rule.

A Basic CQL2 filter expression can be constructed by logically connecting comparison predicates.

Support for the parts of CQL2 that are not part of Basic CQL2 is added in additional requirements classes in Common Query Language enhancements:

- The rules `isLikePredicate`, `isBetweenPredicate` and `isInListPredicate` are added by requirements class Advanced Comparison Operators;
- Support for the `CASEI` function is added by requirements class Case-insensitive Comparison;
- Support for the `ACCENTI` function is added by requirements class Accent-insensitive Comparison;
- The rule `spatialPredicate` is added by requirements classes Basic Spatial Functions and Spatial Functions;

- The rule `temporalPredicate` is added by requirements class Temporal Functions.
- The rule `arrayPredicate` is added by requirements class Array Functions;
- The permission to not support the literal rules on the left-hand side of predicates and the rule `propertyName` on the right-hand side is removed by requirements class Property-Property Comparisons;
- The rule `function` is added by requirements class Functions;
- The rule `arithmeticExpression` is added by requirements class Arithmetic Expressions.

Examples of Basic CQL2 filter expressions are included in the subsequent sub-clauses.

6.3. Data types and literal values

This section documents the data types supported by Basic CQL2 and has examples of literal values for each data type.

A literal value is any part of an CQL2 filter expression that is used exactly as it is specified in the expression.

Other requirements classes add more data types. These are defined in the chapter specifying the requirements class.

6.3.1. Scalar data types

The scalar data types are:

- "string": character strings (rule `characterLiteral`);
- "number": numbers including integers and floating point values (rule `numericLiteral`);
- "boolean": booleans (rule `booleanLiteral`);
- "timestamp": an instant with a granularity of a second or smaller (rule `timestampInstant`)
- "date": an instant with a granularity of a day (rule `dateInstant`)

For character string, numeric and boolean literals, the standard representations are used.

Conceptually, an instant is a "temporal entity with zero extent or duration" [Time Ontology in OWL]. In practice, the temporal position of an instant is described using data types where each value has some duration or granularity that is sufficient for the intended use of the data.

CQL2 supports two commonly used granularities: a second or smaller (data type "timestamp") and days (data type "date"). Literal timestamps are always in the time zone UTC ("Z"), dates are local dates without an associated time zone.

If time zone information is important for the intended use, then the "date" data type should not be used and the temporal information should be provided as an interval with start and end timestamps.

NOTE	While instants (timestamps and dates) are scalar data types that can be used with

> the basic comparison operators, intervals are more complex data types. Support for intervals is added in the requirements class Temporal Functions where intervals can be provided as arguments in temporal comparison functions.

For timestamp and date values representations based on RFC 3339 are used:

- Text: a `DATE` or `TIMESTAMP` constructor with a RFC 3339 `date-time` or `full-date` string
- JSON: an object with a `date` or `timestamp` member with a RFC 3339 `date-time` or `full-date` string

Example 1. Scalar literal examples

- character string

```
'This is a literal string.'
```

- character string with an escaped embedded quote

```
'Via dell''Avvento'
```

- number

```
-100
3.14159
```

- boolean

```
true
false
```

- timestamp (Text)

```
TIMESTAMP('1969-07-20T20:17:40Z')
```

- timestamp (JSON)

```
{ "timestamp": "1969-07-20T20:17:40Z" }
```

- date (Text)

```
DATE('1969-07-20')
```

- date (JSON)

```
{ "date": "1969-07-20" }
```

6.3.2. Escaping in string literals

In general, escaping special character sequences in a string literal will be handled according to the rules of the specific encoding being used. For example, for the JSON encoding of CQL2, an embedded newline in a string literal would be encoded as `\n`. If, however, an XML encoding of CQL2 existed the embedded newline character would be encoded as `&#xA`. Furthermore, additional processing of a string literal may be necessary before it can be passed down to an underlying platform (e.g. RDBMS) for further handling.

For the text encoding of CQL2 see Requirements Class "CQL2 Text" and requirement /req/cql2-text/escaping for additional requirements concerning escaping in string literals.

6.3.3. Type casts

Permission 2	/per/basic-cql2/type-casts
A	If an operator or function has operands that have incompatible data types, the server MAY either return an error or it MAY cast the operands to compatible data types.

This Standard does not prescribe how types are cast. The evaluation of filter expressions that involve type casts will, therefore, be system dependent.

For example, a system that evaluates an expression `'5' > 4` can, for example,

- throw an error (incompatible types in a comparison operator);
- cast the number to a string (`'5' > '4'`);
- cast the string to a number (`5 > 4`).

6.4. Identifiers

An identifier is a token that represents a resource or a named part of a resource within a CQL2 expression that is not a CQL2 keyword or command. Identifiers are composed of a sequence of UTF-8 characters. Valid starting characters for identifiers can include the colon (i.e. ":"), the underscore (i.e. "_") and letters of the alphabet (e.g. "A-Z, a-z"). Additional continuing characters in an identifier can include the period (i.e ".") and numeric digits (i.e. "0-9"). The `identifier` production in the CQL2 BNF enumerates the specific characters that can be used to start an identifier as well as additional identifier continuing characters.

6.5. Property references

Properties in an object being evaluated in the CQL2 filter expression can be referenced by their name (rule `propertyName`).

Requirement 2	/req/basic-cql2/property
A	The property name (rule `propertyName`) SHALL be a queryable of the data.
B	The property name reference SHALL evaluate to its corresponding value, or `NULL` if unset.

For example, a property name used in a scalar expression (rule `scalarExpression`) has to be a queryable of type `string`, `number`, `integer`, `boolean`, `date`, or `timestamp`.

Example 2. Property reference in a scalar expression

In this example, the property `windSpeed` is used in a function that receives an array of numbers and returns a number.

```
avg(windSpeed)
```

```
{ "op": "avg", "args": [ { "property": "windSpeed" } ] }
```

6.6. Standard comparison predicates

Requirement 3	/req/basic-cql2/binary-comparison-predicate
A	A binary comparison predicate as specified by rule `binaryComparisonPredicate` evaluates two scalar expressions to determine if the expressions satisfy the specified comparison operator. If the requirements of the operator are satisfied, then the predicate SHALL evaluate to the Boolean value `TRUE`.
B	If the requirements of the operator are not satisfied, then the predicate SHALL evaluate to `FALSE`.
C	If either scalar expression (rule `scalarExpression`) of the predicate is `NULL` then the predicate SHALL evaluate to the value `NULL`;
D	Both scalar expressions (rule `scalarExpression`) in rule `binaryComparisonPredicate` SHALL evaluate to the same type of literal.

Recommendation 1	/rec/core/string-normalization
A	For any string comparisons, the server SHOULD implement unicode normalization described in the implementation guidelines of the Unicode 15.0.0 standard (see clause 5.6 Normalization).
B	The recommended normalization form is canonical decomposition (NFD).

Instants (timestamps and dates) are scalar data types. All implementations have to support the comparison of two timestamps or two dates. How this is implemented is a decision of the server and will depend on the internal representation. For example, the server could compare the RFC 3339 string representations of the two timestamps.

Example 3. Binary comparison predicates

```
city='Toronto'
```

```
{
  "op": "=",
  "args": [
    { "property": "city" },
    "Toronto"
  ]
}
```

```
avg(windSpeed) < 4
```

```
{
  "op": "<",
  "args": [
    {
      "op": "avg",
      "args": [ { "property": "windSpeed" } ]
    },
    4
  ]
}
```

```
balance-150.0 > 0
```

```
{
  "op": ">",
  "args": [
    {
      "op": "-",
      "args": [
        { "property": "balance" },
        150.0
      ]
    },
    0
  ]
```

```
}
```

```
updated >= date('1970-01-01')
```

```
{
  "op": ">=",
  "args": [
    { "property": "updated" },
    { "date": "1970-01-01" }
  ]
}
```

Requirement 4	/req/basic-cql2/null-predicate
A	The *null predicate* (rule `isNullPredicate`) tests whether the value of a scalar expression is null. The predicate SHALL be evaluated according to the following truth table:

Table 3. True table for the NOT operator

Predicate	NOT(Predicate)
TRUE	FALSE
FALSE	TRUE
NULL	NULL

Example 4. Examples of a NULL predicate

```
geometry IS NOT NULL
```

```
{
  "op": "not",
  "args": [
    {
      "op": "isNull",
      "args": [ { "property": "geometry" } ]
    }
  ]
}
```

6.7. CQL2 Encodings

This document defines a text encoding and a JSON encoding of CQL2 that covers Basic CQL2 and all enhanced capabilities specified in the next clause.

Chapter 7. Common Query Language enhancements

7.1. Overview

This clause specifies requirements for enhancements to Basic CQL2. Specifically, this clause defines requirements for:

- Advanced comparison operators;
- Case-insensitive comparison;
- Accent-insensitive comparison;
- Spatial functions;
- Temporal functions;
- Array functions;
- Property-property and literal-literal comparisons;
- Support for functions in CQL2;
- Support for arithmetic expression in CQL2;

In each case, this clause specifies requirements for the rules in CQL2 BNF not supported by Basic CQL2.

7.2. Requirements Class "Advanced Comparison Operators"

Requirements Class	
http://www.opengis.net/spec/cql2/1.0/req/advanced-comparison-operators	
Target type	Servers that evaluate filter expressions
Dependency	Requirements Class "Basic CQL2"

This requirements class adds support for the operators LIKE, BETWEEN and IN.

Requirement 5	**/req/advanced-comparison-operators/like-predicate**
A	The *like predicate* (rule `isLikePredicate`) tests whether a string value matches the specified pattern. If the value matches the pattern (rule `patternExpression`), then the predicate SHALL evaluate to the Boolean value `TRUE`.
B	If the value does not match the pattern (`patternExpression`), then the predicate SHALL evaluate to the Boolean value `FALSE`.

C	If the character expression (rule `characterExpression`) and/or the pattern expression (rule `patternExpression`) in the predicate is `NULL`, then the predicate SHALL evaluate to the value `NULL`.
D	The character expression (rule `characterExpression`) in rule `isLikePredicate` SHALL evaluate to a `characterLiteral`.
E	The wildcard character SHALL be the percent character (ASCII x25, `%`).
F	The wildcard SHALL match zero of more characters in the test value.
G	The wildcard character SHALL not match the NULL value.
H	The single character wildcard SHALL be the underbar character (ASCII x5F, `_`).
I	The single character wildcard SHALL match one character in the test value.
J	The single character wildcard SHALL not match the NULL value.
K	The escape character SHALL be the back slash (ASCII x5C, `\`).

Permission 3	**/per/advanced-comparison-operators/like-predicate**
A	The server MAY not support `characterLiteral` as the character expression (rule `characterExpression`) in rule `isLikePredicate`.

Example 5. Example of a LIKE predicate

```
name LIKE 'Smith%'
```

```
{
  "op": "like",
  "args": [
    { "property": "name" },
    "Smith%"
  ]
}
```

Requirement 6	**/req/advanced-comparison-operators/between-predicate**
A	The *between predicate* (rule `isBetweenPredicate`) tests whether a numeric value lies within the specified range. The between operator is inclusive. If the value lies within the specified range, then the predicate SHALL evaluate to the Boolean value `TRUE`.
B	If the value lies outside the specified range, then the predicate SHALL evaluate to the Boolean value `FALSE`.

C	If any numeric expression (rule `numericExpression`) in the predicate is `NULL` then the predicate SHALL evaluate to the value `NULL`.
D	Any function (rule `function`) or property (rule `propertyName`) in rule `isBetweenPredicate` SHALL evaluate to a `numericLiteral`.

Permission 4	**/per/advanced-comparison-operators/between-predicate**
A	The server MAY not support a `numericLiteral` as the first operand (rule `numericExpression`) in rule `isBetweenPredicate`.
B	The server MAY not support a `propertyName` as the second and third operand (rule `numericExpression`) in rule `isBetweenPredicate`.

Example 6. Examples of a BETWEEN predicate

```
depth BETWEEN 100.0 and 150.0
```

```
{
  "op": "between",
  "args": [
    { "property": "depth" },
    100.0,
    150.0
  ]
}
```

Requirement 7	**/req/advanced-comparison-operators/in-predicate**
A	The *in-list predicate* (rule `isInListPredicate`) tests, for equality, the value of a scalar expression against a list of values of the same type. If the value on the left side of the predicate is equal to one or more of the values in the list on the right side of the predicate, the predicate SHALL evaluate to the Boolean value `TRUE`. Otherwise the predicate SHALL evaluate to the Boolean value `FALSE`.
B	The items in the list of an in-list predicate (rule `inList`, i.e., the items on the right-hand side of the predicate) SHALL be of the same literal type as the value being tested by the predicate (rule `scalarExpression`, i.e., the left-hand side of the predicate), if evaluated.

Permission 5	**/per/advanced-comparison-operators/in-predicate**
A	The server MAY not support `characterLiteral`, `numericLiteral`, `booleanLiteral` or `instantInstance` as the value to be tested (rule `scalarExpression`, i.e., the left-hand side of the predicate).

B	The server MAY not support `propertyName` as the items in the list of an in-list predicate (rule `inList`, i.e., the items on the right-hand side of the predicate).

Example 7. Examples of a IN predicate

```
cityName IN ('Toronto','Frankfurt','Tokyo','New York')
```

```
{
  "op": "in",
  "args": [
    { "property": "cityName" },
    [ "Toronto", "Frankfurt", "Tokyo", "New York" ]
  ]
}
```

```
category NOT IN (1,2,3,4)
```

```
{
  "op": "not",
  "args": [
    {
      "op": "in",
      "args": [
        { "property": "category" },
        [ 1, 2, 3, 4 ]
      ]
    }
  ]
}
```

7.3. Requirements Class "Case-insensitive Comparison"

Requirements Class	
http://www.opengis.net/spec/cql2/1.0/req/case-insensitive-comparison	
Target type	Servers that evaluate filter expressions
Dependency	Requirements Class "Basic CQL2"

The following requirements class adds support for case-insensitive string comparisons.

This capability is useful to operate across data that has not been normalized or has been normalized to values that are different than they should be. This is implemented via a standardized string function to normalize a string with respect to case (`CASEI`).

For example, the `CASEI` function is useful when a property is set to "PLANET", "Planet", or "planet" and one wants to match either without having to enumerate all the variations.

Implementations of the `CASEI` function can be complex and depend on the locale, but in many cases the underlying datastore will provide a capability that the function can be mapped to.

Requirement 8	**/req/case-insensitive-comparison/casei-function**
A	The server SHALL support a function named `CASEI`.
B	The function SHALL accept one argument that can be a character string literal, the name of a property that evaluates to a character string literal or a function that returns a character string literal (see rules `characterLiteral`, `propertyName`, `function`).
C	The function SHALL return a character string.
D	If the argument to the function is `NULL`, the function SHALL return a `NULL` value.
E	The function SHALL implement the full case folding algorithm defined in the implementation guidelines of the Unicode 15.0.0 standard (see clause 5.18 Case Mappings, sub-clause Caseless Matching, CaseFolding-15.0.0.txt and SpecialCasing-15.0.0.txt).

NOTE

Implementation Guidance for CASEI()
The implementation of case folding makes use of the CaseFolding-15.0.0.txt file and replaces code points in the source string by the corresponding sequence on lines with a 'C'(ommon) or 'F'(ull).

Example 8. Example case-insensitive comparison

```
CASEI(road_class) IN (CASEI('Οδος'),CASEI('Straße'))
```

```
{
  "op": "in",
  "args": [
    {
      "op": "casei",
      "args": [ { "property": "road_class" } ]
    },
    [
      { "op": "casei", "args": [ "Οδος" ] },
      { "op": "casei", "args": [ "Straße" ] }
    ]
  ]
```

```
}
```

The `CASEI` function returns a string typed representation of the input expression that is guaranteed to be equal to any other case insensitive representation of that string. In order to ensure correct comparisons, the function should be applied to both sides of an expression. So, for example, the only durable case-insensitive equality comparison would be `CASEI(some_property) = CASEI('Straße')`. An expression such as `CASEI(some_property) = 'strasse'` might work but is not guaranteed to work across implementations or between versions of the same implementation.

7.4. Requirements Class "Accent-insensitive Comparison"

Requirements Class	
http://www.opengis.net/spec/cql2/1.0/req/accent-insensitive-comparison	
Target type	Servers that evaluate filter expressions
Dependency	Requirements Class "Basic CQL2"

This requirements class adds support for accent-insensitive string comparisons to operate across data that has not been normalized or has been normalized to values that are different than they should be.

Similar to the case-insensitive comparison, this capability is supported via a string function `ACCENTI`.

For example, the `ACCENTI` function is useful when accents (or, more generally, diacritics not available in ASCII) were dropped when indexing a property. This may be useful, for example, to support users that are not familiar with accents or that do not know how to type them on their keyboard. For example, "papa" would also match "papá". Note that accent-insensitive comparisons can match values with a different meaning. E.g., in Spanish "papa" is potato and "papá" is father. "papá" in an accent-insensitive comparison will match both, but this may also be intentional, because the users knows that some of the data has been processed in ASCII.

Implementations of the `ACCENTI` function can be complex, but in many cases the underlying datastore will provide a capability that the function can be mapped to.

Requirement 9	**/req/accent-insensitive-comparison/accenti-function**
A	The server SHALL support a function named `ACCENTI`.
B	The function SHALL accept one argument that can be a character string literal, the name of a property that evaluates to a character string literal or a function that returns a character string literal (see rules `characterLiteral`, `propertyName`, `function`).
C	The function SHALL return a character string.
D	If the argument to the function is `NULL`, the function SHALL return a `NULL` value.

E	The function SHALL implement accent stripping and diacritic folding.

Example 9. Example accent-insensitive comparison

```
ACCENTI(etat_vol) = ACCENTI('débárquér')
```

```
{
  "op": "=",
  "args": [
    {
      "op": "accenti",
      "args": [ { "property": "etat_vol" } ]
    },
    {
      "op": "accenti",
      "args": [ "débárquér" ]
    }
  ]
}
```

Like `CASEI`, the `ACCENTI` function returns a string typed representation of the input expression that is guaranteed to be equal to any other accent insensitive representation of that string. In order to ensure correct comparisons, the function should be applied to both sides of an expression. So, for example, the only durable accent-insensitive equality comparison would be `ACCENTI(some_property) = ACCENTI('papá')`. An expression such as `ACCENTI(some_property) = 'papa'` might work but is not guaranteed to work across implementations or between versions of the same implementation.

NOTE

Implementation guidance for ACCENTI()

The implementation of an ACCENTI() function requires the use of fields 3 and 5 from UnicodeData.txt and the application of the Unicode Normalization Algorithm (NFD or NFKD) by:

- Recursively replacing code points in the source string by their field 5 Decomposition Mappings for those rows with canonical mappings (i.e. those mappings not prefixed by a tag from Table 14, or any of them if applying NFKD; the decomposition type is also available in DerivedDecompositionType.txt).
- Applying special rules to decompose Hangul syllables which do not have a decomposition mapping set up. There are some details in http://www.unicode.org/versions/Unicode9.0.0/ch03.pdf Section 3.12 of the Unicode Standard (from page 142) and https://stackoverflow.com/questions/41309402/breaking-down-a-hangul-syllable-into-letters-jamo. The following C code implements the hangul syllable decomposition:

```
if(codePoint >= 0xAC00 && codePoint < 0xD7B0)
{
   unsigned int syllable = codePoint - 0xAC00;
   unsigned int t = syllable % 28, v, l;
   syllable /= 28;
   v = syllable % 21, l = syllable / 21;
   add(0x1100 + l);
   add(0x1161 + v);
   if(t) add(0x11A7 + t);
}
```

- Applying the Canonical Ordering algorithm which is stable-sorting (e.g., bubble-sort) the decomposed mapping code points by the value of that combining class for any sub-string where the Combining Class (field 3) (also in DerivedCombiningClass.txt) value is non-zero. This step is necessary if there were combining marks in the source text; the Decomposition Mappings should otherwise already be in the correct order. The canonical ordering will only matter for code points that do not get stripped, so it will not matter for any of the combining characters that are non-spacing marks.

then:

- Removing Nonspacing Marks (category Mn) (general category is field 2 of UnicodeData.txt, also available in DerivedGeneralCategory.txt).
- An exception should be made for some characters categorized as Mn, as stripping some non-spacing marks — like the Japanese voicing marks (dakuten [゛] U+3099 and handakuten [゜] U+309A) — can be a lossy change that would turn はじめ "hajime" (beginning) into はしめ "hashime" (fastener).

Recommendation 2	**/rec/accent-insensitive-comparison/japanese-non-spacing-marks** Implementations of the ACCENTI() function SHOULD not remove the Japanese non-spacing marks dakuten U+3099[゛] and handakuten U+309A[゜].

7.5. Requirements Class "Basic Spatial Functions"

Requirements Class	
http://www.opengis.net/spec/cql2/1.0/req/basic-spatial-functions	
Target type	Servers that evaluate filter expressions
Dependency	Requirements Class "Basic CQL2"
Dependency	OGC Simple feature access - Part 1: Common architecture, Architecture

A *spatial predicate* evaluates two geometry-valued expressions to determine if the expressions satisfy the requirements of the specified spatial comparison function.

7.5.1. Basic spatial data types and literal values

The basic spatial data types are (part of rule `spatialInstance`):

- "Point": a point;
- "BBox": a bounding rectangle or box.

For the Point data type, the following representations are used for literal values:

- Text: an OGC Well-Known Text (WKT) literal (see clause 7 of Simple feature access - Part 1: Common architecture)
- JSON: a GeoJSON geometry object (see clause 3.1 of GeoJSON)

For the BBox data type:

In the Text representation, the type is encoded as a `BBOX()` spatial comparison function with four or six numerical arguments, depending on whether the coordinates include a vertical axis (height or depth):

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

In cases where the bounding box spans the antimeridian of a geographic coordinate reference system, the lower-left value (west-most box edge) is larger than the upper-right value (east-most box edge).

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

In JSON, the BBox type is encoded as a JSON object with a "bbox" member with an array with four or six numbers. This representation is consistent with the GeoJSON representation of a bounding box (see clause 5 of GeoJSON).

Since WKT and GeoJSON do not provide a capability to specify the CRS of a geometry literal, the server has to determine the CRS of the geometry literals in a filter expression through another mechanism. For example, a query parameter `filter-crs` is used in OGC API - Features - Part 3: Filtering to pass the CRS information to the server.

Example 10. Spatial literal example

- spatial geometry (Text)

```
POINT(43.5845 -79.5442)
```

- spatial geometry (JSON)

```
{
   "type": "Point",
   "coordinates": [43.5845,-79.5442]
}
```

- bounding box (Text)

```
BBOX(160.6,-55.95,-170,-25.89)
```

- bounding box (JSON)

```
{
   "bbox": [160.6, -55.95, -170, -25.89]
}
```

7.5.2. Spatial Functions

In this conformance class, the only required spatial comparison function is *intersects* and the only required spatial literals are point and BBox. Additional spatial literals are specified in the Basic Spatial Functions with additional Spatial Literals requirements class and additional spatial comparison functions are specified in the Spatial Functions requirements class.

Requirement 10	**/req/basic-spatial-operators/spatial-predicate**
A	If the requirements of the standardized spatial comparison function are satisfied, then the predicate SHALL evaluate to the Boolean value `TRUE`.
B	If the requirements of the standardized spatial comparison function are not satisfied, then the predicate SHALL evaluate to the Boolean value `FALSE`.
C	If either geometry expression (rule `geomExpression`) of the predicate is `NULL` then the predicate SHALL evaluate to the value `NULL`.

Requirement 11	**/req/basic-spatial-functions/spatial-functions**
A	The server SHALL support the standardized `S_INTERSECTS` spatial comparison function as defined by the BNF rule `spatialFunction` in CQL2 BNF.

B	All supported standardized spatial comparison functions SHALL be evaluated as defined in clause 6.1.15 of OpenGIS® Implementation Standard for Geographic information - Simple feature access - Part 1: Common architecture (except that in CQL2 the predicates evaluate to a Boolean, not an Integer).

Permission 6	**/per/basic-spatial-functions/spatial-predicates**
A	The server MAY not support a `spatialInstance` as the first operand (rule `geomExpression`) in rule `spatialPredicate`.
B	The server MAY not support a `propertyName` as the second operand (rule `geomExpression`) in rule `spatialPredicate`.

Permission 7	**/per/basic-spatial-functions/spatial-data-types**
A	The server MAY only support `pointTaggedText` and `bboxTaggedText` in rule `spatialInstance`.

7.5.3. Examples

Example 11. Example spatial predicate

```
S_INTERSECTS(geometry,POINT(36.319836 32.288087))
```

```
{
  "op": "s_intersects",
  "args": [
    { "property": "geometry" },
    {
      "type": "Point",
      "coordinates": [ 36.319836, 32.288087 ]
    }
  ]
}
```

Example 12. Example for the filter-crs query parameter

```
...filter-lang=cql2-text&
   filter-crs=http://www.opengis.net/def/crs/EPSG/0/32635&
   filter=S_INTERSECTS(geometry,POINT(379213.87 3610774.16))...
```

Note that the values of the `filter-crs` and `filter` parameters have not been percent-encoded (see section 2.1 of RFC 3986) in this example for better readability.

7.6. Requirements Class "Basic Spatial Functions with additional Spatial Literals"

Requirements Class	
http://www.opengis.net/spec/cql2/1.0/req/basic-spatial-functions-plus	
Target type	Servers that evaluate filter expressions
Dependency	Requirements Class "Basic Spatial Functions"
Dependency	OGC Simple feature access - Part 1: Common architecture, Architecture

This requirements class is similar to the Basic Spatial Functions requirements class except it removes the restrictions on the spatial literals that can participate in the expression.

7.6.1. Additional spatial data types and literal values

In addition to the spatial types listed in the Basic Spatial Functions requirements class (i.e. "Point", "BBox"), this requirements class allows the following Spatial Literals (part of rule `spatialInstance`):

- "LineString": a curve with linear interpolation between the vertices;
- "Polygon": a planar surface bounded by closed line strings;
- "MultiPoint": a collection of points;
- "MultiLineString": a collection of line strings;
- "MultiPolygon": a collection of polygons;
- "GeometryCollection": a collection of one or more of "Point", "Polygon", "MultiPoint", "MultiLineString", or "MultiPolygon" instances;

Requirement 12	/req/basic-spatial-functions-plus/spatial-data-types
A	The server SHALL support all spatial literals as defined by the BNF rule `spatialInstance` in CQL2 BNF.

The following representations are used for literal values:

- Text: an OGC Well-Known Text (WKT) literal (see clause 7 of Simple feature access - Part 1: Common architecture)
- JSON: a GeoJSON geometry object (see clause 3.1 of GeoJSON)

Example 13. Spatial literal example

- spatial geometry (Text)

```
LINESTRING(43.6776 -79.5792, 43.7089 -79.5532, 43.7184 -79.5169, 43.7314 -79.4503,
43.7592 -79.4037, 43.7681 -79.3384, 43.8118 -79.3473, 43.8118 -79.3473, 43.8416
-79.3673)
```

```
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))
```

- spatial geometry (JSON)

```
{
  "type": "LineString",
  "coordinates": [
    [43.6776,-79.5792],
    [43.7089,-79.5532],
    [43.7184,-79.5169],
    [43.7314,-79.4503],
    [43.7592,-79.4037],
    [43.7681,-79.3384],
    [43.8118,-79.3473],
    [43.8118,-79.3473],
    [43.8416,-79.3673]
  ]
}

{
  "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]
    ]
  ]
}
```

Example 14. Example spatial predicate

```
S_INTERSECTS(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)))
```

```
{
  "op": "s_intersects",
  "args": [
    { "property": "geometry" },
    {
      "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]
        ]
      ]
    }
  ]
}
```

Example 15. Example for the filter-crs query parameter

```
...filter-lang=cql2-text&
   filter-crs=http://www.opengis.net/def/crs/EPSG/0/32635&
   filter=S_INTERSECTS(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)))...
```

Note that the values of the `filter-crs` and `filter` parameters have not been percent-encoded (see section 2.1 of RFC 3986) in this example for better readability.

7.7. Requirements Class "Spatial Functions"

Requirements Class	
http://www.opengis.net/spec/cql2/1.0/req/spatial-functions	
Target type	Servers that evaluate filter expressions
Dependency	Requirements Class "Basic Spatial Functions"

Dependency	Requirements Class "Basic Spatial Functions with additional Spatial Literals"

This requirements class adds:

- a set of Dimensionally Extended Nine-intersection Model (DE-9IM) relation operators that may be used to add spatial predicates to a CQL2 filter expression. These operators are implemented in CQL2 as standardized functions.

7.7.1. Spatial Functions

Requirement 13	/req/spatial-functions/spatial-functions
A	The server SHALL support all standardized spatial comparison functions as defined by the BNF rule `spatialFunction` in CQL2 BNF.

This clause specifies a set of standardized spatial comparison functions that can be used to evaluate whether a specific spatial relationship exists between a pair of geometries.

The definition of these spatial comparison functions is based on a Dimensionally Extended 9-Intersection Model (DE-9IM) and further discussion and explanation about this model can be found at DE-9IM and Dimensionally Extended 9-Intersection Model.

Consider geometries `a` and `b`.

The spatial relationships between `a` and `b` can be represented by the following intersection matrix (see DE-9IM):

$$\text{DE9IM}(a,b) = \begin{bmatrix} \dim(I(a) \cap I(b)) & \dim(I(a) \cap B(b)) & \dim(I(a) \cap E(b)) \\ \dim(B(a) \cap I(b)) & \dim(B(a) \cap B(b)) & \dim(B(a) \cap E(b)) \\ \dim(E(a) \cap I(b)) & \dim(E(a) \cap B(b)) & \dim(E(a) \cap E(b)) \end{bmatrix}$$

Figure 1. The DE-9IM intersection matrix.

`I()` represents the set of all positions in the interior of the geometry, `B()` represents the set of all positions on the boundary of the geometry and `E()` represents the set of all exterior positions. `dim()` represents the dimension of the intersection of the interior (`I`), boundary (`B`) and exterior (`E`) of geometries `a` and `b`.

The value of each cell in this intersection matrix is either:

- `0` (i.e. the dimension of the intersection is a point),
- `1` (i.e. the dimension of the intersection is a line),
- `2` (i.e. the dimension of the intersection is an area) or,
- `` for the empty set or no intersection.

These values are sometimes simplified to:

- `T` representing `{0,1,2}` (if the actual value of the dimension does not matter),

- `F` representing the empty set and,
- `*` representing a value that is not relevant to the evaluation of a spatial comparison function.

An example of such an intersection matrix is:

```
[[T,"*",F],["*","*",F],[F,F,"*"]]
```

The following table lists the mathematical definitions of each standardized spatial comparison function as described in OpenGIS® Implementation Standard for Geographic information - Simple feature access - Part 1: Common architecture and also using DE-9IM.

Table 4. Mathematical definitions of standardized spatial comparison functions

Spatial comparison function	**Definition**	**Intersection matrix**
S_CONTAINS	S_CONTAINS(a,b) ⇔ b WITHIN a	`[[T,"*","*"],["*","*","*"],[F,F,"*"]]`
S_CROSSES	S_CROSSES(a,b) ⇔ [I(a) ∩ I(b) ≠ ∅) ∧ (a ∩ b ≠ a) ∧ (a ∩ b ≠ b)]	`[[T,"*",T],["*","*","*"],["*","*","*"]]`, `[[T,"*","*"],["*","*","*"],[T,"*","*"]]`, `[[0,"*","*"],["*","*","*"],["*","*","*"]]`
S_DISJOINT	S_DISJOINT(a,b) ⇔ a ∩ b = ∅	`[[F,F,"*"],[F,F,"*"],["*","*","*"]]`
S_EQUALS	S_EQUALS(a,b) ⇔ a ⊆ b ∧ b ⊆ a	`[[T,"*",F],["*","*",F],[F,F,"*"]]`
S_INTERSECTS	S_INTERSECTS(a,b) ⇔ ! a DISJOINT b	`[[T,"*","*"],["*","*","*"],["*","*","*"]]`, `[["*",T,"*"],["*","*","*"],["*","*","*"]]`, `[["*","*","*"],[T,"*","*"],["*","*","*"]]`, `[["*","*","*"],["*",T,"*"],["*","*","*"]]`
S_OVERLAPS	S_OVERLAPS(a,b) ⇔ (dim(I(a)) = dim(I(b)) = dim(I(a) ∩ I(b))) ∧ (a ∩ b ≠ a) ∧ (a ∩ b ≠ b)	`[[T,"*",T],["*","*","*"],[T,"*","*"]]`,`[[1,"*",T],["*","*","*"],[T,"*","*"]]`,
S_TOUCHES	S_TOUCHES(a,b) ⇔ (I(a) ∩ I(b) = ∅) ∧ (a ∩ b) ≠ ∅	`[[F,T,"*"],["*","*","*"],["*","*","*"]]`,`[[F,"*","*"],[F,"*","*"],["*","*","*"]]`,`[[F,"*","*"],["*",T,"*"],["*","*","*"]]`
S_WITHIN	S_WITHIN(a,b) ⇔ (a ∩ b = a) ∧ (I(a) ∩ E(b) = ∅)	`[[T,"*",F],["*","*",F],["*","*","*"]]`

The following diagrams illustrate the meaning of the `S_CROSSES`, `S_OVERLAPS`, `S_TOUCHES` and `S_WITHIN` spatial comparison functions.

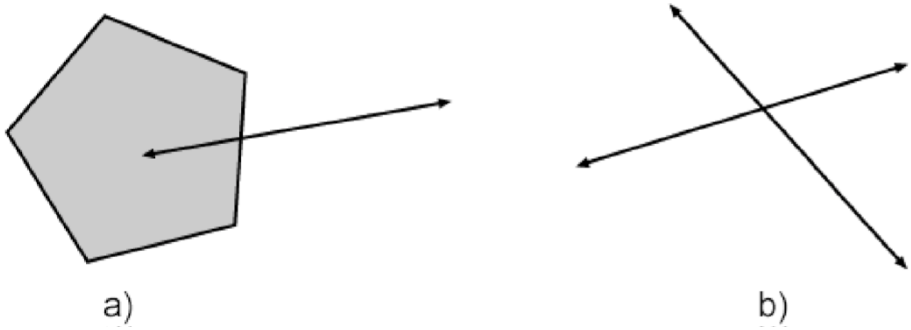

Figure 2. Examples of the S_CROSSES relationship Polygon/LineString(a) and LineString/LineString(b).

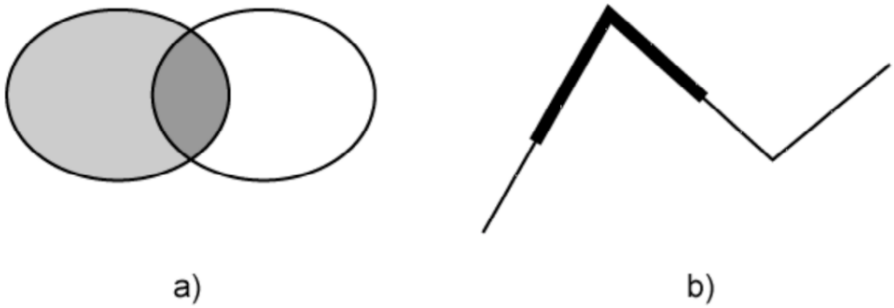

Figure 3. Examples of the S_OVERLAPS relationship Polygon/LineString(a) and LineString/LineString(b).

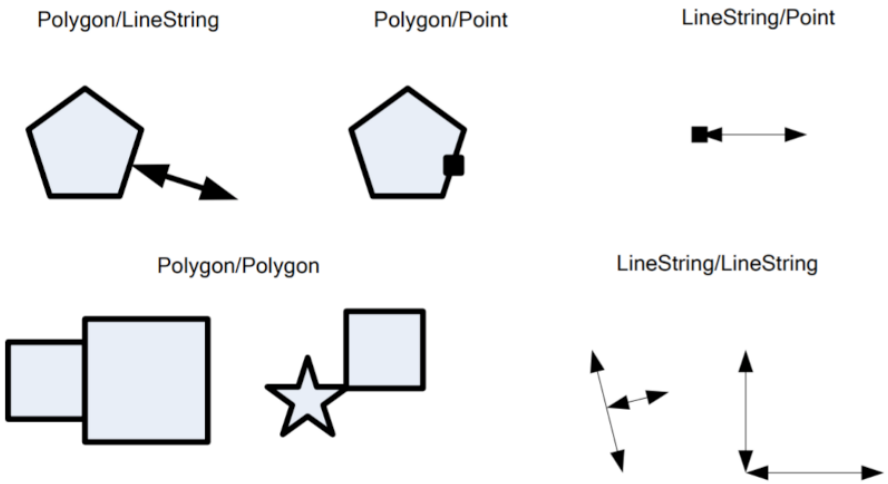

Figure 4. Examples of the S_TOUCHES relationship

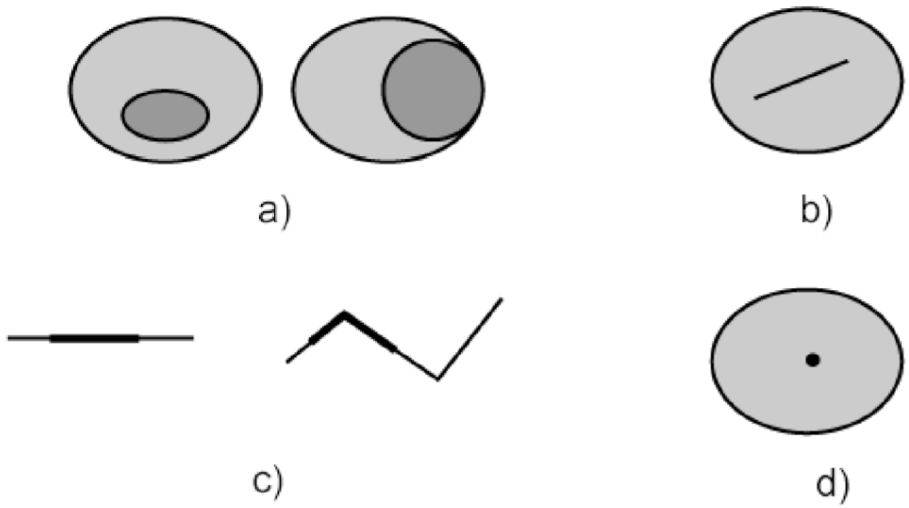

Figure 5. Examples of the S_WITHIN relationship Polygon/Polygon(a), Polygon/LineString(b), LineString/LineString(c), and Polygon/Point(d)

NOTE If geometry ***a*** `S_CONTAINS` geometry ***b***, then geometry ***b*** is `S_WITHIN` geometry ***a***.

Example 16. Example of a spatial relationship between a property and a literal geometry.

```
S_CROSSES(road,POLYGON((43.7286 -79.2986, 43.7311 -79.2996, 43.7323 -79.2972,
                        43.7326 -79.2971, 43.7350 -79.2981, 43.7350 -79.2982,
                        43.7352 -79.2982, 43.7357 -79.2956, 43.7337 -79.2948,
                        43.7343 -79.2933, 43.7339 -79.2923, 43.7327 -79.2947,
                        43.7320 -79.2942, 43.7322 -79.2937, 43.7306 -79.2930,
                        43.7303 -79.2930, 43.7299 -79.2928, 43.7286 -79.2986)))
```

```
{
  "op": "s_crosses",
  "args": [
    { "property": "road" },
    {
      "type": "Polygon",
      "coordinates": [
        [
          [ 43.7286, -79.2986 ], [ 43.7311, -79.2996 ], [ 43.7323, -79.2972 ],
          [ 43.7326, -79.2971 ], [ 43.7350, -79.2981 ], [ 43.7350, -79.2982 ],
          [ 43.7352, -79.2982 ], [ 43.7357, -79.2956 ], [ 43.7337, -79.2948 ],
          [ 43.7343, -79.2933 ], [ 43.7339, -79.2923 ], [ 43.7327, -79.2947 ],
          [ 43.7320, -79.2942 ], [ 43.7322, -79.2937 ], [ 43.7306, -79.2930 ],
          [ 43.7303, -79.2930 ], [ 43.7299, -79.2928 ], [ 43.7286, -79.2986 ]
        ]
      ]
    }
  ]
}
```

7.8. Requirements Class "Temporal Functions"

Requirements Class	
http://www.opengis.net/spec/cql2/1.0/req/temporal-functions	
Target type	Servers that evaluate filter expressions
Dependency	Requirements Class "Basic CQL2"
Dependency	W3C/OGC Time Ontology in OWL, Topological Temporal Relations

A temporal predicate evaluates two time-valued expressions to determine, if the expressions satisfy the requirements of the specified standardized temporal comparison function.

The operands in a temporal predicate are temporal geometries. A temporal geometry is either an instant or an interval.

7.8.1. Temporal data types and instances

An instant is either a date (rule `dateInstant`) or a timestamp (rule `timestampInstant`) in accordance with RFC 3339 (RFC 3339 rules `full-date` or `date-time`). Note that since time is continuous, every instant has a duration and a start/end. Nevertheless, the geometry can be considered an instant in the temporal resolution that is applicable for the specific property.

An interval is the time between a start instant and an end instant, including both bounding instances (rule `intervalInstance`). Unbounded interval ends are represented by a double-dot string ("..") based on the convention specified in ISO 8601-2.

CQL2 follows ISO 8601-1/ISO 8601-2 in defining intervals as closed at both start and end. Note that some implementations and other specifications use a different definition and it may be necessary to convert between the interval representations. For example, SQL uses half-closed intervals - closed at the start, open at the end.

Depending on the implementation environment, the underlying datastore may or may not support intervals as data types of properties. If not, intervals are typically represented by two separate properties that are instants, one for the start and one for the end of the interval.

All temporal geometries are in the Gregorian Calendar. This is a deliberate restriction to keep implementations of CQL2 simple, avoiding requirements to transform time instants to other temporal coordinate reference systems, but still cover a large number of use cases. This is consistent with the use of RFC 3339 as a key standard for expressing date and time on the internet, including in the OGC API Standards.

For intervals, the following representations are used:

- Text: an `INTERVAL` function with two instants or double-dot strings as parameters;
- JSON: an object with an `interval` member with an array of two instants or double-dot strings as parameters.

In case two instants are provided, both instants have the same granularity (i.e., they are either

timestamps or dates).

NOTE	Instants are also scalar data types; for the representations and examples of instances see Scalar data types.

Example 17. Interval examples

- intervals (Text)

```
INTERVAL('1969-07-16', '1969-07-24')
INTERVAL('1969-07-16T05:32:00Z', '1969-07-24T16:50:35Z')
INTERVAL('2019-09-09', '..')
```

- intervals (JSON)

```
{ "interval": [ "1969-07-16", "1969-07-24" ] }
{ "interval": [ "1969-07-16T05:32:00Z", "1969-07-24T16:50:35Z" ] }
{ "interval": [ "2019-09-09", ".." ] }
```

7.8.2. Temporal Functions

The standardized temporal comparison functions in CQL2 are based on the temporal operator definitions in the W3C/OGC Time Ontology in OWL.

NOTE	Simple temporal predicates involving time instants can also be evaluated using the standard comparison operators.

The following table specifies the definition of the standardized temporal comparison functions where both operands may be instants or intervals, including mixed combinations.

Table 5. Definitions of standardized temporal comparison functions that support both instants and intervals

Temporal comparison function	**Definition (t1: first operand, t2: second operand)**
T_AFTER	See after
T_BEFORE	See before
T_DISJOINT	(t1 T_BEFORE t2) OR (t1 T_AFTER t2)
T_EQUALS	Start and end of t1 and t2 are coincident
T_INTERSECTS	NOT (t1 T_DISJOINT t2)

Additional temporal comparison functions are available, but only applicable for intervals. Using these functions with instants will result in a client error.

Table 6. Definitions of standardized temporal comparison function between intervals

Temporal comparison function	Definition
T_CONTAINS	See intervalContains
T_DURING	See intervalDuring
T_FINISHEDBY	See intervalFinishedBy
T_FINISHES	See intervalFinishes
T_MEETS	See intervalMeets
T_METBY	See intervalMetBy
T_OVERLAPPEDBY	See intervalOverlappedBy
T_OVERLAPS	See intervalOverlaps
T_STARTEDBY	See intervalStarts
T_STARTS	See intervalStartedBy

The following diagram illustrates the meaning of most of the standardized temporal comparison functions when applied to intervals.

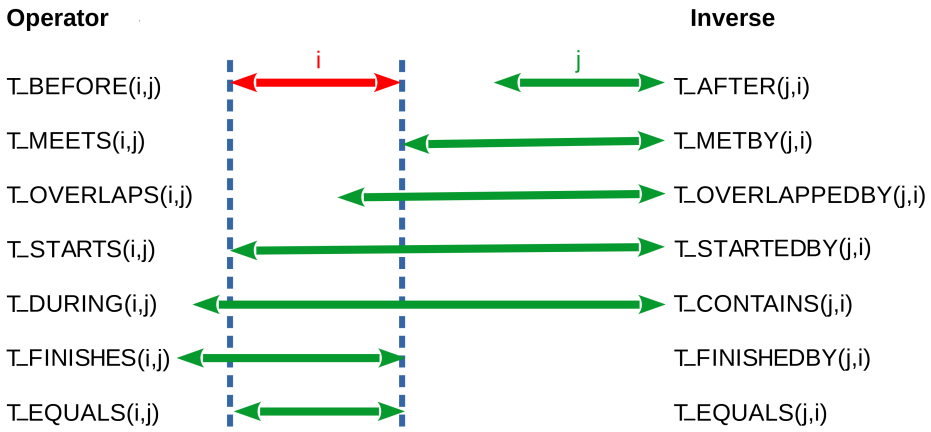

Figure 6. The elementary relations between time intervals

Requirement 14	/req/temporal-operators/temporal-predicates
A	If the requirements of the standardized temporal comparison function are satisfied, then the predicate SHALL evaluate to the Boolean value `TRUE`.
B	If the requirements of the standardized temporal comparison function are not satisfied, then the predicate SHALL evaluate to the Boolean value `FALSE`.
C	If the either temporal expression (rule `temporalExpression`) of the standardized temporal comparison function is `NULL`, then the predicate SHALL evaluate to the value `NULL`.

Requirement 15	/req/temporal-functions/temporal-functions
A	The server SHALL support all temporal functions as defined by the BNF rule `temporalFunction` in CQL2 BNF.

B	The temporal functions SHALL be evaluated as defined in the tables Table 5 and Table 6.

Permission 8	**/per/temporal-functions/temporal-operands**
A	The server MAY not support a `temporalInstance` as the first operand (rule `temporalExpression`) in rule `temporalPredicate`.
B	The server MAY not support a `propertyName` as the second operand (rule `temporalExpression`) in rule `temporalPredicate`.

7.8.3. Type casts

As stated in Type casts, the evaluation of filter expressions that involve type casts is system dependent.

For example, a system that evaluates the interval `INTERVAL('2022-01-01','2022-04-11T12:41:13Z')` can, for example,

- throw an error (incompatible types in an interval);
- cast the value to an interval of dates, e.g., `INTERVAL('2022-01-01','2022-04-11')`;
- cast the value to an interval of timestamps, e.g., `INTERVAL('2022-01-01Z00:00:00Z','2022-04-11T12:41:13Z')`.

7.8.4. Examples

Example 18. Examples of temporal predicate using T_INTERSECTS

```
T_INTERSECTS(event_time, INTERVAL('1969-07-16T05:32:00Z', '1969-07-24T16:50:35Z'))
```

```
{
  "op": "t_intersects",
  "args": [
    { "property": "event_time" },
    { "interval": [ "1969-07-16T05:32:00Z", "1969-07-24T16:50:35Z" ] }
  ]
}
```

Example 19. Examples of temporal relationships using a property and a temporal literal.

```
T_DURING(INTERVAL(touchdown, liftOff), INTERVAL('1969-07-16T13:32:00Z', '1969-07-
24T16:50:35Z'))
```

```
{
```

```
  "op": "t_during",
  "args": [
    { "interval": [ { "property": "touchdown" }, { "property": "liftOff" } ] },
    { "interval": [ "1969-07-16T13:32:00Z", "1969-07-24T16:50:35Z" ] }
  ]
}
```

7.9. Requirements class "Array Functions"

Requirements Class	
http://www.opengis.net/spec/cql2/1.0/req/array-functions	
Target type	Servers that evaluate filter expressions
Dependency	Requirements Class "Basic CQL2"

This clause specifies requirements for supporting array expression in CQL2.

7.9.1. Arrays

An array is a bracket-delimited, comma-separated list of array elements. An array element is either a scalar value, a geometry, an interval, or another array.

Example 20. Array examples

- arrays (Text)

```
( 'a', 'c' )
( 'a', true, 1 )
( DATE('1969-07-16'), DATE('1969-07-20'), DATE('1969-07-24') )
```

- arrays (JSON)

```
[ "a", "c" ]
[ "a", true, 1 ]
[ { "date" : "1969-07-16" }, { "date" : "1969-07-20" }, { "date" : "1969-07-24" }
]
```

7.9.2. Array Functions

Array expressions can be tested in a predicate for equality, if one array is a subset of another, if one array is a superset of another or if two arrays overlap or share elements using a standardized set of array comparison functions.

Requirement 16	/req/array-functions/array-predicates

A	The server SHALL support arrays as defined by the BNF rule `arrayPredicate` in CQL2 BNF with the exception of the following rules: • `function` in `arrayExpression`, `arrayElement` • `arithmeticExpression` and `function` in `arrayElement`.
B	Both array expressions SHALL be evaluated as sets. No inherent order SHALL be implied in an array of values.
C	The semantics of the standardized array comparison functions SHALL be evaluated as follows: • `A_EQUALS` evaluates to the Boolean value `TRUE`, if both sets are identical; otherwise the predicate SHALL evaluate to the Boolean value `FALSE`. • `A_CONTAINS` evaluates to the Boolean value `TRUE`, if the first set is a superset of the second set; otherwise the predicate SHALL evaluate to the Boolean value `FALSE`. • `A_CONTAINEDBY` evaluates to the Boolean value `TRUE`, if the first set is a subset of the second set; otherwise the predicate SHALL evaluate to the Boolean value `FALSE`. • `A_OVERLAPS` evaluates to the Boolean value `TRUE`, if both sets share at least one common element; otherwise the predicate SHALL evaluate to the Boolean value `FALSE`.

Permission 9	**/per/array-functions/array-predicates**
A	The server MAY not support an `array` as the first operand (rule `arrayExpression`) in rule `arrayPredicate`.
B	The server MAY not support a `propertyName` as the second operand (rule `arrayExpression`) in rule `arrayPredicate`.

NOTE Support for the BNF rule `function` is added by the requirements class Functions. Support for the BNF rule `arithmeticExpression` is added by the requirements class Arithmetic Expressions.

7.9.3. Examples

Example 21. Evaluate if the value of an array property contains the specified subset of values.

```
A_CONTAINS(layer:ids, ('layers-ca','layers-us'))
```

```
{
  "op": "a_contains",
  "args": [
```

```
    { "property": "layer:ids" },
    [ "layers-ca", "layers-us" ]
  ]
}
```

7.10. Requirements Class "Property-Property Comparisons"

Requirements Class	
http://www.opengis.net/spec/cql2/1.0/req/property-property	
Target type	Servers that evaluate filter expressions
Dependency	Requirements Class "Basic CQL2"
Conditional Dependency	Requirements Class "Advanced Comparison Operators"
Conditional Dependency	Requirements Class "Basic Spatial Functions"
Conditional Dependency	Requirements Class "Spatial Functions"
Conditional Dependency	Requirements Class "Temporal Functions"
Conditional Dependency	Requirements Class "Array Functions"

This requirements class adds support for properties on the right side of predicates and for literal on the left side of predicates.

Requirement 17	**/req/property-property/withdraw-permissions**
A	The following permissions SHALL not apply: • **/per/basic-cql2/cql2filter** • **/per/advanced-comparison-operators/like-predicate** • **/per/advanced-comparison-operators/between-predicate** • **/per/advanced-comparison-operators/in-predicate** • **/per/basic-spatial-functions/spatial-predicates** • **/per/temporal-functions/temporal-predicates** • **/per/array-functions/array-predicates**

Example 22. Example of a spatial relationship between two literal geometries.

```
S_CROSSES(LINESTRING(43.72992 -79.2998, 43.73005 -79.2991, 43.73006 -79.2984,
                     43.73140 -79.2956, 43.73259 -79.2950, 43.73266 -79.2945,
                     43.73320 -79.2936, 43.73378 -79.2936, 43.73486 -79.2917),
        POLYGON((43.7286 -79.2986, 43.7311 -79.2996, 43.7323 -79.2972, 43.7326
-79.2971,
                 43.7350 -79.2981, 43.7350 -79.2982, 43.7352 -79.2982, 43.7357
-79.2956,
                 43.7337 -79.2948, 43.7343 -79.2933, 43.7339 -79.2923, 43.7327
-79.2947,
                 43.7320 -79.2942, 43.7322 -79.2937, 43.7306 -79.2930, 43.7303
-79.2930,
                 43.7299 -79.2928, 43.7286 -79.2986)))
```

```
{
  "op": "s_crosses",
  "args": [
    {
      "type": "LineString",
      "coordinates": [
        [ 43.72992, -79.2998 ], [ 43.73005, -79.2991 ], [ 43.73006, -79.2984 ],
        [ 43.73140, -79.2956 ], [ 43.73259, -79.2950 ], [ 43.73266, -79.2945 ],
        [ 43.73320, -79.2936 ], [ 43.73378, -79.2936 ], [ 43.73486, -79.2917 ]
      ]
    },
    {
      "type": "Polygon",
      "coordinates": [
        [
          [ 43.7286, -79.2986 ], [ 43.7311, -79.2996 ], [ 43.7323, -79.2972 ],
          [ 43.7326, -79.2971 ], [ 43.7350, -79.2981 ], [ 43.7350, -79.2982 ],
          [ 43.7352, -79.2982 ], [ 43.7357, -79.2956 ], [ 43.7337, -79.2948 ],
          [ 43.7343, -79.2933 ], [ 43.7339, -79.2923 ], [ 43.7327, -79.2947 ],
          [ 43.7320, -79.2942 ], [ 43.7322, -79.2937 ], [ 43.7306, -79.2930 ],
          [ 43.7303, -79.2930 ], [ 43.7299, -79.2928 ], [ 43.7286, -79.2986 ]
        ]
      ]
    }
  ]
}
```

Example 23. Examples of temporal relationships using temporal literals.

```
T_DURING(INTERVAL('1969-07-20T20:17:40Z', '1969-07-21T17:54:00Z'), INTERVAL('1969-
07-16T13:32:00Z', '1969-07-24T16:50:35Z'))
```

```
{
  "op": "t_during",
  "args": [
    { "interval": [ "1969-07-20T20:17:40Z", "1969-07-21T17:54:00Z" ] },
    { "interval": [ "1969-07-16T13:32:00Z", "1969-07-24T16:50:35Z" ] }
  ]
}
```

7.11. Requirements Class "Functions"

Requirements Class	
http://www.opengis.net/spec/cql2/1.0/req/functions	
Target type	Servers that evaluate filter expressions
Dependency	Requirements Class "Basic CQL2"

This sub-clause specifies requirements for supporting custom functions in CQL2. Functions allow implementations to extend the language.

Requirement 18	**/req/functions/functions**
A	The server SHALL support custom functions as defined by the BNF rules `function` in CQL2 BNF with the exception of the rule `arithmeticExpression` in `argument`.
B	The function SHALL evaluate to its return value that is allowed in the same rule in which the function is used.

NOTE Support for the BNF rule `arithmeticExpression` is added by the requirements class Arithmetic Expressions.

Example 24. Example of a spatial relationship between a property and a function that return a geometry value.

It should be noted that the function "Buffer()" in this example is not part of CQL2 but is an example of a function that an implementation may offer that returns a geometry value.

```
S_WITHIN(road,Buffer(geometry,10,'m'))
```

```
{
  "op": "s_within",
  "args": [
    { "property": "road" },
    {
      "op": "Buffer",
```

```
      "args": [
        { "property": "geometry" },
        10,
        "m"
      ]
    }
  ]
}
```

7.12. Requirements Class "Arithmetic Expressions"

Requirements Class	
http://www.opengis.net/spec/cql2/1.0/req/arithmetic	
Target type	Servers that evaluate filter expressions
Dependency	Requirements Class "Basic CQL2"

This clause specifies requirements for supporting arithmetic expressions in CQL2. An arithmetic expression is an expression composed of an arithmetic operand (a property name, a number or a function that returns a number), an arithmetic operator (i.e., one of `+`, `-`, `*`, `/`, `%`, `div`, or `^`) and another arithmetic operand.

`+`, `-`, `*`, and `/` are the four basic arithmetic operations (addition, subtraction, multiplication and division). In addition, the modulo operator (`%`), integer division (`div`), and the exponention operator (`^`) are supported.

Requirement 19	/req/arithmetic/arithmetic
A	The server SHALL support arithmetic expressions as defined by the BNF rules `arithmeticExpression` in CQL2 BNF with the exception of the rule `function` in `arithmeticOperand`.
B	Is any `arithmeticOperand` in an arithmetic expression is `NULL`, then the expression SHALL evaluate to `NULL`.

NOTE Support for the BNF rule `function` is added by the requirements class Functions.

Example 25. Predicate with an arithmetic expression finding all vehicles that are too tall to pass under a bridge.

```
vehicle_height > (bridge_clearance-1)
```

```
{
  "op": ">",
  "args": [
    { "property": "vehicle_height" },
```

```
    {
      "op": "-",
      "args": [
        { "property": "bridge_clearance" },
        1
      ]
    }
  ]
}
```

Chapter 8. Requirements classes for encodings

8.1. Overview

This clause specifies requirements for a text encoding and a JSON encoding of CQL2.

8.2. Requirements Class "CQL2 Text"

Requirements Class	
http://www.opengis.net/spec/cql2/1.0/req/cql2-text	
Target type	Servers that evaluate filter expressions
Dependency	Requirements Class "Basic CQL2"
Conditional Dependency	Simple feature access - Part 1: Common architecture, Well-known Text Representation for Geometry
Conditional Dependency	Requirements Class "Advanced Comparison Operators"
Conditional Dependency	Requirements Class "Case-insensitive Comparisons"
Conditional Dependency	Requirements Class "Accent-insensitive Comparisons"
Conditional Dependency	Requirements Class "Basic Spatial Functions"
Conditional Dependency	Requirements Class "Basic Spatial Functions with additional Spatial Literals"
Conditional Dependency	Requirements Class "Spatial Functions"
Conditional Dependency	Requirements Class "Temporal Functions"
Conditional Dependency	Requirements Class "Array Functions"
Conditional Dependency	Requirements Class "Property-Property Comparisons"
Conditional Dependency	Requirements Class "Functions"
Conditional Dependency	Requirements Class "Arithmetic Expressions"

This requirements class defines a Well Known Text (WKT) encoding of CQL2. Such an encoding

would be suitable for use with the GET query parameter such as the `filter` query parameter specified by the "Filter" requirements class in OGC API - Features - Part 3: Filtering.

The "CQL2 Text" encoding is defined by the BNF grammar defined in CQL2 BNF.

Keywords in the BNF grammar are case-insensitive. Augmented BNF for Syntax Specifications states:

> ABNF strings are case insensitive and the character set for these strings is US-ASCII.

The list of CQL2 keywords includes:

- "A_EQUALS"
- "A_CONTAINS"
- "A_CONTAINEDBY"
- "A_OVERLAPS"
- "ACCENTI"
- "AND"
- "BBOX"
- "BETWEEN"
- "CASEI"
- "DATE"
- "DIV"
- "FALSE"
- "GEOMETRYCOLLECTION"
- "IN"
- "IS"
- "LIKE"
- "LINESTRING"
- "MULTILINESTRING"
- "MULTIPOINT"
- "MULTIPOLYGON"
- "NOT"
- "NULL"
- "OR"
- "POINT"
- "POLYGON"

- "S_INTERSECTS"
- "S_EQUALS"
- "S_DISJOINT"
- "S_TOUCHES"
- "S_WITHIN"
- "S_OVERLAPS"
- "S_CROSSES"
- "S_CONTAINS"
- "T_AFTER"
- "T_BEFORE"
- "T_CONTAINS"
- "T_DISJOINT"
- "T_DURING"
- "T_EQUALS"
- "T_FINISHEDBY"
- "T_FINISHES"
- "T_INTERSECTS"
- "T_MEETS"
- "T_METBY"
- "T_OVERLAPPEDBY"
- "T_OVERLAPS"
- "T_STARTEDBY"
- "T_STARTS"
- "TIMESTAMP"
- "TRUE"

If a queryable uses a property name that is one of the keywords, the property name can be used in double quotes to avoid conflicts with the keyword.

Requirement 20	**/req/cql2-text/basic-cql2**
A	The server SHALL be able to parse and evaluate all filter expressions encoded as a text string that validate against the BNF rules identified in the Basic CQL2 requirements class.

Requirement 21	**/req/cql2-text/escaping**
A	The escape character in a character literal SHALL be the backslash (\\).

B	Embedded single quotations (') in a character literal SHALL be escaped using a double single quotation ('') OR the backslash (\\) character.
C	The server SHALL be able to parse the following escaped sequences for encoding control characters in a character literal: • '\a' BELL CHR(07) • '\b' BACKSPACE CHR(08) • '\t' HORIZONTAL TAB CHR(09) • '\n' NEWLINE CHR(10) • '\v' VERTICAL TAB CHR(11) • '\f' FORM FEED CHR(12) • '\r' CARRIAGE RETURN CHR(13)

Requirement 22	**/req/cql2-text/advanced-comparison-operators**
Condition	Server implements requirements class Advanced Comparison Operators
A	The server SHALL be able to parse and evaluate all spatial functions encoded as a text string that validate against the BNF production fragments identified in the Advanced Comparison Operators requirements class.

Requirement 23	**/req/cql2-text/case-insensitive-comparison**
Condition	Server implements requirements class Case-insensitive Comparisons
A	The server SHALL be able to parse and evaluate all standardized functions encoded as a text string that validate against the BNF production fragments identified in the Case-insensitive Comparisons requirements class.

Requirement 24	**/req/cql2-text/accent-insensitive-comparison**
Condition	Server implements requirements class Accent-insensitive Comparisons
A	The server SHALL be able to parse and evaluate all standardized functions encoded as a text string that validate against the BNF production fragments identified in the Accent-insensitive Comparisons requirements class.

Requirement 25	**/req/cql2-text/basic-spatial-functions**
Condition	Server implements requirements class Basic Spatial Functions

A	The server SHALL be able to parse and evaluate all standardized spatial comparison functions encoded as a text string that validate against the BNF production fragments identified in the Basic Spatial Functions requirements class.

Requirement 26	**/req/cql2-text/basic-spatial-functions-plus**
Condition	Server implements requirements class Basic Spatial Functions with additional Spatial Literals
A	The server SHALL be able to parse all spatial instances encoded as a text string that validate against the BNF production fragments identified in the Basic Spatial Functions with additional Spatial Literals requirements class.
B	The server SHALL support spatial literals with coordinates in a 2D or 3D CRS, independent of including the `Z` or not in the value.

OGC WKT uses a "Z" character to distinguish the dimensionality of the coordinate reference system (CRS), e.g. `POINT(7 51)` for a 2D CRS and `POINT Z(7 51 100)` for a 3D CRS. In CQL2 Text, processors are required to be more tolerant and to determine the coordinate dimension from the coordinates.

When creating a CQL2 Text expression, it is recommended to encode a geometry literal as required by OGC WKT.

Requirement 27	**/req/cql2-text/spatial-functions**
Condition	Server implements requirements class Spatial Functions
A	The server SHALL be able to parse and evaluate all standardized spatial comparison functions encoded as a text string that validate against the BNF production fragments identified in the Spatial Functions requirements class.

Requirement 28	**/req/cql2-text/temporal-functions**
Condition	Server implements requirements class Temporal Functions
A	The server SHALL be able to parse and evaluate all standardized temporal comparison functions encoded as a text string that validate against the BNF production fragments identified in the Temporal Functions requirements class.

Requirement 29	**/req/cql2-text/arrays**
Condition	Server implements requirements class Array Functions
A	The server SHALL be able to parse and evaluate all array expressions encoded as a text string that validate against the BNF production fragments identified in the Array Expressions requirements class.

Requirement 30	/req/cql2-text/property-property
Condition	Server implements requirements class Property-Property Comparisons
A	The server SHALL be able to parse and evaluate literal values on the left-hand side of comparison, spatial, temporal or array operators and property references on the right-hand side of such operators.

Requirement 31	/req/cql2-text/functions
Condition	Server implements requirements class Functions
A	The server SHALL be able to parse and evaluate all function call expressions encoded as a text string that validate against the BNF production fragments identified in the Functions requirements class.

Requirement 32	/req/cql2-text/arithmetic
Condition	Server implements requirements class Arithmetic Expressions
A	The server SHALL be able to parse and evaluate all arithmetic expressions encoded as a text string that validate against the BNF production fragments identified in the Arithmetic Expressions requirements class.

8.3. Requirements Class "CQL2 JSON"

Requirements Class	
http://www.opengis.net/spec/cql2/1.0/req/cql2-json	
Target type	Servers that evaluate filter expressions
Dependency	Requirements Class "Basic CQL2"
Conditional Dependency	GeoJSON, Geometry Objects
Conditional Dependency	Requirements Class "Advanced Comparison Operators"
Conditional Dependency	Requirements Class "Case-insensitive Comparisons"
Conditional Dependency	Requirements Class "Accent-insensitive Comparisons"
Conditional Dependency	Requirements Class "Basic Spatial Functions"
Conditional Dependency	Requirements Class "Basic Spatial Functions with additional Spatial Literals"

Conditional Dependency	Requirements Class "Spatial Functions"
Conditional Dependency	Requirements Class "Temporal Functions"
Conditional Dependency	Requirements Class "Array Functions"
Conditional Dependency	Requirements Class "Property-Property Comparisons"
Conditional Dependency	Requirements Class "Functions"
Conditional Dependency	Requirements Class "Arithmetic Expressions"

This requirements class defines a JSON encoding of CQL2. Such an encoding would be suitable as the body of an HTTP POST request.

Requirement 33	**/req/cql2-json/basic-cql2**
A	The server SHALL be able to parse and evaluate all filter expressions encoded as JSON that validate against the JSON Schema in JSON Schema for CQL2 and that do not use the following schema components: • "#/$defs/isLikePredicate" • "#/$defs/isBetweenPredicate" • "#/$defs/isInListPredicate" • "#/$defs/casei" • "#/$defs/accenti" • "#/$defs/spatialPredicate" • "#/$defs/temporalPredicate" • "#/$defs/arrayPredicate" • "#/$defs/functionRef" • "#/$defs/arithmeticExpression"

Permission 10	**/per/cql2-json/basic-cql2**

A	The server MAY not support • the schema component "#/$defs/propertyRef" in an operand of a comparison operator, standardized spatial, temporal or array comparison functions starting with the second operand, • the schema components "#/$defs/characterExpression", "#/$defs/numericExpression", "#/$defs/spatialInstance", "#/$defs/instantInstance", "#/$defs/intervalInstance", "#/$defs/array", "#/$defs/functionRef", or a boolean literal in the first operand of a comparison operator, standardized spatial, temporal or array comparison function.

Requirement 34	**/req/cql2-text/advanced-comparison-operators**
Condition	Server implements requirements class Advanced Comparison Operators
A	In addition to the Basic CQL2 requirement, the server SHALL be able to parse and evaluate filter expressions encoded as JSON that use the following schema components: • "#/$defs/isLikePredicate" • "#/$defs/isBetweenPredicate" • "#/$defs/isInListPredicate"

Requirement 35	**/req/cql2-text/case-insensitive-comparison**
Condition	Server implements requirements class Case-insensitive Comparisons
A	In addition to the Basic CQL2 requirement, the server SHALL be able to parse and evaluate filter expressions encoded as JSON that use the following schema component: • "#/$defs/casei"

Requirement 36	**/req/cql2-text/accent-insensitive-comparison**
Condition	Server implements requirements class Accent-insensitive Comparisons
A	In addition to the Basic CQL2 requirement, the server SHALL be able to parse and evaluate filter expressions encoded as JSON that use the following schema component: • "#/$defs/accenti"

Requirement 37	**/req/cql2-json/basic-spatial-functions**
Condition	Server implements requirements class Basic Spatial Functions

A	The server SHALL be able to parse and evaluate filter expressions encoded as JSON that use the following schema components: • "#/$defs/spatialPredicate" where property "op" has the value "s_intersects" • "#/$defs/spatialInstance" where the value is either "#/$defs/point" or "#/$defs/bboxLiteral"

Requirement 38	**/req/cql2-json/basic-spatial-functions-plus**
Condition	Server implements requirements class Basic Spatial Functions with additional Spatial Literals
A	The server SHALL be able to parse and evaluate filter expressions encoded as JSON that use the following schema components: • "#/$defs/spatialPredicate" where property "op" has the value "s_intersects"

Requirement 39	**/req/cql2-json/spatial-functions**
Condition	Server implements requirements class Spatial Functions
A	The server SHALL be able to parse and evaluate filter expressions encoded as JSON that use the following schema components: • "#/$defs/spatialPredicate"

Requirement 40	**/req/cql2-json/temporal-functions**
Condition	Server implements requirements class Temporal Functions
A	In addition to the Basic CQL2 requirement, the server SHALL be able to parse and evaluate filter expressions encoded as JSON that use the following schema components: • "#/$defs/temporalPredicate"

Requirement 41	**/req/cql2-json/arrays**
Condition	Server implements requirements class Array Functions
A	In addition to the Basic CQL2 requirement, the server SHALL be able to parse and evaluate filter expressions encoded as JSON that use the following schema component: • "#/$defs/arrayPredicate"

Requirement 42	**/req/cql2-json/property-property**
Condition	Server implements requirements class Property-Property Comparisons

A	In addition to the Basic CQL2 requirement, the server SHALL be able to parse and evaluate filter expressions encoded as JSON that use • the schema component "#/$defs/propertyRef" in an operand of a comparison operator, standardized spatial, temporal or array comparison function starting with the second operand, • the schema components "#/$defs/characterExpression", "#/$defs/numericExpression", "#/$defs/spatialInstance", "#/$defs/instantInstance", "#/$defs/intervalInstance", "#/$defs/array", "#/$defs/functionRef", or a boolean literal in the first operand of a comparison operator, standardized spatial, temporal or array comparison function.

Requirement 43	**/req/cql2-json/functions**
Condition	Server implements requirements class Functions
A	In addition to the Basic CQL2 requirement, the server SHALL be able to parse and evaluate filter expressions encoded as JSON that use the following schema component: • "#/$defs/functionRef"

Requirement 44	**/req/cql2-json/arithmetic**
Condition	Server implements requirements class Arithmetic Expressions
A	In addition to the Basic CQL2 requirement, the server SHALL be able to parse and evaluate filter expressions encoded as JSON that use the following schema component: • "#/$defs/arithmeticExpression"

8.4. XML encoding

This document does not specifically define an XML-encoding for CQL2. However, it is recognized that XML is still in common use and so implementers are directed to review the OGC Filter Encoding 2.0 standard which defines an XML-encoding for filter expressions that closely matches most of the functionality of CQL2.

Chapter 9. Media Types

No media type has been registered for the CQL2 encodings. The reason is the assumption that filter expressions as such will rarely be used as standalone documents, but usually be part of another document, e.g. a HTTP request or response, and be embedded in it.

Annex A: Abstract Test Suite (Normative)

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

Each implementation under test supports the evaluation of filter expressions on one or more data sources. A data source may be, for example, a feature collection in an OGC Web API that implements the OGC API - Features Standard. The implementation must declare the queryable properties (queryables) for each data source.

The requirements specified in this Standard can only be tested, if the test can assess whether the result of an evaluation matches the filter expression. However, in general, the queryables may not be part of the response; that is, it is not possible to perform such an assessment in general without knowledge about the data and the relationship between the queryables and the data. In addition, to assess the implementation of some requirements (e.g., case folding) the test needs to know the data.

With just the knowledge about the queryables and their data types, the tests can typically assess that valid filter expressions for a set of queryables are evaluated without an error. These tests are basic tests specified in this test suite and can be executed against any data.

In addition, to properly test an implementation, conditional tests are provided, if the implementation operates on a test dataset and where the queryables are properties of the features.

The test dataset contains feature types with point, line string and polygon geometries. It is available as a GeoPackage file with the associated queryables specified in JSON Schema for each feature collection in the OGC API Features GitHub repository. The test dataset has been derived from three layers of the Natural Earth vector dataset at scale 1:110 million (ne_110m_admin_0_countries, ne_110m_populated_places_simple, ne_110m_rivers_lake_centerlines). Some columns have been removed and a few columns have been added with random data in order to also test filter expressions on date, timestamp and boolean properties.

The tests assume that

- all feature properties are also queryables;
- the queryable for the feature geometry is `geom`.

All eleven conformance classes for the standardization target type "servers that evaluate filter expressions" have the following parameter:

- Filter Language: The CQL2 encoding to be used in the test, either "CQL2 Text" or "CQL2 JSON".

General rules for the encoding of all filter expressions in tests:

- The conformance tests in this annex are specified using the CQL2 BNF grammar. Depending on the Filter Language parameter, the filter expressions have to be instantiated in an executable test as CQL2 Text or CQL2 JSON;
- If a property name in a filter expression is a reserved CQL2 keyword, the property name has to be placed in double quotes.

Executable test suites for the eleven conformance classes will also need to decide on the following

questions and support at least one option per question:

- How to execute the evaluation of a filter expression for a data source? At a minimum, Web API endpoints specified in OGC API - Features - Part 3: Filtering, requirements class "Filter", should be supported;
- What are the queryables for a data source? At a minimum, queryables specified using JSON Schema, see OGC API - Features - Part 3: Filtering, requirements class "Filter", should be supported;
- What is the format of the response that matches the filter expression? At a minimum, GeoJSON feature collections should be supported.

To qualify as an OGC Reference Implementation for CQL2, an implementation under test has to

- support the tests with the test dataset;
- support both CQL2 Text and CQL2 JSON;
- support at least the conformance classes "Case-insensitive comparison", "Spatial functions" and "Temporal functions" (including all dependencies).

A.1. Conformance Class "CQL2 Text"

Conformance Class	
http://www.opengis.net/spec/cql2/1.0/conf/cql2-text	
Target type	Servers that evaluate filter expressions
Requirements class	Requirements Class "CQL2 Text"
Dependency	Basic CQL2
Conditional Dependency	Advanced Comparison Operators
Conditional Dependency	Case-insensitive Comparisons
Conditional Dependency	Accent-insensitive Comparisons
Conditional Dependency	Basic Spatial Functions
Conditional Dependency	Basic Spatial Functions with additional Spatial Literals
Conditional Dependency	Spatial Functions
Conditional Dependency	Temporal Functions

Conditional Dependency	Array Functions
Conditional Dependency	Property-Property Comparisons
Conditional Dependency	Functions
Conditional Dependency	Arithmetic Expressions

A.1.1. Conformance Test 1

Test id:	/conf/cql2-text/validate
Requirements:	all requirements
Test purpose:	Validate that CQL2 Text is supported by the server
Test method:	Given: • n/a When: Execute conformance tests for all supported conformance classes with the parameter "Filter Language". Use the value "CQL2 Text". Then: • assert that all conformance tests are successful.

A.1.2. Conformance Test 2

Test id:	/conf/cql2-text/escaping
Requirements:	/req/cql2-text/escaping
Test purpose:	Test escaping in string literals.

Test method:	Given: • One or more data sources containing string literals with embedded single quotation (') and/or BELL, and/or BACKSPACE, and/or HORIZONTAL TAB, and/or NEWLINE, and/or VERTICAL TAB, and/or FORM FEED, and/or CARRIAGE RETURN characters. When: Decode each string literal. Then: • assert that the escaped embedded characters have been correctly recovered.

A.2. Conformance Class "CQL2 JSON"

Conformance Class	
http://www.opengis.net/spec/cql2/1.0/conf/cql2-json	
Target type	Servers that evaluate filter expressions
Requirements class	Requirements Class "CQL2 JSON"
Dependency	Basic CQL2
Conditional Dependency	Advanced Comparison Operators
Conditional Dependency	Case-insensitive Comparisons
Conditional Dependency	Accent-insensitive Comparisons
Conditional Dependency	Basic Spatial Functions
Conditional Dependency	Basic Spatial Functions with additional Spatial Literals
Conditional Dependency	Spatial Functions
Conditional Dependency	Temporal Functions
Conditional Dependency	Array Functions
Conditional Dependency	Property-Property Comparisons

Conditional Dependency	Functions
Conditional Dependency	Arithmetic Expressions

A.2.1. Conformance Test 3

Test id:	/conf/cql2-json/validate
Requirements:	all requirements
Test purpose:	Validate that CQL2 JSON is supported by the server
Test method:	Given: • A filter expression When: Execute conformance tests for all supported conformance classes with the parameter "Filter Language". Use the value "CQL2 JSON". Note that the filter expressions in the test cases have to be converted to a CQL2 JSON representation. Then: • assert the validation is successful.

A.3. Conformance Class "Basic-CQL2"

Conformance Class	
http://www.opengis.net/spec/cql2/1.0/conf/basic-cql2	
Target type	Servers that evaluate filter expressions
Parameter	Filter Language: "CQL2 Text" or "CQL2 JSON"
Requirements class	Requirements Class "Basic-CQL2"

A.3.1. Conformance Test 4

Test id:	/conf/basic-cql2/basic-test
Requirements:	n/a
Test purpose:	Implementation under test provides sufficient information to construct filter expressions and supports comparison predicates

Test method:	Given: • One or more data sources, each with a list of queryables. When: n/a Then: • assert that there is at least one queryable for each data source; • assert that the data type (String, Number, Integer, Boolean, Timestamp, Date, Interval, Point, MultiPoint, LineString, MultiLineString, Polygon, MultiPolygon, Geometry, GeometryCollection, or Array) is specified for each queryable; • assert that at least one queryable for each data source is of data type String, Boolean, Number, Integer, Timestamp or Date.

A.3.2. Conformance Test 5

Test id:	/conf/basic-cql2/comparison
Requirements:	/req/basic-cql2/cql2-filter, /req/basic-cql2/property, /req/basic-cql2/binary-comparison-predicate
Test purpose:	Test comparison predicates

Test method:	Given: • One or more data sources, each with a list of queryables. • Test '/conf/basic-cql2/basic-test' passes. When: For each queryable `{queryable}` of one of the data types String, Boolean, Number, Integer, Timestamp or Date, evaluate the following filter expressions • `{queryable} = {value}` • `{queryable} <> {value}` • `{queryable} > {value}` • `{queryable} < {value}` • `{queryable} >= {value}` • `{queryable} <= {value}` where `{value}` depends on the data type: • String: `'foo'` • Boolean: `true` • Number: `3.14` • Integer: `1` • Timestamp: `TIMESTAMP('2022-04-14T14:48:46Z')` • Date: `DATE('2022-04-14')` Then: • assert successful execution of the evaluation; • assert that the two result sets for each queryable for the operators `=` and `<>` have no item in common; • assert that the two result sets for each queryable for the operators `>` and `<=` have no item in common; • assert that the two result sets for each queryable for the operators `<` and `>=` have no item in common; • store the valid predicates for each data source.

A.3.3. Conformance Test 6

Test id:	/conf/basic-cql2/is-null
Requirements:	/req/basic-cql2/cql2-filter, /req/basic-cql2/property, /req/basic-cql2/null-predicate

Test purpose:	Test IS NULL predicate
Test method:	Given: • One or more data sources, each with a list of queryables. • Test '/conf/basic-cql2/basic-test' passes. When: For each queryable `{queryable}`, evaluate the following filter expressions • `{queryable} IS NULL` • `{queryable} is not null` Then: • assert successful execution of the evaluation; • assert that the two result sets for each queryable have no item in common; • store the valid predicates for each data source.

A.3.4. Conformance Test 7

Test id:	/conf/basic-cql2/boolean
Requirements:	/req/basic-cql2/cql2-filter
Test purpose:	Test boolean filter expression
Test method:	Given: • One or more data sources. • Test '/conf/basic-cql2/basic-test' passes. When: For each data source, evaluate the following filter expressions • `true` • `false` Then: • assert successful execution of the evaluation; • assert that the result sets for `false` are empty; • store the valid predicates for each data source.

A.3.5. Conformance Test 8

Test id:	/conf/basic-cql2/test-data

Requirements:	all requirements
Test purpose:	Test predicates against the test dataset
Test method:	Given: • The implementation under test uses the test dataset. When: Evaluate each predicate in Predicates and expected results. Then: • assert successful execution of the evaluation; • assert that the expected result is returned; • store the valid predicates for each data source.

Table 7. Predicates and expected results

Data Source	Predicate	Expected number of items
ne_110m_admin_0_countries	`NAME='Luxembourg'`	1
ne_110m_admin_0_countries	`NAME>='Luxembourg'`	84
ne_110m_admin_0_countries	`NAME>'Luxembourg'`	83
ne_110m_admin_0_countries	`NAME<='Luxembourg'`	94
ne_110m_admin_0_countries	`NAME<'Luxembourg'`	93
ne_110m_admin_0_countries	`NAME<>'Luxembourg'`	176
ne_110m_admin_0_countries	`POP_EST=37589262`	1
ne_110m_admin_0_countries	`POP_EST>=37589262`	39
ne_110m_admin_0_countries	`POP_EST>37589262`	38
ne_110m_admin_0_countries	`POP_EST<=37589262`	139
ne_110m_admin_0_countries	`POP_EST<37589262`	138
ne_110m_admin_0_countries	`POP_EST<>37589262`	176
ne_110m_populated_places_simple	`name IS NOT NULL`	243
ne_110m_populated_places_simple	`name IS NULL`	0
ne_110m_populated_places_simple	`name='København'`	1
ne_110m_populated_places_simple	`name>='København'`	137
ne_110m_populated_places_simple	`name>'København'`	136

Data Source	Predicate	Expected number of items
ne_110m_populated_places_simple	`name<='København'`	107
ne_110m_populated_places_simple	`name<'København'`	106
ne_110m_populated_places_simple	`name<>'København'`	242
ne_110m_populated_places_simple	`pop_other IS NOT NULL`	243
ne_110m_populated_places_simple	`pop_other IS NULL`	0
ne_110m_populated_places_simple	`pop_other=1038288`	1
ne_110m_populated_places_simple	`pop_other>=1038288`	123
ne_110m_populated_places_simple	`pop_other>1038288`	122
ne_110m_populated_places_simple	`pop_other<=1038288`	121
ne_110m_populated_places_simple	`pop_other<1038288`	120
ne_110m_populated_places_simple	`pop_other<>1038288`	242
ne_110m_populated_places_simple	`"date" IS NOT NULL`	3
ne_110m_populated_places_simple	`"date" IS NULL`	240
ne_110m_populated_places_simple	`"date"=DATE('2022-04-16')`	1
ne_110m_populated_places_simple	`"date">=DATE('2022-04-16')`	2
ne_110m_populated_places_simple	`"date">DATE('2022-04-16')`	1
ne_110m_populated_places_simple	`"date"<=DATE('2022-04-16')`	2
ne_110m_populated_places_simple	`"date"<DATE('2022-04-16')`	1
ne_110m_populated_places_simple	`"date"<>DATE('2022-04-16')`	2

Data Source	Predicate	Expected number of items
ne_110m_populated_places_simple	`start IS NOT NULL`	3
ne_110m_populated_places_simple	`start IS NULL`	240
ne_110m_populated_places_simple	`start=TIMESTAMP('2022-04-16T10:13:19Z')`	1
ne_110m_populated_places_simple	`start<=TIMESTAMP('2022-04-16T10:13:19Z')`	2
ne_110m_populated_places_simple	`start<TIMESTAMP('2022-04-16T10:13:19Z')`	1
ne_110m_populated_places_simple	`start>=TIMESTAMP('2022-04-16T10:13:19Z')`	2
ne_110m_populated_places_simple	`start>TIMESTAMP('2022-04-16T10:13:19Z')`	1
ne_110m_populated_places_simple	`start<>TIMESTAMP('2022-04-16T10:13:19Z')`	2
ne_110m_populated_places_simple	`boolean IS NOT NULL`	3
ne_110m_populated_places_simple	`boolean IS NULL`	240
ne_110m_populated_places_simple	`boolean=true`	2
ne_110m_populated_places_simple	`boolean=false`	1

A.3.6. Conformance Test 9

Test id:	/conf/basic-cql2/logical
Requirements:	/req/basic-cql2/cql2-filter
Test purpose:	Test filter expressions with AND, OR and NOT including sub-expressions

Test method:	Given: • One or more data sources. • The stored predicates for each data source. When: Evaluate each predicate in Combinations of predicates and expected results. For the data source 'ne_110m_populated_places_simple', evaluate the filter expression `(NOT ({p2}) AND {p1}) OR ({p3} and {p4}) or not ({p1} OR {p4})` for each combination of predicates `{p1}` to `{p4}` in Combinations of predicates and expected results. Then: • assert successful execution of the evaluation; • assert that the expected result is returned.

Table 8. Combinations of predicates and expected results

p1	p2	p3	p4	Expected number of items
`pop_other<>1038288`	`name<>'København'`	`pop_other IS NULL`	`name<'København'`	1
`pop_other<>1038288`	`name>'København'`	`name<='København'`	`boolean=true`	107
`start IS NULL`	`pop_other IS NOT NULL`	`pop_other IS NOT NULL`	`pop_other>1038288`	124
`pop_other<1038288`	`pop_other>1038288`	`pop_other IS NULL`	`start<TIMESTAMP('2022-04-16T10:13:19Z')`	121
`start=TIMESTAMP('2022-04-16T10:13:19Z')`	`pop_other<1038288`	`start=TIMESTAMP('2022-04-16T10:13:19Z')`	`name<>'København'`	2
`start<=TIMESTAMP('2022-04-16T10:13:19Z')`	`name<>'København'`	`boolean=true`	`name<'København'`	2
`pop_other=1038288`	`start IS NULL`	`start<>TIMESTAMP('2022-04-16T10:13:19Z')`	`boolean IS NOT NULL`	242
`start IS NULL`	`pop_other>1038288`	`start IS NOT NULL`	`name>'København'`	122
`pop_other<1038288`	`name<>'København'`	`name='København'`	`start<TIMESTAMP('2022-04-16T10:13:19Z')`	2
`start>=TIMESTAMP('2022-04-16T10:13:19Z')`	`name IS NOT NULL`	`start IS NULL`	`pop_other<1038288`	120

p1	p2	p3	p4	Expected number of items
name>='København'	start IS NOT NULL	boolean=true	start>=TIMESTAMP('2022-04-16T10:13:19Z')	137
start IS NOT NULL	name>='København'	start IS NOT NULL	name IS NOT NULL	3
name IS NULL	name<'København'	pop_other IS NOT NULL	boolean IS NOT NULL	243
start>=TIMESTAMP('2022-04-16T10:13:19Z')	name>'København'	pop_other=1038288	name<'København'	3
start<TIMESTAMP('2022-04-16T10:13:19Z')	name<='København'	boolean IS NULL	name>'København'	138
pop_other IS NOT NULL	start IS NULL	pop_other>=1038288	name>'København'	62
name='København'	start=TIMESTAMP('2022-04-16T10:13:19Z')	boolean=true	pop_other IS NULL	243
name>'København'	pop_other<1038288	pop_other>1038288	name<='København'	122
pop_other<>1038288	name='København'	name<='København'	start>TIMESTAMP('2022-04-16T10:13:19Z')	243
start<TIMESTAMP('2022-04-16T10:13:19Z')	start>=TIMESTAMP('2022-04-16T10:13:19Z')	pop_other=1038288	start IS NULL	3
name<>'København'	boolean=true	start=TIMESTAMP('2022-04-16T10:13:19Z')	start IS NULL	2
name IS NULL	start<>TIMESTAMP('2022-04-16T10:13:19Z')	start<TIMESTAMP('2022-04-16T10:13:19Z')	name IS NULL	243
start<>TIMESTAMP('2022-04-16T10:13:19Z')	name>'København'	start<=TIMESTAMP('2022-04-16T10:13:19Z')	name IS NOT NULL	3
name<>'København'	pop_other<>1038288	pop_other<1038288	start>=TIMESTAMP('2022-04-16T10:13:19Z')	2
boolean IS NULL	pop_other>1038288	boolean IS NOT NULL	pop_other IS NULL	122
pop_other=1038288	start IS NULL	start>TIMESTAMP('2022-04-16T10:13:19Z')	pop_other IS NOT NULL	2
pop_other<>1038288	start IS NULL	pop_other>1038288	boolean=true	2
start>TIMESTAMP('2022-04-16T10:13:19Z')	pop_other<1038288	name<='København'	pop_other=1038288	2

p1	p2	p3	p4	Expected number of items
start>=TIMESTAMP('2022-04-16T10:13:19Z')	start<=TIMESTAMP('2022-04-16T10:13:19Z')	name<='København'	name<>'København'	107
boolean=true	name IS NOT NULL	boolean IS NULL	pop_other=1038288	1
start=TIMESTAMP('2022-04-16T10:13:19Z')	pop_other=1038288	pop_other<1038288	name<>'København'	122
pop_other<>1038288	start<=TIMESTAMP('2022-04-16T10:13:19Z')	start IS NOT NULL	start=TIMESTAMP('2022-04-16T10:13:19Z')	3
name<>'København'	pop_other<>1038288	pop_other IS NOT NULL	name IS NOT NULL	243
name='København'	pop_other<1038288	start IS NOT NULL	pop_other<>1038288	3
name<'København'	start<>TIMESTAMP('2022-04-16T10:13:19Z')	start>TIMESTAMP('2022-04-16T10:13:19Z')	start=TIMESTAMP('2022-04-16T10:13:19Z')	2
boolean=true	pop_other<1038288	name IS NOT NULL	start<=TIMESTAMP('2022-04-16T10:13:19Z')	3
pop_other<=1038288	name<'København'	pop_other<1038288	pop_other<1038288	243
pop_other IS NULL	name<='København'	name='København'	start>TIMESTAMP('2022-04-16T10:13:19Z')	2
pop_other<1038288	name<>'København'	pop_other<>1038288	name<>'København'	243
start<=TIMESTAMP('2022-04-16T10:13:19Z')	pop_other IS NULL	start<TIMESTAMP('2022-04-16T10:13:19Z')	name IS NOT NULL	2
start<>TIMESTAMP('2022-04-16T10:13:19Z')	name='København'	boolean IS NULL	pop_other<>1038288	241
boolean=true	pop_other<=1038288	name<>'København'	pop_other IS NULL	2
name IS NOT NULL	pop_other<=1038288	start IS NOT NULL	boolean IS NOT NULL	124
pop_other<=1038288	pop_other<1038288	start>TIMESTAMP('2022-04-16T10:13:19Z')	pop_other>1038288	1
start IS NOT NULL	boolean IS NOT NULL	name>='København'	pop_other IS NOT NULL	137
start<>TIMESTAMP('2022-04-16T10:13:19Z')	start IS NOT NULL	pop_other>1038288	pop_other<1038288	1
pop_other<=1038288	name<='København'	boolean IS NULL	start IS NOT NULL	198
name>='København'	name>='København'	name<='København'	name>='København'	107

p1	p2	p3	p4	Expected number of items
boolean=true	start<TIMESTAMP('2022-04-16T10:13:19Z')	boolean IS NOT NULL	name<'København'	2
start>TIMESTAMP('2022-04-16T10:13:19Z')	start>=TIMESTAMP('2022-04-16T10:13:19Z')	pop_other IS NULL	pop_other<=1038288	1
pop_other<1038288	name='København'	start>=TIMESTAMP('2022-04-16T10:13:19Z')	name<'København'	181
pop_other<1038288	pop_other<=1038288	pop_other IS NULL	start IS NOT NULL	121
name>='København'	pop_other>=1038288	boolean=true	name IS NOT NULL	79
boolean IS NULL	name<>'København'	boolean IS NULL	pop_other IS NOT NULL	240
pop_other<1038288	start>=TIMESTAMP('2022-04-16T10:13:19Z')	name>'København'	pop_other<=1038288	199
name<='København'	start>TIMESTAMP('2022-04-16T10:13:19Z')	name<'København'	boolean IS NULL	106
pop_other IS NOT NULL	name<>'København'	pop_other<1038288	pop_other<=1038288	121
name>='København'	start IS NOT NULL	name>='København'	name IS NOT NULL	137
pop_other<1038288	start<TIMESTAMP('2022-04-16T10:13:19Z')	name IS NULL	pop_other>=1038288	1
pop_other>=1038288	name>'København'	boolean IS NOT NULL	start IS NOT NULL	184
start IS NOT NULL	name<>'København'	name<='København'	name IS NULL	241
name>='København'	pop_other<>1038288	start=TIMESTAMP('2022-04-16T10:13:19Z')	name<>'København'	2
boolean IS NOT NULL	pop_other<=1038288	pop_other=1038288	start=TIMESTAMP('2022-04-16T10:13:19Z')	1
name IS NOT NULL	start IS NOT NULL	start IS NOT NULL	name>='København'	241
pop_other=1038288	pop_other IS NOT NULL	start IS NOT NULL	name<>'København'	2
start=TIMESTAMP('2022-04-16T10:13:19Z')	start IS NULL	pop_other>1038288	pop_other<=1038288	1
name IS NULL	start IS NOT NULL	start=TIMESTAMP('2022-04-16T10:13:19Z')	name IS NOT NULL	1
boolean IS NOT NULL	name='København'	boolean IS NOT NULL	name IS NOT NULL	3

p1	p2	p3	p4	Expected number of items
pop_other<>1038288	pop_other<>1038288	pop_other=1038288	pop_other<=1038288	1
pop_other IS NULL	start<>TIMESTAMP('2022-04-16T10:13:19Z')	start>TIMESTAMP('2022-04-16T10:13:19Z')	boolean IS NOT NULL	241
start<TIMESTAMP('2022-04-16T10:13:19Z')	boolean IS NULL	start>TIMESTAMP('2022-04-16T10:13:19Z')	name<'København'	2
pop_other>1038288	pop_other<>1038288	start<>TIMESTAMP('2022-04-16T10:13:19Z')	name<>'København'	2
start>=TIMESTAMP('2022-04-16T10:13:19Z')	start=TIMESTAMP('2022-04-16T10:13:19Z')	pop_other=1038288	name IS NOT NULL	2
pop_other<=1038288	start IS NOT NULL	start<=TIMESTAMP('2022-04-16T10:13:19Z')	boolean IS NOT NULL	242
boolean=true	start>TIMESTAMP('2022-04-16T10:13:19Z')	pop_other<1038288	pop_other<>1038288	122
pop_other>=1038288	pop_other>1038288	boolean IS NULL	pop_other=1038288	121
name<'København'	pop_other>1038288	start=TIMESTAMP('2022-04-16T10:13:19Z')	boolean=true	44

A.4. Conformance Class "Advanced Comparison Operators"

Conformance Class	
http://www.opengis.net/spec/cql2/1.0/conf/advanced-comparison-operators	
Target type	Servers that evaluate filter expressions
Parameter	Filter Language: "CQL2 Text" or "CQL2 JSON"
Requirements class	Requirements Class "Advanced Comparison Operators"
Dependency	Basic CQL2

A.4.1. Conformance Test 10

Test id:	/conf/advanced-comparison-operators/like
Requirements:	/req/advanced-comparison-operators/like-predicate
Test purpose:	Test LIKE predicate

Test method:	Given: • One or more data sources, each with a list of queryables. When: For each queryable `{queryable}` of type String, evaluate the following filter expressions • `{queryable} LIKE '%'` • `{queryable} like '_%'` • `{queryable} like ''` • `{queryable} like '%%'` • `{queryable} like '\\%\\_'` Then: • assert successful execution of the evaluation; • assert that the two result sets for each queryable for the pattern expression `'_%'` and `''` have no item in common; • assert that the two result sets for each queryable for the pattern expression `'%'` and `'%%'` are identical; • store the valid predicates for each data source.

A.4.2. Conformance Test 11

Test id:	/conf/advanced-comparison-operators/between
Requirements:	/req/advanced-comparison-operators/between-predicate
Test purpose:	Test BETWEEN predicate
Test method:	Given: • One or more data sources, each with a list of queryables. When: for each queryable `{queryable}` of type Number or Integer, evaluate the following filter expressions • `{queryable} BETWEEN 0 AND 100` • `{queryable} between 100.0 and 1.0` Then: • assert successful execution of the evaluation; • store the valid predicates for each data source.

A.4.3. Conformance Test 12

Test id:	/conf/advanced-comparison-operators/in
Requirements:	/req/advanced-comparison-operators/in-predicate
Test purpose:	Test IN predicate
Test method:	Given: • One or more data sources, each with a list of queryables. When: • for each queryable `{queryable}` of type Number or Integer, evaluate the following filter expression `{queryable} IN (1, 2, 3)`; • for each queryable `{queryable}` of type String, evaluate the following filter expression `{queryable} in ('foo', 'bar')`; • for each queryable `{queryable}` of type Boolean, evaluate the following filter expression `{queryable} in (true)`; • for each queryable `{queryable}` of type Timestamp, evaluate the following filter expression `{queryable} in ('2022-04-14T14:52:56Z', '2022-04-14T15:52:56Z')`; • for each queryable `{queryable}` of type Date, evaluate the following filter expression `{queryable} in ('2022-04-14', '2022-04-15')`; Then: • assert successful execution of the evaluation; • store the valid predicates for each data source.

A.4.4. Conformance Test 13

Test id:	/conf/advanced-comparison-operators/test-data
Requirements:	all requirements
Test purpose:	Test predicates against the test dataset

Test method:	Given: • The implementation under test uses the test dataset. When: Evaluate each predicate in Predicates and expected results. Then: • assert successful execution of the evaluation; • assert that the expected result is returned; • store the valid predicates for each data source.

Table 9. Predicates and expected results

Data Source	Predicate	Expected number of items
ne_110m_populated_places_simple	`name LIKE 'B_r%'`	3
ne_110m_populated_places_simple	`name NOT LIKE 'B_r%'`	240
ne_110m_populated_places_simple	`pop_other between 1000000 and 3000000`	75
ne_110m_populated_places_simple	`pop_other not between 1000000 and 3000000`	168
ne_110m_populated_places_simple	`name IN ('Kiev','kobenhavn','Berlin','athens','foo')`	2
ne_110m_populated_places_simple	`name NOT IN ('Kiev','kobenhavn','Berlin','athens','foo')`	241
ne_110m_populated_places_simple	`pop_other in (1038288,1611692,3013258,3013257,3013259)`	3
ne_110m_populated_places_simple	`pop_other not in (1038288,1611692,3013258,3013257,3013259)`	240
ne_110m_populated_places_simple	`"date" in (DATE('2021-04-16'),DATE('2022-04-16'),DATE('2022-04-18'))`	2
ne_110m_populated_places_simple	`"date" not in (DATE('2021-04-16'),DATE('2022-04-16'),DATE('2022-04-18'))`	1
ne_110m_populated_places_simple	`start in (TIMESTAMP('2022-04-16T10:13:19Z'))`	1
ne_110m_populated_places_simple	`start not in (TIMESTAMP('2022-04-16T10:13:19Z'))`	2

Data Source	Predicate	Expected number of items
ne_110m_populated_places_simple	`boolean in (true)`	2
ne_110m_populated_places_simple	`boolean not in (false)`	2

A.4.5. Conformance Test 14

Test id:	/conf/advanced-comparison-operators/logical
Requirements:	n/a
Test purpose:	Test filter expressions with AND, OR and NOT including sub-expressions
Test method:	Given: • The stored predicates for each data source, including from the dependencies. When: For each data source, select at least 10 random combinations of four predicates (`{p1}` to `{p4}`) from the stored predicates and evaluate the filter expression `((NOT {p1} AND {p2}) OR ({p3} and NOT {p4}) or not ({p1} AND {p4}))`. Then: • assert successful execution of the evaluation.

A.5. Conformance Class "Case-insensitive Comparison"

Conformance Class	
http://www.opengis.net/spec/cql2/1.0/conf/case-insensitive-comparison	
Target type	Servers that evaluate filter expressions
Parameter	Filter Language: "CQL2 Text" or "CQL2 JSON"
Requirements class	Requirements Class "Case-insensitive Comparison"
Dependency	Basic CQL2
Conditional Dependency	Advanced Comparison Operators

A.5.1. Conformance Test 15

Test id:	/conf/case-insensitive-comparison/casei
Requirements:	/req/case-insensitive-comparison/casei-function

Test purpose:	Test the CASEI function in comparisons
Test method:	Given: • One or more data sources, each with a list of queryables. When: For each queryable `{queryable}` of type String, evaluate the following filter expressions • `CASEI({queryable}) = casei('foo')` • `CASEI({queryable}) <> casei('FOO')` Then: • assert successful execution of the evaluation; • assert that the two result sets for each queryable have no item in common; • store the valid predicates for each data source.

A.5.2. Conformance Test 16

Test id:	/conf/case-insensitive-comparison/casei-like
Requirements:	/req/case-insensitive-comparison/casei-function
Test purpose:	Test the CASEI function in LIKE predicates
Test method:	Given: • One or more data sources, each with a list of queryables. • The conformance class Advanced Comparison Operators passes. When: For each queryable `{queryable}` of type String, evaluate the following filter expressions • `CASEI({queryable}) LIKE casei('foo%')` • `CASEI({queryable}) LIKE casei('FOO%')` Then: • assert successful execution of the evaluation; • assert that the two result sets for each queryable are identical; • store the valid predicates for each data source.

A.5.3. Conformance Test 17

Test id:	/conf/case-insensitive-comparison/test-data
Requirements:	all requirements
Test purpose:	Test predicates against the test dataset
Test method:	Given: • The implementation under test uses the test dataset. When: Evaluate each predicate in Predicates and expected results, if the conditional dependency is met. Then: • assert successful execution of the evaluation; • assert that the expected result is returned; • store the valid predicates for each data source.

Table 10. Predicates and expected results

Dependency	Data Source	Predicate	Expected number of items
n/a	ne_110m_populated_places_simple	`CASEI(name)=casei('KIEV')`	1
n/a	ne_110m_populated_places_simple	`CASEI(name)=casei('kiev')`	1
n/a	ne_110m_populated_places_simple	`CASEI(name)=casei('Kiev')`	1
n/a	ne_110m_populated_places_simple	`CASEI(name)=casei('København')`	1
n/a	ne_110m_populated_places_simple	`CASEI(name)=casei('københavn')`	1
n/a	ne_110m_populated_places_simple	`CASEI(name)=casei('KØBENHAVN')`	1
Advanced Comparison Operators	ne_110m_populated_places_simple	`CASEI(name) LIKE casei('B_r%')`	3
Advanced Comparison Operators	ne_110m_populated_places_simple	`CASEI(name) LIKE casei('b_r%')`	3
Advanced Comparison Operators	ne_110m_populated_places_simple	`CASEI(name) LIKE casei('B_R%')`	3

Dependency	Data Source	Predicate	Expected number of items
Advanced Comparison Operators	ne_110m_populated_places_simple	`CASEI(name) IN (casei('Kiev'), casei('kobenhavn'), casei('Berlin'), casei('athens'), casei('foo'))`	3

A.5.4. Conformance Test 18

Test id:	/conf/case-insensitive-comparison/logical
Requirements:	n/a
Test purpose:	Test filter expressions with AND, OR and NOT including sub-expressions
Test method:	Given: • The stored predicates for each data source, including from the dependencies. When: For each data source, select at least 10 random combinations of four predicates (`{p1}` to `{p4}`) from the stored predicates and evaluate the filter expression `((NOT {p1} AND {p2}) OR ({p3} and NOT {p4}) or not ({p1} AND {p4}))`. Then: • assert successful execution of the evaluation.

A.6. Conformance Class "Accent-insensitive Comparison"

Conformance Class	
http://www.opengis.net/spec/cql2/1.0/conf/accent-insensitive-comparison	
Target type	Servers that evaluate filter expressions
Parameter	Filter Language: "CQL2 Text" or "CQL2 JSON"
Requirements class	Requirements Class "Accent-insensitive Comparison"
Dependency	Basic CQL2
Conditional Dependency	Advanced Comparison Operators
Conditional Dependency	Case-insensitive Comparison

A.6.1. Conformance Test 19

Test id:	/conf/accent-insensitive-comparison/accenti
Requirements:	/req/accent-insensitive-comparison/accenti-function
Test purpose:	Test the ACCENTI function in comparisons
Test method:	Given: • One or more data sources, each with a list of queryables. When: For each queryable `{queryable}` of type String, evaluate the following filter expressions • `ACCENTI({queryable}) = accenti('äöüéáí')` • `ACCENTI({queryable}) <> accenti('aoueai')` Then: • assert successful execution of the evaluation; • assert that the two result sets for each queryable have no item in common; • store the valid predicates for each data source.

A.6.2. Conformance Test 20

Test id:	/conf/accent-insensitive-comparison/accenti-like
Requirements:	/req/accent-insensitive-comparison/accenti-function
Test purpose:	Test the ACCENTI function in LIKE predicates

Test method:	Given: • One or more data sources, each with a list of queryables. • The conformance class Advanced Comparison Operators passes. When: For each queryable `{queryable}` of type String, evaluate the following filter expressions • `ACCENTI({queryable}) LIKE accenti('Ä%')` • `ACCENTI({queryable}) LIKE accenti('A%')` Then: • assert successful execution of the evaluation; • assert that the two result sets for each queryable are identical; • store the valid predicates for each data source.

A.6.3. Conformance Test 21

Test id:	/conf/accent-insensitive-comparison/accenti-casei
Requirements:	/req/accent-insensitive-comparison/accenti-function
Test purpose:	Test the ACCENTI function with the CASEI function
Test method:	Given: • One or more data sources, each with a list of queryables. • The conformance class Case-insensitive Comparison passes. When: For each queryable `{queryable}` of type String, evaluate the following filter expressions • `ACCENTI(CASEI({queryable})) = accenti(casei('ÄÉ'))` • `ACCENTI(CASEI({queryable})) = accenti(casei('ae'))` Then: • assert successful execution of the evaluation; • assert that the two result sets for each queryable are identical; • store the valid predicates for each data source.

A.6.4. Conformance Test 22

Test id:	/conf/accent-insensitive-comparison/accenti-casei-like
Requirements:	/req/accent-insensitive-comparison/accenti-function
Test purpose:	Test the ACCENTI function with the CASEI function in LIKE predicates
Test method:	Given: • One or more data sources, each with a list of queryables. • The conformance class Case-insensitive Comparison passes. • The conformance class Advanced Comparison Operators passes. When: For each queryable `{queryable}` of type String, evaluate the following filter expressions • `ACCENTI(CASEI({queryable})) LIKE accenti(casei('Ä%'))` • `ACCENTI(CASEI({queryable})) LIKE accenti(casei('a%'))` Then: • assert successful execution of the evaluation; • assert that the two result sets for each queryable are identical; • store the valid predicates for each data source.

A.6.5. Conformance Test 23

Test id:	/conf/accent-insensitive-comparison/test-data
Requirements:	all requirements
Test purpose:	Test predicates against the test dataset
Test method:	Given: • The implementation under test uses the test dataset. When: Evaluate each predicate in Predicates and expected results, if the conditional dependency is met. Then: • assert successful execution of the evaluation; • assert that the expected result is returned; • store the valid predicates for each data source.

Table 11. Predicates and expected results

Dependency	Data Source	Predicate	Expected number of items
n/a	ne_110m_populated_places_simple	`ACCENTI(name)=accenti('Chi□inău')`	1
n/a	ne_110m_populated_places_simple	`ACCENTI(name)=accenti('Chisinau')`	1
n/a	ne_110m_populated_places_simple	`ACCENTI(name)=accenti('Kiev')`	1
Case-insensitive Comparison	ne_110m_populated_places_simple	`ACCENTI(CASEI(name))=accenti(casei('chi□inău'))`	1
Case-insensitive Comparison	ne_110m_populated_places_simple	`ACCENTI(CASEI(name))=accenti(casei('chisinau'))`	1
Case-insensitive Comparison	ne_110m_populated_places_simple	`ACCENTI(CASEI(name))=accenti(casei('CHISINAU'))`	1
Case-insensitive Comparison	ne_110m_populated_places_simple	`ACCENTI(CASEI(name))=accenti(casei('CHI□INĂU'))`	1
Advanced Comparison Operators	ne_110m_populated_places_simple	`ACCENTI(name) LIKE accenti('Ch%')`	2
Case-insensitive Comparison, Advanced Comparison Operators	ne_110m_populated_places_simple	`ACCENTI(CASEI(name)) LIKE accenti(casei('Chi□%'))`	2
Case-insensitive Comparison, Advanced Comparison Operators	ne_110m_populated_places_simple	`ACCENTI(CASEI(name)) LIKE accenti(casei('cHis%'))`	2
Case-insensitive Comparison, Advanced Comparison Operators	ne_110m_populated_places_simple	`ACCENTI(CASEI(name)) IN (accenti(casei('Kiev')), accenti(casei('chi□inău')), accenti(casei('Berlin')), accenti(casei('athens')), accenti(casei('foo')))`	4

A.6.6. Conformance Test 24

Test id:	/conf/accent-insensitive-comparison/logical
Requirements:	n/a
Test purpose:	Test filter expressions with AND, OR and NOT including sub-expressions

Test method:	Given: • The stored predicates for each data source, including from the dependencies. When: For each data source, select at least 10 random combinations of four predicates (`{p1}` to `{p4}`) from the stored predicates and evaluate the filter expression `((NOT {p1} AND {p2}) OR ({p3} and NOT {p4}) or not ({p1} AND {p4}))`. Then: • assert successful execution of the evaluation.

A.7. Conformance Class "Basic Spatial Functions"

Conformance Class	
http://www.opengis.net/spec/cql2/1.0/conf/basic-spatial-functions	
Target type	Servers that evaluate filter expressions
Parameter	Filter Language: "CQL2 Text" or "CQL2 JSON"
Requirements class	Requirements Class "Basic Spatial Functions"
Dependency	Basic CQL2

The term "geometry data type" is used for the following data types: Point, MultiPoint, LineString, MultiLineString, Polygon, MultiPolygon, Geometry, or GeometryCollection.

A.7.1. Conformance Test 25

Test id:	/conf/basic-spatial-functions/s_intersects
Requirements:	/req/basic-spatial-functions/spatial-predicate, /req/basic-spatial-functions/spatial-functions, /req/basic-spatial-functions/spatial-data-types
Test purpose:	Test the S_INTERSECTS spatial comparison function with points and bounding boxes.

Test method:	Given: • One or more data sources, each with a list of queryables. • At least one queryable has a geometry data type. When: For each queryable `{queryable}` with a geometry data type, evaluate the following filter expressions • `S_INTERSECTS({queryable},BBOX(-180,-90,180,90))` • `S_INTERSECTS({queryable},POINT(7.02 49.92))` • `S_INTERSECTS({queryable},POINT(90 180))` • `S_INTERSECTS({queryable},BBOX(-180,-90,-90,90)) AND S_INTERSECTS({queryable},BBOX(90,-90,180,90))` Then: • assert successful execution of the evaluation for the first two filter expressions; • assert unsuccessful execution of the evaluation for the third filter expressions (invalid coordinate); • store the valid predicates for each data source.

A.7.2. Conformance Test 26

Test id:	/conf/basic-spatial-functions/test-data
Requirements:	all requirements
Test purpose:	Test predicates against the test dataset
Test method:	Given: • The implementation under test uses the test dataset. When: Evaluate each predicate in Predicates and expected results. Then: • assert successful execution of the evaluation; • assert that the expected result is returned; • store the valid predicates for each data source.

Table 12. Predicates and expected results

Data Source	Predicate	Expected number of items
ne_110m_admin_0_countries	`S_INTERSECTS(geom,BBOX(0,40,10,50))`	8
ne_110m_admin_0_countries	`S_INTERSECTS(geom,BBOX(150,-90,-150,90))`	10
ne_110m_admin_0_countries	`S_INTERSECTS(geom,POINT(7.02 49.92))`	1
ne_110m_admin_0_countries	`S_INTERSECTS(geom,BBOX(0,40,10,50)) and S_INTERSECTS(geom,BBOX(5,50,10,60))`	3
ne_110m_admin_0_countries	`S_INTERSECTS(geom,BBOX(0,40,10,50)) and not S_INTERSECTS(geom,BBOX(5,50,10,60))`	5
ne_110m_admin_0_countries	`S_INTERSECTS(geom,BBOX(0,40,10,50)) or S_INTERSECTS(geom,BBOX(-90,40,-60,50))`	10
ne_110m_populated_places_simple	`S_INTERSECTS(geom,BBOX(0,40,10,50))`	7
ne_110m_rivers_lake_centerlines	`S_INTERSECTS(geom,BBOX(-180,-90,0,90))`	4

A.7.3. Conformance Test 27

Test id:	/conf/basic-spatial-functions/logical
Requirements:	n/a
Test purpose:	Test filter expressions with AND, OR and NOT including sub-expressions
Test method:	Given: • The stored predicates for each data source, including from the dependencies. When: For each data source, select at least 10 random combinations of four predicates (`{p1}` to `{p4}`) from the stored predicates and evaluate the filter expression `((NOT {p1} AND {p2}) OR ({p3} and NOT {p4}) or not ({p1} AND {p4}))`. Then: • assert successful execution of the evaluation.

A.8. Conformance Class "Basic Spatial Functions with additional Spatial Literals"

Conformance Class	
http://www.opengis.net/spec/cql2/1.0/conf/basic-spatial-functions-plus	
Target type	Servers that evaluate filter expressions
Parameter	Filter Language: "CQL2 Text" or "CQL2 JSON"
Requirements class	Requirements Class "Basic Spatial Functions with additional Spatial Literals"
Dependency	Basic Spatial Functions

The term "geometry data type" is used for the following data types: Point, MultiPoint, LineString, MultiLineString, Polygon, MultiPolygon, or GeometryCollection.

A.8.1. Conformance Test 28

Test id:	/conf/basic-spatial-functions-plus/s_intersects
Requirements:	/req/basic-spatial-functions-plus/spatial-predicate, /req/basic-spatial-functions-plus/spatial-functions, /req/basic-spatial-functions-plus/spatial-data-types
Test purpose:	Test the S_INTERSECTS spatial comparison function with points, multi-points, line strings, multi-line string, polygons, multi-polygons, geometry collections and bounding boxes.

Test method:	Given: • One or more data sources, each with a list of queryables. • At least one queryable has a geometry data type. When: For each queryable `{queryable}` with a geometry data type, evaluate the following filter expressions • `S_INTERSECTS({queryable},MULTIPOINT(7.02 49.92, 90 180))` • `S_INTERSECTS({queryable},LINESTRING(-180 -45, 0 -45))` • `S_INTERSECTS({queryable},MULTILINESTRING((-180 -45, 0 -45), (0 45, 180 45)))` • `S_INTERSECTS({queryable},POLYGON((-180 -90, -90 -90, -90 90, -180 90, -180 -90), (-120 -50, -100 -50, -100 -40, -120 -40, -120 -50)))` • `S_INTERSECTS({queryable},MULTIPOLYGON(((-180 -90, -90 -90, -90 90, -180 90, -180 -90), (-120 -50, -100 -50, -100 -40, -120 -40, -120 -50)),((0 0, 10 0, 10 10, 0 10, 0 0))))` • `S_INTERSECTS({queryable},GEOMETRYCOLLECTION(POINT(7.02 49.92), POLYGON((0 0, 10 0, 10 10, 0 10, 0 0))))` Then: • assert successful execution of the evaluation for all filter expressions except the first; • assert unsuccessful execution of the evaluation for the first filter expressions (invalid coordinate); • store the valid predicates for each data source.

A.8.2. Conformance Test 29

Test id:	/conf/basic-spatial-functions-plus/test-data
Requirements:	all requirements
Test purpose:	Test predicates against the test dataset

Test method:	Given: • The implementation under test uses the test dataset. When: Evaluate each predicate in Predicates and expected results. Then: • assert successful execution of the evaluation; • assert that the expected result is returned.

Table 13. Predicates and expected results

Data Source	Predicate	Expected number of items
ne_110m_admin_0_countries	`S_INTERSECTS(geom,LINESTRING(-180 -45, 0 -45))`	2
ne_110m_admin_0_countries	`S_INTERSECTS(geom,MULTILINESTRING((-180 -45, 0 -45), (0 45, 180 45)))`	14
ne_110m_admin_0_countries	`S_INTERSECTS(geom,POLYGON((-180 -90, -90 -90, -90 90, -180 90, -180 -90), (-120 -50, -100 -50, -100 -40, -120 -40, -120 -50)))`	8
ne_110m_admin_0_countries	`S_INTERSECTS(geom,MULTIPOLYGON(((-180 -90, -90 -90, -90 90, -180 90, -180 -90), (-120 -50, -100 -50, -100 -40, -120 -40, -120 -50)),((0 0, 10 0, 10 10, 0 10, 0 0))))`	15
ne_110m_admin_0_countries	`S_INTERSECTS(geom,GEOMETRYCOLLECTION(POINT(7.02 49.92), POLYGON((0 0, 10 0, 10 10, 0 10, 0 0))))`	8
ne_110m_admin_0_countries	`S_INTERSECTS(geom,POLYGON((-180 -90, -90 -90, -90 90, -180 90, -180 -90), (-120 -50, -100 -50, -100 -40, -120 -40, -120 -50))) or S_INTERSECTS(geom,POLYGON((0 0, 10 0, 10 10, 0 10, 0 0)))`	15
ne_110m_admin_0_countries	`S_INTERSECTS(geom,POLYGON((-180 -90, -90 -90, -90 90, -180 90, -180 -90), (-120 -50, -100 -50, -100 -40, -120 -40, -120 -50))) and not S_INTERSECTS(geom,POLYGON((-130 0, 0 0, 0 50, -130 50, -130 0)))`	3

A.9. Conformance Class "Spatial Functions"

Conformance Class	
http://www.opengis.net/spec/cql2/1.0/conf/spatial-functions	
Target type	Servers that evaluate filter expressions
Parameter	Filter Language: "CQL2 Text" or "CQL2 JSON"
Requirements class	Requirements Class "Spatial Functions"
Dependency	Basic Spatial Functions with additional Spatial Literals

A.9.1. Conformance Test 30

Test id:	/conf/spatial-functions/s_intersects
Requirements:	/req/spatial-functions/spatial-functions, /req/spatial-functions/spatial-data-types
Test purpose:	Test the S_INTERSECTS spatial function
Test method:	Given: • One or more data sources, each with a list of queryables. • At least one queryable has a geometry data type. When: For each queryable `{queryable}` with a geometry data type, evaluate the following filter expressions • `S_INTERSECTS({queryable},BBOX(-180,-90,180,90))` • `S_INTERSECTS({queryable},POLYGON((-180 -90,180 -90,180 90,-180 90,-180 -90)))` • `S_INTERSECTS({queryable},LINESTRING(7 50, 10 51))` • `S_INTERSECTS({queryable},POINT(7.02 49.92))` • `S_INTERSECTS({queryable},POINT(90 180))` Then: • assert successful execution of the evaluation for the first four filter expressions; • assert unsuccessful execution of the evaluation for the fifth filter expressions (invalid coordinate); • assert that the two result sets of the first two filter expressions for each queryable are identical; • store the valid predicates for each data source.

A.9.2. Conformance Test 31

Test id:	/conf/spatial-functions/s_disjoint
Requirements:	/req/spatial-functions/spatial-functions, /req/spatial-functions/spatial-data-types
Test purpose:	Test the S_DISJOINT spatial function
Test method:	Given: • One or more data sources, each with a list of queryables. When: for each queryable `{queryable}` with a geometry data type, evaluate the following filter expressions • `S_DISJOINT({queryable},BBOX(-180,-90,180,90))` • `S_DISJOINT({queryable},POLYGON((-180 -90,180 -90,180 90,-180 90,-180 -90)))` • `S_DISJOINT({queryable},LINESTRING(7 50,10 51))` • `S_DISJOINT({queryable},POINT(7.02 49.92))` • `S_DISJOINT({queryable},POINT(90 180))` Then: • assert successful execution of the evaluation for the first four filter expressions; • assert unsuccessful execution of the evaluation for the fifth filter expressions (invalid coordinate); • assert that the two result sets of the first two filter expressions for each queryable are empty; • assert that the results sets of the third and fourth filter expressions for each queryable do not have an item in common with the corresponding S_INTERSECTS expression; • store the valid predicates for each data source.

A.9.3. Conformance Test 32

Test id:	/conf/spatial-functions/s_equals
Requirements:	/req/spatial-functions/spatial-functions, /req/spatial-functions/spatial-data-types
Test purpose:	Test the S_EQUALS spatial function

Test method:	Given: • One or more data sources, each with a list of queryables. When: for each queryable `{queryable}` with a geometry data type, evaluate the following filter expressions • `S_EQUALS({queryable},POLYGON((-180 -90,180 -90,180 90,-180 90,-180 -90)))` • `S_EQUALS({queryable},LINESTRING(7 50,10 51))` • `S_EQUALS({queryable},POINT(7.02 49.92))` Then: • assert successful execution of the evaluation; • assert that the two result sets of the first two filter expressions for each queryable are identical; • store the valid predicates for each data source.

A.9.4. Conformance Test 33

Test id:	/conf/spatial-functions/s_touches
Requirements:	/req/spatial-functions/spatial-functions, /req/spatial-functions/spatial-data-types
Test purpose:	Test the S_TOUCHES spatial function
Test method:	Given: • One or more data sources, each with a list of queryables. When: for each queryable `{queryable}` with a geometry data type, evaluate the following filter expressions • `S_TOUCHES({queryable},BBOX(-180,-90,180,90))` • `S_TOUCHES({queryable},POLYGON((-180 -90,180 -90,180 90,-180 90,-180 -90)))` • `S_TOUCHES({queryable},LINESTRING(7 50,10 51))` Then: • assert successful execution of the evaluation; • store the valid predicates for each data source.

A.9.5. Conformance Test 34

Test id:	/conf/spatial-functions/s_crosses
Requirements:	/req/spatial-functions/spatial-functions, /req/spatial-functions/spatial-data-types
Test purpose:	Test the S_CROSSES spatial function
Test method:	Given: • One or more data sources, each with a list of queryables. When: for each queryable `{queryable}` of type Point, MultiPoint, LineString or MultiLineString, evaluate the following filter expressions • `S_CROSSES({queryable},BBOX(-180,-90,180,90))` • `S_CROSSES({queryable},POLYGON((-180 -90,180 -90,180 90,-180 90,-180 -90)))` • `S_CROSSES({queryable},LINESTRING(7 50,10 51))` Then: • assert successful execution of the evaluation; • store the valid predicates for each data source.

A.9.6. Conformance Test 35

Test id:	/conf/spatial-functions/s_within
Requirements:	/req/spatial-functions/spatial-functions, /req/spatial-functions/spatial-data-types
Test purpose:	Test the S_WITHIN spatial function

Test method:	Given: • One or more data sources, each with a list of queryables. When: for each queryable `{queryable}` with a geometry data type, evaluate the following filter expressions • `S_WITHIN({queryable},BBOX(-180,-90,180,90))` • `S_WITHIN({queryable},POLYGON((-180 -90,180 -90,180 90,-180 90,-180 -90)))` • `S_WITHIN({queryable},LINESTRING(7 50,10 51))` • `S_WITHIN({queryable},MULTIPOINT(7 50,10 51))` Then: • assert successful execution of the evaluation; • assert that the two result sets of the first two filter expressions for each queryable are identical; • store the valid predicates for each data source.

A.9.7. Conformance Test 36

Test id:	/conf/spatial-functions/s_contains
Requirements:	/req/spatial-functions/spatial-functions, /req/spatial-functions/spatial-data-types
Test purpose:	Test the S_CONTAINS spatial function

Test method:	Given: • One or more data sources, each with a list of queryables. When: for each queryable `{queryable}` with a geometry data type, evaluate the following filter expressions • `S_CONTAINS({queryable},BBOX(-180,-90,180,90))` • `S_CONTAINS({queryable},POLYGON((-180 -90,180 -90,180 90,-180 90,-180 -90)))` • `S_CONTAINS({queryable},LINESTRING(7 50,10 51))` • `S_CONTAINS({queryable},MULTIPOINT(7 50,10 51))` Then: • assert successful execution of the evaluation; • assert that the two result sets of the first two filter expressions for each queryable are identical; • assert that the results sets for each queryable do not have an item in common with the corresponding S_WITHIN expression; • store the valid predicates for each data source.

A.9.8. Conformance Test 37

Test id:	/conf/spatial-functions/s_overlaps
Requirements:	/req/spatial-functions/spatial-functions, /req/spatial-functions/spatial-data-types
Test purpose:	Test the S_OVERLAPS spatial function

Test method:	Given: • One or more data sources, each with a list of queryables. When: • For each queryable `{queryable}` of type Point or MultiPoint, evaluate the filter expression `S_OVERLAPS({queryable},MULTIPOINT(7 50,10 51))` • For each queryable `{queryable}` of type LineString or MultiLineString, evaluate the filter expression `S_OVERLAPS({queryable},LINESTRING(7 50,10 51))` • For each queryable `{queryable}` of type Polygon or MultiPolygon, evaluate the filter expression `S_OVERLAPS({queryable},POLYGON((-180 -90,180 -90,180 90,-180 90,-180 -90)))` Then: • assert successful execution of the evaluation; • store the valid predicates for each data source.

A.9.9. Conformance Test 38

Test id:	/conf/spatial-functions/test-data
Requirements:	all requirements
Test purpose:	Test predicates against the test dataset
Test method:	Given: • The implementation under test uses the test dataset. When: Evaluate each predicate in Predicates and expected results. Then: • assert successful execution of the evaluation; • assert that the expected result is returned; • store the valid predicates for each data source.

Table 14. Predicates and expected results

Data Source	Predicate	Expected number of items
ne_110m_admin_0_countries	`S_INTERSECTS(geom,POLYGON((0 40,10 40,10 50,0 50,0 40)))`	8
ne_110m_admin_0_countries	`S_INTERSECTS(geom,LINESTRING(0 40,10 50))`	4

Data Source	Predicate	Expected number of items
ne_110m_populated_places_simple	`S_INTERSECTS(geom,POLYGON((0 40,10 40,10 50,0 50,0 40)))`	7
ne_110m_rivers_lake_centerlines	`S_INTERSECTS(geom,LINESTRING(-60 -90,-60 90))`	2
ne_110m_admin_0_countries	`S_DISJOINT(geom,BBOX(0,40,10,50))`	169
ne_110m_admin_0_countries	`S_DISJOINT(geom,POLYGON((0 40,10 40,10 50,0 50,0 40)))`	169
ne_110m_admin_0_countries	`S_DISJOINT(geom,LINESTRING(0 40,10 50))`	173
ne_110m_admin_0_countries	`S_DISJOINT(geom,POINT(7.02 49.92))`	176
ne_110m_populated_places_simple	`S_DISJOINT(geom,BBOX(0,40,10,50))`	236
ne_110m_populated_places_simple	`S_DISJOINT(geom,POLYGON((0 40,10 40,10 50,0 50,0 40)))`	236
ne_110m_rivers_lake_centerlines	`S_DISJOINT(geom,BBOX(-180,-90,0,90))`	9
ne_110m_rivers_lake_centerlines	`S_DISJOINT(geom,LINESTRING(-60 -90,-60 90))`	11
ne_110m_populated_places_simple	`S_EQUALS(geom,POINT(6.130002849.6116604))`	1
ne_110m_admin_0_countries	`S_TOUCHES(geom,POLYGON((6.043073357781111 50.128051662794235,6.242751092156993 49.90222565367873,6.186320428094177 49.463802802114515,5.897759230176348 49.44266714130711,5.674051954784829 49.529483547557504,5.782417433300907 50.09032786722122,6.043073357781111 50.128051662794235)))`	3
ne_110m_admin_0_countries	`S_TOUCHES(geom,POINT(6.043073357781111 50.128051662794235))`	3
ne_110m_admin_0_countries	`S_TOUCHES(geom,POINT(6.242751092156993 49.90222565367873))`	2
ne_110m_admin_0_countries	`S_TOUCHES(geom,LINESTRING(6.043073357781111 50.128051662794235,6.242751092156993 49.90222565367873))`	3

Data Source	Predicate	Expected number of items
ne_110m_rivers_lake_centerlines	`S_CROSSES(geom,BBOX(0,40,10,50))`	1
ne_110m_rivers_lake_centerlines	`S_CROSSES(geom,LINESTRING(-60 -90,-60 90))`	2
ne_110m_admin_0_countries	`S_WITHIN(geom,BBOX(-180,-90,0,90))`	44
ne_110m_populated_places_simple	`S_WITHIN(geom,BBOX(-180,-90,0,90))`	74
ne_110m_rivers_lake_centerlines	`S_WITHIN(geom,BBOX(-180,-90,0,90))`	4
ne_110m_admin_0_countries	`S_CONTAINS(geom,BBOX(7,50,8,51))`	1
ne_110m_admin_0_countries	`S_CONTAINS(geom,LINESTRING(7 50,8 51))`	1
ne_110m_admin_0_countries	`S_CONTAINS(geom,POINT(7.02 49.92))`	1
ne_110m_admin_0_countries	`S_OVERLAPS(geom,BBOX(-180,-90,0,90))`	11

A.9.10. Conformance Test 39

Test id:	/conf/spatial-functions/logical
Requirements:	n/a
Test purpose:	Test filter expressions with AND, OR and NOT including sub-expressions
Test method:	Given: • The stored predicates for each data source, including from the dependencies. When: For each data source, select at least 10 random combinations of four predicates (`{p1}` to `{p4}`) from the stored predicates and evaluate the filter expression `((NOT {p1} AND {p2}) OR ({p3} and NOT {p4}) or not ({p1} AND {p4}))`. Then: • assert successful execution of the evaluation.

A.10. Conformance Class "Temporal Functions"

Conformance Class

http://www.opengis.net/spec/cql2/1.0/conf/temporal-functions	
Target type	Servers that evaluate filter expressions
Parameter	Filter Language: "CQL2 Text" or "CQL2 JSON"
Requirements class	Requirements Class "Temporal Functions"
Dependency	Basic CQL2

The term "temporal data type" is used for the following data types: Timestamp, Date, or Interval.

A.10.1. Conformance Test 40

Test id:	/conf/temporal-functions/temporal-functions-1
Requirements:	/req/temporal-functions/temporal-predicates, /req/temporal-functions/temporal-functions
Test purpose:	Test the T_AFTER, T_BEFORE, T_DISJOINT, T_EQUALS, T_INTERSECTS temporal comparison functions.

Test method: Given:

- One or more data sources, each with a list of queryables with at least one queryable of type Timestamp or Date.

When:

For each queryable `{queryable}` of data type Timestamp, evaluate the following filter expressions

- `T_AFTER({queryable},TIMESTAMP('2022-04-24T07:59:57Z'))`
- `T_AFTER({queryable},INTERVAL('2021-01-01T00:00:00Z','2021-12-31T23:59:59Z'))`
- `T_BEFORE({queryable},TIMESTAMP('2022-04-24T07:59:57Z'))`
- `T_BEFORE({queryable},INTERVAL('2021-01-01T00:00:00Z','2021-12-31T23:59:59Z'))`
- `T_DISJOINT({queryable},TIMESTAMP('2022-04-24T07:59:57Z'))`
- `T_DISJOINT({queryable},INTERVAL('2021-01-01T00:00:00Z','2021-12-31T23:59:59Z'))`
- `T_EQUALS({queryable},TIMESTAMP('2022-04-24T07:59:57Z'))`
- `T_EQUALS({queryable},INTERVAL('2021-01-01T00:00:00Z','2021-12-31T23:59:59Z'))`
- `T_INTERSECTS({queryable},TIMESTAMP('2022-04-24T07:59:57Z'))`
- `T_INTERSECTS({queryable},INTERVAL('2021-01-01T00:00:00Z','2021-12-31T23:59:59Z'))`

For each queryable `{queryable}` of data type Date, evaluate the following filter expressions

- `T_AFTER({queryable},DATE('2022-04-24'))`
- `T_AFTER({queryable},INTERVAL('2021-01-01','2021-12-31'))`
- `T_BEFORE({queryable},DATE('2022-04-24'))`
- `T_BEFORE({queryable},INTERVAL('2021-01-01','2021-12-31'))`
- `T_DISJOINT({queryable},DATE('2022-04-24'))`
- `T_DISJOINT({queryable},INTERVAL('2021-01-01','2021-12-31'))`
- `T_EQUALS({queryable},DATE('2022-04-24'))`
- `T_EQUALS({queryable},INTERVAL('2021-01-01','2021-12-31'))`
- `T_INTERSECTS({queryable},DATE('2022-04-24'))`
- `T_INTERSECTS({queryable},INTERVAL('2021-01-01','2021-12-31'))`

Then:

A.10.2. Conformance Test 41

Test id:	/conf/temporal-functions/temporal-functions-2
Requirements:	/req/temporal-functions/temporal-predicates, /req/temporal-functions/temporal-functions
Test purpose:	Test the temporal comparison functions with intervals

Test method: Given:

- One or more data sources, each with a list of queryables with at least two queryables of type Timestamp or Date.

When:

For each pair of queryables `{queryable2}` and `{queryable2}` of data type Timestamp, evaluate the following filter expressions

- `T_AFTER(INTERVAL({queryable1},{queryable2}),INTERVAL('2021-01-01T00:00:00Z','2021-12-31T23:59:59Z'))`
- `T_BEFORE(INTERVAL({queryable1},{queryable2}),INTERVAL('2021-01-01T00:00:00Z','2021-12-31T23:59:59Z'))`
- `T_DISJOINT(INTERVAL({queryable1},{queryable2}),INTERVAL('2021-01-01T00:00:00Z','2021-12-31T23:59:59Z'))`
- `T_EQUALS(INTERVAL({queryable1},{queryable2}),INTERVAL('2021-01-01T00:00:00Z','2021-12-31T23:59:59Z'))`
- `T_INTERSECTS(INTERVAL({queryable1},{queryable2}),INTERVAL('2021-01-01T00:00:00Z','2021-12-31T23:59:59Z'))`
- `T_CONTAINS(INTERVAL({queryable1},{queryable2}),INTERVAL('2021-01-01T00:00:00Z','2021-12-31T23:59:59Z'))`
- `T_DURING(INTERVAL({queryable1},{queryable2}),INTERVAL('2021-01-01T00:00:00Z','2021-12-31T23:59:59Z'))`
- `T_FINISHEDBY(INTERVAL({queryable1},{queryable2}),INTERVAL('2021-01-01T00:00:00Z','2021-12-31T23:59:59Z'))`
- `T_FINISHES(INTERVAL({queryable1},{queryable2}),INTERVAL('2021-01-01T00:00:00Z','2021-12-31T23:59:59Z'))`
- `T_MEETS(INTERVAL({queryable1},{queryable2}),INTERVAL('2021-01-01T00:00:00Z','2021-12-31T23:59:59Z'))`
- `T_METBY(INTERVAL({queryable1},{queryable2}),INTERVAL('2021-01-01T00:00:00Z','2021-12-31T23:59:59Z'))`
- `T_OVERLAPPEDBY(INTERVAL({queryable1},{queryable2}),INTERVAL('2021-01-01T00:00:00Z','2021-12-31T23:59:59Z'))`
- `T_OVERLAPS(INTERVAL({queryable1},{queryable2}),INTERVAL('2021-01-01T00:00:00Z','2021-12-31T23:59:59Z'))`
- `T_STARTEDBY(INTERVAL({queryable1},{queryable2}),INTERVAL('2021-01-01T00:00:00Z','2021-12-31T23:59:59Z'))`
- `T_STARTS(INTERVAL({queryable1},{queryable2}),INTERVAL('2021-01-01T00:00:00Z','2021-12-31T23:59:59Z'))`

For each pair of queryables `{queryable2}` and `{queryable2}` of data type Date, evaluate the following filter expressions

A.10.3. Conformance Test 42

Test id:	/conf/temporal-functions/test-data
Requirements:	all requirements
Test purpose:	Test predicates against the test dataset
Test method:	Given: • The implementation under test uses the test dataset. When: Evaluate each predicate in Predicates and expected results. Then: • assert successful execution of the evaluation; • assert that the expected result is returned; • store the valid predicates for each data source.

Table 15. Predicates and expected results

Data Source	Predicate	Expected number of items
ne_110m_populated_places_simple	`t_after("date",date('2022-04-16'))`	1
ne_110m_populated_places_simple	`t_before("date",date('2022-04-16'))`	1
ne_110m_populated_places_simple	`t_disjoint("date",date('2022-04-16'))`	2
ne_110m_populated_places_simple	`t_equals("date",date('2022-04-16'))`	1
ne_110m_populated_places_simple	`t_intersects("date",date('2022-04-16'))`	1
ne_110m_populated_places_simple	`t_after("date",interval('2022-01-01','2022-12-31'))`	1
ne_110m_populated_places_simple	`t_before("date",interval('2022-01-01','2022-12-31'))`	1
ne_110m_populated_places_simple	`t_disjoint("date",interval('2022-01-01','2022-12-31'))`	2
ne_110m_populated_places_simple	`t_equals("date",interval('2022-01-01','2022-12-31'))`	0
ne_110m_populated_places_simple	`t_equals("date",interval('2022-04-16','2022-04-16'))`	1

Data Source	Predicate	Expected number of items
ne_110m_populated_places_simple	`t_intersects("date",interval('2022-01-01','2022-12-31'))`	1
ne_110m_populated_places_simple	`t_after(start,timestamp('2022-04-16T10:13:19Z'))`	1
ne_110m_populated_places_simple	`t_before(start,timestamp('2022-04-16T10:13:19Z'))`	1
ne_110m_populated_places_simple	`t_disjoint(start,timestamp('2022-04-16T10:13:19Z'))`	2
ne_110m_populated_places_simple	`t_equals(start,timestamp('2022-04-16T10:13:19Z'))`	1
ne_110m_populated_places_simple	`t_intersects(start,timestamp('2022-04-16T10:13:19Z'))`	1
ne_110m_populated_places_simple	`t_after(start,interval('2022-01-01T00:00:00Z','2022-12-31T23:59:59Z'))`	0
ne_110m_populated_places_simple	`t_before(start,interval('2022-01-01T00:00:00Z','2022-12-31T23:59:59Z'))`	1
ne_110m_populated_places_simple	`t_disjoint(start,interval('2022-01-01T00:00:00Z','2022-12-31T23:59:59Z'))`	1
ne_110m_populated_places_simple	`t_equals(start,interval('2022-01-01T00:00:00Z','2022-12-31T23:59:59Z'))`	0
ne_110m_populated_places_simple	`t_intersects(start,interval('2022-01-01T00:00:00Z','2022-12-31T23:59:59Z'))`	2
ne_110m_populated_places_simple	`t_after(interval(start,end),interval('..','2022-04-16T10:13:19Z'))`	1
ne_110m_populated_places_simple	`t_before(interval(start,end),interval('2023-01-01T00:00:00Z','..'))`	2
ne_110m_populated_places_simple	`t_disjoint(interval(start,end),interval('2022-04-16T10:13:19Z','2022-04-16T10:15:09Z'))`	1
ne_110m_populated_places_simple	`t_equals(interval(start,end),interval('2021-04-16T10:15:59Z','2022-04-16T10:16:06Z'))`	1
ne_110m_populated_places_simple	`t_intersects(interval(start,end),interval('2022-04-16T10:13:19Z','2022-04-16T10:15:09Z'))`	2

Data Source	Predicate	Expected number of items
ne_110m_populated_places_simple	`T_CONTAINS(interval(start,end),interval('2022-04-16T10:13:19Z','2022-04-16T10:15:10Z'))`	1
ne_110m_populated_places_simple	`T_DURING(interval(start,end),interval('2022-01-01T00:00:00Z','2022-12-31T23:59:59Z'))`	1
ne_110m_populated_places_simple	`T_FINISHES(interval(start,end),interval('2020-04-16T10:13:19Z','2022-04-16T10:16:06Z'))`	1
ne_110m_populated_places_simple	`T_FINISHEDBY(interval(start,end),interval('2022-04-16T10:13:19Z','2022-04-16T10:16:06Z'))`	1
ne_110m_populated_places_simple	`T_MEETS(interval(start,end),interval('2022-04-16T10:13:19Z','2022-04-16T10:15:10Z'))`	0
ne_110m_populated_places_simple	`T_METBY(interval(start,end),interval('2022-04-16T10:13:19Z','2022-04-16T10:15:10Z'))`	1
ne_110m_populated_places_simple	`T_OVERLAPPEDBY(interval(start,end),interval('2020-04-16T10:13:19Z','2022-04-16T10:15:10Z'))`	2
ne_110m_populated_places_simple	`T_OVERLAPS(interval(start,end),interval('2022-04-16T10:13:19Z','2023-04-16T10:15:10Z'))`	1
ne_110m_populated_places_simple	`T_STARTEDBY(interval(start,end),interval('2022-04-16T10:13:19Z','2022-04-16T10:15:10Z'))`	1
ne_110m_populated_places_simple	`T_STARTS(interval(start,end),interval('2022-04-16T10:13:19Z','2022-04-16T10:15:10Z'))`	0

A.10.4. Conformance Test 43

Test id:	/conf/temporal-functions/logical
Requirements:	n/a
Test purpose:	Test filter expressions with AND, OR and NOT including sub-expressions

Test method:	Given: • The stored predicates for each data source, including from the dependencies. When: For each data source, select at least 10 random combinations of four predicates (`{p1}` to `{p4}`) from the stored predicates and evaluate the filter expression `((NOT {p1} AND {p2}) OR ({p3} and NOT {p4}) or not ({p1} AND {p4}))`. Then: • assert successful execution of the evaluation.

A.11. Conformance Class "Array Functions"

Conformance Class	
http://www.opengis.net/spec/cql2/1.0/conf/array-functions	
Target type	Servers that evaluate filter expressions
Parameter	Filter Language: "CQL2 Text" or "CQL2 JSON"
Requirements class	Requirements Class "Array Functions"
Dependency	Basic CQL2

A.11.1. Conformance Test 44

Test id:	/conf/array-functions/array-predicates
Requirements:	/req/array-functions/array-predicates
Test purpose:	Test the array comparison functions

Test method:	Given: • One or more data sources, each with a list of queryables. • At least one queryable has an array data type. When: For each queryable `{queryable}` with an array data type, evaluate the following filter expressions • `A_CONTAINS({queryable},("foo","bar"))` • `A_CONTAINEDBY({queryable},("foo","bar"))` • `A_EQUALS({queryable},("foo","bar"))` • `A_OVERLAPS({queryable},("foo","bar"))` Then: • assert successful execution of the evaluation; • store the valid predicates for each data source.

A.11.2. Conformance Test 45

Test id:	/conf/array-functions/logical
Requirements:	n/a
Test purpose:	Test filter expressions with AND, OR and NOT including sub-expressions
Test method:	Given: • The stored predicates for each data source, including from the dependencies. When: For each data source, select at least 10 random combinations of four predicates (`{p1}` to `{p4}`) from the stored predicates and evaluate the filter expression `((NOT {p1} AND {p2}) OR ({p3} and NOT {p4}) or not ({p1} AND {p4}))`. Then: • assert successful execution of the evaluation.

A.12. Conformance Class "Property-Property Comparisons"

Conformance Class

http://www.opengis.net/spec/cql2/1.0/conf/property-property	
Target type	Servers that evaluate filter expressions
Parameter	Filter Language: "CQL2 Text" or "CQL2 JSON"
Requirements class	Requirements Class "Property-Property Comparisons"
Dependency	Basic CQL2
Conditional Dependency	Advanced Comparison Operators
Conditional Dependency	Basic Spatial Functions
Conditional Dependency	Spatial Functions
Conditional Dependency	Temporal Functions

A.12.1. Conformance Test 46

Test id:	/conf/property-property/comparison-value-property
Requirements:	/req/property-property/withdraw-permissions
Test purpose:	Test comparison predicates with properties on the right-hand side and values on the left-hand side

Test method:	Given: • One or more data sources, each with a list of queryables. When: For each queryable `{queryable}` of one of the data types String, Boolean, Number, Integer, Timestamp or Date, evaluate the following filter expressions • `{value} = {queryable}` • `{value} <> {queryable}` • `{value} > {queryable}` • `{value} < {queryable}` • `{value} >= {queryable}` • `{value} <= {queryable}` where `{value}` depends on the data type: • String: `'foo'` • Boolean: `true` • Number: `3.14` • Integer: `1` • Timestamp: `TIMESTAMP('2022-04-14T14:48:46Z')` • Date: `DATE('2022-04-14')` Then: • assert successful execution of the evaluation; • assert that the two result sets for each queryable for the operators `=` and `<>` have no item in common; • assert that the two result sets for each queryable for the operators `>` and `<=` have no item in common; • assert that the two result sets for each queryable for the operators `<` and `>=` have no item in common; • store the valid predicates for each data source.

A.12.2. Conformance Test 47

Test id:	/conf/property-property/comparison-property-property
Requirements:	/req/property-property/withdraw-permissions
Test purpose:	Test comparison predicates with properties on both sides

Test method:	Given: • One or more data sources, each with a list of queryables. When: For each queryable `{queryable}` of one of the data types String, Boolean, Number, Integer, Timestamp or Date, evaluate the following filter expressions • `{queryable} = {queryable}` • `{queryable} <> {queryable}` • `{queryable} > {queryable}` • `{queryable} < {queryable}` • `{queryable} >= {queryable}` • `{queryable} <= {queryable}` Then: • assert successful execution of the evaluation; • assert that the result sets for each queryable for the operators `<>`, `<` and `>` is empty; • assert that the result sets for each queryable for the operators `=`, `>=` and `<=` are identical; • store the valid predicates for each data source.

A.12.3. Conformance Test 48

Test id:	/conf/property-property/comparison-value-value
Requirements:	/req/property-property/withdraw-permissions
Test purpose:	Test comparison predicates with values on both sides

Test method:	Given: • n/a When: Evaluate the following filter expressions • `{value} = {value}` • `{value} <> {value}` • `{value} > {value}` • `{value} < {value}` • `{value} >= {value}` • `{value} <= {value}` for each `{value}` from the following list: • `'foo'` • `true` • `3.14` • `1` • `TIMESTAMP('2022-04-14T14:48:46Z')` • `DATE('2022-04-14')` Then: • assert successful execution of the evaluation; • assert that the result sets for each queryable for the operators `<>`, `<` and `>` is empty; • assert that the result sets for each queryable for the operators `=`, `>=` and `<=` are identical; • store the valid predicates for each data source.

A.12.4. Conformance Test 49

Test id:	/conf/property-property/test-data
Requirements:	all requirements
Test purpose:	Test predicates against the test dataset

Test method:	Given: • The implementation under test uses the test dataset. When: Evaluate each predicate in Predicates and expected results, if the conditional dependency is met. Then: • assert successful execution of the evaluation; • assert that the expected result is returned; • store the valid predicates for each data source.

Table 16. Predicates and expected results

Dependency	Data Source	Predicate	Expected number of items
n/a	ne_110m_populated_places_simple	`'København'=name`	1
n/a	ne_110m_populated_places_simple	`'København'<=name`	137
n/a	ne_110m_populated_places_simple	`'København'<name`	136
n/a	ne_110m_populated_places_simple	`'København'>=name`	107
n/a	ne_110m_populated_places_simple	`'København'>name`	106
n/a	ne_110m_populated_places_simple	`'København'<>name`	242
n/a	ne_110m_populated_places_simple	`name=nameascii`	230
n/a	ne_110m_populated_places_simple	`name>=nameascii`	243
n/a	ne_110m_populated_places_simple	`name>nameascii`	13
n/a	ne_110m_populated_places_simple	`name<=nameascii`	230
n/a	ne_110m_populated_places_simple	`name<nameascii`	0
n/a	ne_110m_populated_places_simple	`name<>nameascii`	13

Dependency	Data Source	Predicate	Expected number of items
n/a	ne_110m_populated_places_simple	`1038288=pop_other`	1
n/a	ne_110m_populated_places_simple	`1038288<=pop_other`	123
n/a	ne_110m_populated_places_simple	`1038288<pop_other`	122
n/a	ne_110m_populated_places_simple	`1038288>=pop_other`	121
n/a	ne_110m_populated_places_simple	`1038288>pop_other`	120
n/a	ne_110m_populated_places_simple	`1038288<>pop_other`	242
n/a	ne_110m_populated_places_simple	`pop_min=pop_max`	27
n/a	ne_110m_populated_places_simple	`pop_min<=pop_max`	243
n/a	ne_110m_populated_places_simple	`pop_min<pop_max`	216
n/a	ne_110m_populated_places_simple	`pop_min>=pop_max`	27
n/a	ne_110m_populated_places_simple	`pop_min>pop_max`	0
n/a	ne_110m_populated_places_simple	`pop_min<>pop_max`	216
n/a	ne_110m_populated_places_simple	`start=end`	0
n/a	ne_110m_populated_places_simple	`start<=end`	3
n/a	ne_110m_populated_places_simple	`start<end`	3
n/a	ne_110m_populated_places_simple	`start>=end`	0
n/a	ne_110m_populated_places_simple	`start>end`	0
n/a	ne_110m_populated_places_simple	`start<>end`	3
Advanced Comparison Operators	ne_110m_populated_places_simple	`'København' LIKE 'K_benhavn'`	243

Dependency	Data Source	Predicate	Expected number of items
Advanced Comparison Operators	ne_110m_populated_places_simple	`'København' NOT LIKE 'K_benhavn'`	0
Advanced Comparison Operators	ne_110m_populated_places_simple	`pop_other between pop_min and pop_max`	94
Advanced Comparison Operators	ne_110m_populated_places_simple	`pop_other not between pop_min and pop_max`	149
Basic Spatial Functions	ne_110m_admin_0_countries	`S_INTERSECTS(BBOX(0,40,10,50),geom)`	8
Basic Spatial Functions	ne_110m_admin_0_countries	`S_INTERSECTS(BBOX(150,-90,-150,90),geom)`	10
Basic Spatial Functions	ne_110m_admin_0_countries	`S_INTERSECTS(POINT(7.02 49.92),geom)`	1
Basic Spatial Functions	ne_110m_populated_places_simple	`S_INTERSECTS(BBOX(0,40,10,50),geom)`	7
Basic Spatial Functions	ne_110m_rivers_lake_centerlines	`S_INTERSECTS(BBOX(-180,-90,0,90),geom)`	4
Spatial Functions	ne_110m_admin_0_countries	`S_INTERSECTS(POLYGON((0 40,10 40,10 50,0 50,0 40)),geom)`	8
Spatial Functions	ne_110m_admin_0_countries	`S_INTERSECTS(LINESTRING(0 40,10 50),geom)`	4
Spatial Functions	ne_110m_populated_places_simple	`S_INTERSECTS(POLYGON((0 40,10 40,10 50,0 50,0 40)),geom)`	7
Spatial Functions	ne_110m_rivers_lake_centerlines	`S_INTERSECTS(LINESTRING(-60 -90,-60 90),geom)`	2
Spatial Functions	ne_110m_admin_0_countries	`S_DISJOINT(BBOX(0,40,10,50),geom)`	169
Spatial Functions	ne_110m_admin_0_countries	`S_DISJOINT(POLYGON((0 40,10 40,10 50,0 50,0 40)),geom)`	169
Spatial Functions	ne_110m_admin_0_countries	`S_DISJOINT(LINESTRING(0 40,10 50),geom)`	173
Spatial Functions	ne_110m_admin_0_countries	`S_DISJOINT(POINT(7.02 49.92),geom)`	176
Spatial Functions	ne_110m_populated_places_simple	`S_DISJOINT(BBOX(0,40,10,50),geom)`	236
Spatial Functions	ne_110m_populated_places_simple	`S_DISJOINT(POLYGON((0 40,10 40,10 50,0 50,0 40)),geom)`	236

Dependency	Data Source	Predicate	Expected number of items
Spatial Functions	ne_110m_rivers_lake_centerlines	S_DISJOINT(BBOX(-180,-90,0,90),geom)	9
Spatial Functions	ne_110m_rivers_lake_centerlines	S_DISJOINT(LINESTRING(-60 -90,-60 90),geom)	11
Spatial Functions	ne_110m_populated_places_simple	S_EQUALS(POINT(6.1300028 49.6116604),geom)	1
Spatial Functions	ne_110m_admin_0_countries	S_TOUCHES(POLYGON((6.043073357781111 50.128051662794235,6.242751092156993 49.90222565367873,6.186320428094177 49.463802802114515,5.897759230176348 49.44266714130711,5.674051954784829 49.529483547557504,5.782417433300907 50.09032786722122,6.043073357781111 50.128051662794235)),geom)	3
Spatial Functions	ne_110m_admin_0_countries	S_TOUCHES(POINT(6.043073357781111 50.128051662794235),geom)	3
Spatial Functions	ne_110m_admin_0_countries	S_TOUCHES(POINT(6.242751092156993 49.90222565367873),geom)	2
Spatial Functions	ne_110m_admin_0_countries	S_TOUCHES(LINESTRING(6.043073357781111 50.128051662794235,6.242751092156993 49.90222565367873),geom)	3
Spatial Functions	ne_110m_rivers_lake_centerlines	S_CROSSES(BBOX(0,40,10,50),geom)	1
Spatial Functions	ne_110m_rivers_lake_centerlines	S_CROSSES(LINESTRING(-60 -90,-60 90),geom)	2
Spatial Functions	ne_110m_admin_0_countries	S_CONTAINS(BBOX(-180,-90,0,90),geom)	44
Spatial Functions	ne_110m_populated_places_simple	S_CONTAINS(BBOX(-180,-90,0,90),geom)	74
Spatial Functions	ne_110m_rivers_lake_centerlines	S_CONTAINS(BBOX(-180,-90,0,90),geom)	4

Dependency	Data Source	Predicate	Expected number of items
Spatial Functions	ne_110m_admin_0_countries	`S_WITHIN(BBOX(7,50,8,51),geom)`	1
Spatial Functions	ne_110m_admin_0_countries	`S_WITHIN(LINESTRING(7 50,8 51),geom)`	1
Spatial Functions	ne_110m_admin_0_countries	`S_WITHIN(POINT(7.02 49.92),geom)`	1
Spatial Functions	ne_110m_admin_0_countries	`S_OVERLAPS(BBOX(-180,-90,0,90),geom)`	11
Temporal Functions	ne_110m_populated_places_simple	`t_after(date('2022-04-16'),"date")`	1
Temporal Functions	ne_110m_populated_places_simple	`t_before(date('2022-04-16'),"date")`	1
Temporal Functions	ne_110m_populated_places_simple	`t_disjoint(date('2022-04-16'),"date")`	2
Temporal Functions	ne_110m_populated_places_simple	`t_equals(date('2022-04-16'),"date")`	1
Temporal Functions	ne_110m_populated_places_simple	`t_intersects(date('2022-04-16'),"date")`	1
Temporal Functions	ne_110m_populated_places_simple	`t_after(interval('2022-01-01','2022-12-31'),"date")`	1
Temporal Functions	ne_110m_populated_places_simple	`t_before(interval('2022-01-01','2022-12-31'),"date")`	1
Temporal Functions	ne_110m_populated_places_simple	`t_disjoint(interval('2022-01-01','2022-12-31'),"date")`	2
Temporal Functions	ne_110m_populated_places_simple	`t_equals(interval('2022-01-01','2022-12-31'),"date")`	0
Temporal Functions	ne_110m_populated_places_simple	`t_equals(interval('2022-04-16','2022-04-16'),"date")`	1
Temporal Functions	ne_110m_populated_places_simple	`t_intersects(interval('2022-01-01','2022-12-31'),"date")`	1
Temporal Functions	ne_110m_populated_places_simple	`t_after(timestamp('2022-04-16T10:13:19Z'),start)`	1
Temporal Functions	ne_110m_populated_places_simple	`t_before(timestamp('2022-04-16T10:13:19Z'),start)`	1

Dependency	Data Source	Predicate	Expected number of items
Temporal Functions	ne_110m_populated_places_simple	`t_disjoint(timestamp('2022-04-16T10:13:19Z'),start)`	2
Temporal Functions	ne_110m_populated_places_simple	`t_equals(timestamp('2022-04-16T10:13:19Z'),start)`	1
Temporal Functions	ne_110m_populated_places_simple	`t_intersects(timestamp('2022-04-16T10:13:19Z'),start)`	1
Temporal Functions	ne_110m_populated_places_simple	`t_after(interval('2022-01-01T00:00:00Z','2022-12-31T23:59:59Z'),start)`	1
Temporal Functions	ne_110m_populated_places_simple	`t_before(interval('2022-01-01T00:00:00Z','2022-12-31T23:59:59Z'),start)`	0
Temporal Functions	ne_110m_populated_places_simple	`t_disjoint(interval('2022-01-01T00:00:00Z','2022-12-31T23:59:59Z'),start)`	1
Temporal Functions	ne_110m_populated_places_simple	`t_equals(interval('2022-01-01T00:00:00Z','2022-12-31T23:59:59Z'),start)`	0
Temporal Functions	ne_110m_populated_places_simple	`t_intersects(interval('2022-01-01T00:00:00Z','2022-12-31T23:59:59Z'),start)`	2
Temporal Functions	ne_110m_populated_places_simple	`t_after(interval('2023-01-01T00:00:00Z','..'),interval(start,end))`	2
Temporal Functions	ne_110m_populated_places_simple	`t_before(interval('..','2022-04-16T10:13:19Z'),interval(start,end))`	1
Temporal Functions	ne_110m_populated_places_simple	`t_disjoint(interval('2022-04-16T10:13:19Z','2022-04-16T10:15:09Z'),interval(start,end))`	1

Dependency	Data Source	Predicate	Expected number of items
Temporal Functions	ne_110m_populated_places_simple	t_equals(interval('2021-04-16T10:15:59Z','2022-04-16T10:16:06Z'),interval(start,end))	1
Temporal Functions	ne_110m_populated_places_simple	t_intersects(interval('2022-04-16T10:13:19Z','2022-04-16T10:15:09Z'),interval(start,end))	2
Temporal Functions	ne_110m_populated_places_simple	T_CONTAINS(interval('2021-04-16T10:13:19Z','2023-04-16T10:15:10Z'),interval(start,end))	2
Temporal Functions	ne_110m_populated_places_simple	T_DURING(interval('2022-07-01T00:00:00Z','2022-12-31T23:59:59Z'),interval(start,end))	1
Temporal Functions	ne_110m_populated_places_simple	T_FINISHES(interval('2022-04-16T10:13:19Z','2022-04-16T10:16:06Z'),interval(start,end))	1
Temporal Functions	ne_110m_populated_places_simple	T_FINISHEDBY(interval('2022-04-16T10:13:19Z','2022-04-16T10:16:06Z'),interval(start,end))	0
Temporal Functions	ne_110m_populated_places_simple	T_MEETS(interval('2022-04-16T10:13:19Z','2022-04-16T10:15:10Z'),interval(start,end))	1
Temporal Functions	ne_110m_populated_places_simple	T_METBY(interval('2022-04-16T10:13:19Z','2022-04-16T10:15:10Z'),interval(start,end))	0

Dependency	Data Source	Predicate	Expected number of items
Temporal Functions	ne_110m_populated_places_simple	`T_OVERLAPPEDBY(interval('2020-04-16T10:13:19Z','2022-04-16T10:15:10Z'),interval(start,end))`	0
Temporal Functions	ne_110m_populated_places_simple	`T_OVERLAPS(interval('2022-04-16T10:13:19Z','2023-04-16T10:15:10Z'),interval(start,end))`	0
Temporal Functions	ne_110m_populated_places_simple	`T_STARTEDBY(interval('2022-04-16T10:13:19Z','2022-04-16T10:15:10Z'),interval(start,end))`	0
Temporal Functions	ne_110m_populated_places_simple	`T_STARTS(interval('2022-04-16T10:13:19Z','2022-04-16T10:15:10Z'),interval(start,end))`	1

A.12.5. Conformance Test 50

Test id:	/conf/property-property/logical
Requirements:	n/a
Test purpose:	Test filter expressions with AND, OR and NOT including sub-expressions
Test method:	Given: • The stored predicates for each data source, including from the dependencies. When: For each data source, select at least 10 random combinations of four predicates (`{p1}` to `{p4}`) from the stored predicates and evaluate the filter expression `((NOT {p1} AND {p2}) OR ({p3} and NOT {p4}) or not ({p1} AND {p4}))`. Then: • assert successful execution of the evaluation.

A.13. Conformance Class "Functions"

Conformance Class	
http://www.opengis.net/spec/cql2/1.0/conf/functions	
Target type	Servers that evaluate filter expressions
Parameter	Filter Language: "CQL2 Text" or "CQL2 JSON"
Requirements class	Requirements Class "Functions"
Dependency	Basic CQL2

A.13.1. Conformance Test 51

Test id:	/conf/functions/functions
Requirements:	/req/functions/functions
Test purpose:	Test predicates with functions
Test method:	Given: • The list of functions with arguments and return type supported by the implementation under test. When: • For each function construct multiple valid filter expressions involving different operators. Then: • assert successful execution of the evaluation.

A.14. Conformance Class "Arithmetic Expressions"

Conformance Class	
http://www.opengis.net/spec/cql2/1.0/conf/arithmetic	
Target type	Servers that evaluate filter expressions
Parameter	Filter Language: "CQL2 Text" or "CQL2 JSON"
Requirements class	Requirements Class "Arithmetic Expressions"
Dependency	Basic CQL2
Conditional Dependency	Advanced Comparison Operators

Conditional Dependency	Property-Property Comparisons

A.14.1. Conformance Test 52

Test id:	/conf/arithmetic/arithmetic
Requirements:	/req/arithmetic/arithmetic
Test purpose:	Test predicates with arithmetic expressions
Test method:	Given: • One or more data sources, each with a list of queryables. • At least one queryable has a numeric data type. When: • For each queryable construct multiple valid filter expressions involving arithmetic expressions. Then: • assert successful execution of the evaluation.

A.14.2. Conformance Test 53

Test id:	/conf/arithmetic/test-data
Requirements:	all requirements
Test purpose:	Test predicates against the test dataset
Test method:	Given: • The implementation under test uses the test dataset. When: Evaluate each predicate in Predicates and expected results, if the conditional dependency is met. Then: • assert successful execution of the evaluation; • assert that the expected result is returned; • store the valid predicates for each data source.

Table 17. Predicates and expected results

Dependency	Data Source	Predicate	Expected number of items
n/a	ne_110m_populated_places_simple	`pop_other=1038280+8`	1
n/a	ne_110m_populated_places_simple	`pop_other>=1038290-2*2^0`	123
n/a	ne_110m_populated_places_simple	`pop_other>1038290-20/10`	122
n/a	ne_110m_populated_places_simple	`pop_other>1038290-21 div 10`	122
n/a	ne_110m_populated_places_simple	`pop_other>1038290-5%2`	122
n/a	ne_110m_populated_places_simple	`pop_other<=1038200+8*11`	121
n/a	ne_110m_populated_places_simple	`pop_other<1038280+2^3`	120
n/a	ne_110m_populated_places_simple	`pop_other<>1038290-2^1`	242
Advanced Comparison Operators	ne_110m_populated_places_simple	`pop_other between 4000000/4 and (3*(900000+100000))`	75
Advanced Comparison Operators	ne_110m_populated_places_simple	`pop_other not between 4000000/4 and (3*(900000+100000))`	168
Advanced Comparison Operators	ne_110m_populated_places_simple	`pop_other in (1000000+38288,1000000+600000+11692,3*1000000+13258,3*1000000+13257,30*100000+13259)`	3
Advanced Comparison Operators	ne_110m_populated_places_simple	`pop_other not in (1000000+38288,1000000+600000+11692,3*1000000+13258,3*1000000+13257,30*100000+13259)`	240
Property-Property Comparisons	ne_110m_populated_places_simple	`1038280+8=pop_other`	1

A.14.3. Conformance Test 54

Test id:	/conf/arithmetic/logical
Requirements:	n/a
Test purpose:	Test filter expressions with AND, OR and NOT including sub-expressions

Test method:	Given: • The stored predicates for each data source, including from the dependencies. When: For each data source, select at least 10 random combinations of four predicates (`{p1}` to `{p4}`) from the stored predicates and evaluate the filter expression `((NOT {p1} AND {p2}) OR ({p3} and NOT {p4}) or not ({p1} AND {p4}))`. Then: • assert successful execution of the evaluation.

Annex B: CQL2 BNF (Normative)

NOTE

Because there are many variations of EBNF, this standard has focused on verifying that this grammar validates using the following online validator:

https://www.icosaedro.it/bnf_chk/bnf_chk-on-line.html

If other tools are used (e.g. ANTLR), the grammar will likely need to be adapted to suit the target implementation context.

```
#
# MODULE:  cql2.bnf
# PURPOSE: A BNF grammar for the Common Query Language (CQL2).
# HISTORY:
# DATE         EMAIL                                   DESCRIPTION
# 13-SEP-2019  pvretano[at]cubewerx.com                Initial creation
# 28-OCT-2019  pvretano[at]cubewerx.com                Initial check-in into github
# 22-DEC-2020  pvretano[at]cubewerx.com                1.0.0-draft.1 (version for
public review)
#              portele[at]interactive-instruments.de
# 07-MAR-2024  pvretano[at]cubewerx.com                1.0.0-rc.1 (release candidate)
#              portele[at]interactive-instruments.de
# 02-JUL-2024  pvretano[at]cubewerx.com                1.0.0 (final release)
#              portele[at]interactive-instruments.de

#=============================================================================#
# A CQL2 filter is a logically connected expression of one or more predicates.
# Predicates include scalar or comparison predicates, spatial predicates or
# temporal predicates.
#
# *** DISCLAIMER: ***
# Because there are many variations of EBNF, this standard has focused
# on verifying that this grammar validates using the following online
# validator:
#
# https://www.icosaedro.it/bnf_chk/bnf_chk-on-line.html
#
# If other tools are used (e.g. ANTLR), the grammar will likely need to be
# adapted to suite the target implementation context.
#=============================================================================#
booleanExpression = booleanTerm [ {"OR" booleanTerm} ];

booleanTerm = booleanFactor [ {"AND" booleanFactor} ];

booleanFactor = ["NOT"] booleanPrimary;

booleanPrimary = function
               | predicate
               | booleanLiteral
```

```
               | "(" booleanExpression ")";

predicate = comparisonPredicate
          | spatialPredicate
          | temporalPredicate
          | arrayPredicate;

#============================================================================#
# A comparison predicate evaluates if two scalar expression statisfy the
# specified comparison operator.  The comparion operators includes an operator
# to evaluate pattern matching expressions (LIKE), a range evaluation operator
# and an operator to test if a scalar expression is NULL or not.
#============================================================================#
comparisonPredicate = binaryComparisonPredicate
                    | isLikePredicate
                    | isBetweenPredicate
                    | isInListPredicate
                    | isNullPredicate;

# Binary comparison predicate
#
binaryComparisonPredicate = scalarExpression
                            comparisonOperator
                            scalarExpression;

scalarExpression = characterClause
                 | numericLiteral
                 | instantInstance
                 | arithmeticExpression
                 | booleanLiteral
                 | propertyName
                 | function;

comparisonOperator = "="      # equal
                   | "<" ">"  # not equal
                   | "<"      # less than
                   | ">"      # greater than
                   | "<" "="  # less than or equal
                   | ">" "="; # greater than or equal

# LIKE predicate
#
isLikePredicate =  characterExpression ["NOT"] "LIKE" patternExpression;

patternExpression = "CASEI" "(" patternExpression ")"
                  | "ACCENTI" "(" patternExpression ")"
                  | characterLiteral;

# BETWEEN predicate
#
isBetweenPredicate = numericExpression ["NOT"] "BETWEEN"
```

```
                    numericExpression "AND" numericExpression;

numericExpression = arithmeticExpression
                  | numericLiteral
                  | propertyName
                  | function;

# IN LIST predicate
#
isInListPredicate = scalarExpression ["NOT"] "IN" "(" inList ")";

inList = scalarExpression [ {"," scalarExpression} ];

# IS NULL predicate
#
isNullPredicate = isNullOperand "IS" ["NOT"] "NULL";

isNullOperand = characterClause
              | numericLiteral
              | temporalInstance
              | spatialInstance
              | arithmeticExpression
              | booleanExpression
              | propertyName
              | function;

#============================================================================#
# A spatial predicate evaluates if two spatial expressions satisfy the
# condition implied by a standardized spatial comparison function.  If the
# conditions of the spatial comparison function are met, the function returns
# a Boolean value of true.  Otherwise the function returns false.
#============================================================================#
spatialPredicate =  spatialFunction "(" geomExpression "," geomExpression ")";

# NOTE: The buffer functions (DWITHIN and BEYOND) are not included because
#       these are outside the scope of a "simple" core for CQL2.  These
#       can be added as extensions.
#
spatialFunction = "S_INTERSECTS"
                | "S_EQUALS"
                | "S_DISJOINT"
                | "S_TOUCHES"
                | "S_WITHIN"
                | "S_OVERLAPS"
                | "S_CROSSES"
                | "S_CONTAINS";

# A geometric expression is a property name of a geometry-valued property,
# a geometric literal (expressed as WKT) or a function that returns a
# geometric value.
#
```

```
geomExpression = spatialInstance
               | propertyName
               | function;

#============================================================================#
# A temporal predicate evaluates if two temporal expressions satisfy the
# condition implied by a standardized temporal comparison function.  If the
# conditions of the temporal comparison function are met, the function returns
# a Boolean value of true.  Otherwise the function returns false.
#============================================================================#
temporalPredicate = temporalFunction
                    "(" temporalExpression "," temporalExpression ")";

temporalExpression = temporalInstance
                   | propertyName
                   | function;

temporalFunction = "T_AFTER"
                 | "T_BEFORE"
                 | "T_CONTAINS"
                 | "T_DISJOINT"
                 | "T_DURING"
                 | "T_EQUALS"
                 | "T_FINISHEDBY"
                 | "T_FINISHES"
                 | "T_INTERSECTS"
                 | "T_MEETS"
                 | "T_METBY"
                 | "T_OVERLAPPEDBY"
                 | "T_OVERLAPS"
                 | "T_STARTEDBY"
                 | "T_STARTS";

#============================================================================#
# An array predicate evaluates if two array expressions satisfy the
# condition implied by a standardized array comparison function.  If the
# conditions of the array comparison function are met, the function returns
# a Boolean value of true.  Otherwise the function returns false.
#============================================================================#
arrayPredicate = arrayFunction
                 "(" arrayExpression "," arrayExpression ")";

arrayExpression = array
                | propertyName
                | function;

# An array is a parentheses-delimited, comma-separated list of array
# elements.
array = "(" ")"
      | "(" arrayElement [ { "," arrayElement } ] ")";
```

```
# An array element is either a character literal, a numeric literal,
# a geometric literal, a temporal instance, a property name, a function,
# an arithmetic expression or an array.
arrayElement = characterClause
             | numericLiteral
             | temporalInstance
             | spatialInstance
             | array
             | arithmeticExpression
             | booleanExpression
             | propertyName
             | function;

arrayFunction = "A_EQUALS"
              | "A_CONTAINS"
              | "A_CONTAINEDBY"
              | "A_OVERLAPS";

#========================================================================#
# An arithmetic expression is an expression composed of an arithmetic
# operand (a property name, a number or a function that returns a number),
# an arithmetic operators (+,-,*,/,%,div,^) and another arithmetic operand.
#========================================================================#

arithmeticExpression = arithmeticTerm [ {arithmeticOperatorPlusMinus arithmeticTerm}
];

arithmeticOperatorPlusMinus = "+" | "-";

arithmeticTerm = powerTerm [ {arithmeticOperatorMultDiv powerTerm} ];

arithmeticOperatorMultDiv = "*" | "/" | "%" | "div";

powerTerm = arithmeticFactor [ "^" arithmeticFactor ];

arithmeticFactor = "(" arithmeticExpression ")"
                 | [ "-" ] arithmeticOperand;

arithmeticOperand = numericLiteral
                  | propertyName
                  | function;

#========================================================================#
# Definition of a PROPERTYNAME
# Production copied from: https://www.w3.org/TR/REC-xml/#sec-common-syn,
#                         "Names and Tokens".
#========================================================================#
propertyName = identifier | "\"" identifier "\"";

identifier = identifierStart [ { identifierPart } ];
```

```
identifierPart = identifierStart
               | "."                    # "\x002E"
               | digit                  # 0-9
               | "\x0300".."\x036F"     # combining and diacritical marks
               | "\x203F".."\x2040";    # ‿ and ⁀

identifierStart = "\x003A"              # colon
                | "\x005F"              # underscore
                | "\x0041".."\x005A"    # A-Z
                | "\x0061".."\x007A"    # a-z
                | "\x00C0".."\x00D6"    # À-Ö Latin-1 Supplement Letters
                | "\x00D8".."\x00F6"    # Ø-ö Latin-1 Supplement Letters
                | "\x00F8".."\x02FF"    # ø-ÿ Latin-1 Supplement Letters
                | "\x0370".."\x037D"    # Ͱ-ͽ Greek and Coptic (without ";")
                | "\x037F".."\x1FFE"    # See note 1.
                | "\x200C".."\x200D"    # zero width non-joiner and joiner
                | "\x2070".."\x218F"    # See note 2.
                | "\x2C00".."\x2FEF"    # See note 3.
                | "\x3001".."\xD7FF"    # See note 4.
                | "\xF900".."\xFDCF"    # See note 5.
                | "\xFDF0".."\xFFFD"    # See note 6.
                | "\x10000".."\xEFFFF"; # See note 7.

# See: https://unicode-table.com/en/blocks/

# Note 1: Greek, Coptic, Cyrillic, Cyrillic Supplement, Armenian, Hebrew,
#         Arabic, Syriac, Arabic Supplement, Thaana, NKo, Samaritan, Mandaic,
#         Syriac Supplement, Arabic Extended-A, Devanagari, Bengali, Gurmukhi,
#         Gujarati, Oriya, Tamil, Telugu, Kannada, Malayalam, Sinhala, Thai,
#         Lao, Tibetan, Myanmar, Georgian, Hangul Jamo, Ethiopic, Ethiopic
#         Supplement, Cherokee, Unified Canadian Aboriginal Syllabics, Ogham,
#         Runic, Tagalog, Hanunoo, Buhid, Tagbanwa, Khmer, Mongolian, Unified
#         Canadian Aboriginal Syllabics Extended, Limbu, Tai Le, New Tai Lue,
#         Khmer Symbols, Buginese, Tai Tham, Combining Diacritical Marks
#         Extended, Balinese, Sundanese, Batak, Lepcha, Ol Chiki, Cyrillic
#         Extended C, Georgian Extended, Sundanese Supplement, Vedic
#         Extensions, Phonetic Extensions, Phonetic Extensions Supplement,
#         Combining Diacritical Marks Supplement, Latin Extended Additional,
#         Greek Extended
#
# Note 2: Superscripts and Subscripts, Currency Symbols, Combining Diacritical
#         Marks for Symbols, Letterlike Symbols, Number Forms (e.g. Roman
#         numbers)
#
# Note 3: Glagolitic, Latin Extended-C, Coptic, Georgian Supplement, Tifinagh,
#         Ethiopic Extended, Cyrillic Extended-A, Supplemental Punctuation,
#         CJK Radicals Supplement, Kangxi Radicals
#
# Note 4: CJK Symbols and Punctuation Hiragana, Katakana, Bopomofo, Hangul
#         Compatibility Jamo, Kanbun, Bopomofo Extended, CJK Strokes, Katakana
#         Phonetic Extensions, Enclosed CJK Letters and Months, CJK
```

```
#         Compatibility, CJK Unified Ideographs Extension A, Yijing Hexagram
#         Symbols, CJK Unified Ideographs, Yi Syllables, Yi Radicals, Lisu,
#         Vai, Cyrillic Extended-B, Bamum, Modifier Tone Letters, Latin
#         Extended-D, Syloti Nagri, Common Indic Number Forms, Phags-pa,
#         Saurashtra, Devanagari Extended, Kayah Li, Rejang, Hangul Jamo
#         Extended-A, Javanese, Myanmar Extended-B, Cham, Myanmar Extended-A,
#         Tai Viet, Meetei Mayek Extensions, Ethiopic Extended-A, Latin
#         Extended-E, Cherokee Supplement, Meetei Mayek, Hangul Syllables,
#         Hangul Jamo Extended-B
#
# Note 5: CJK Compatibility Ideographs, Alphabetic Presentation Forms,
#         Arabic Presentation Forms-A
#
# Note 6: Arabic Presentation Forms-A, Variation Selectors, Vertical Forms,
#         Combining Half Marks, CJK Compatibility Forms, Small Form Variants,
#         Arabic Presentation Forms-B, Halfwidth and Fullwidth Forms, Specials
#
# Note 7: Linear B Syllabary, Linear B Ideograms, Aegean Numbers, Ancient
#         Greek Numbers, Ancient Symbols, Phaistos Disc, Lycian, Carian,
#         Coptic Epact Numbers, Old Italic, Gothic, Old Permic, Ugaritic, Old
#         Persian, Deseret, Shavian, Osmanya, Osage, Elbasan, Caucasian
#         Albanian, Linear A, Cypriot Syllabary, Imperial Aramaic, Palmyrene,
#         Nabataean, Hatran, Phoenician, Lydian, Meroitic Hieroglyphs,
#         Meroitic Cursive, Kharoshthi, Old South Arabian, Old North Arabian,
#         Manichaean, Avestan, Inscriptional Parthian, Inscriptional Pahlavi,
#         Psalter Pahlavi, Old Turkic, Old Hungarian, Hanifi Rohingya, Rumi
#         Numeral Symbols, Yezidi, Old Sogdian, Sogdian, Chorasmian, Elymaic,
#         Brahmi, Kaithi, Sora Sompeng, Chakma, Mahajani, Sharada, Sinhala
#         Archaic Numbers, Khojki, Multani, Khudawadi, Grantha, Newa, Tirhuta,
#         Siddham, Modi, Mongolian Supplement, Takri, Ahom, Dogra, Warang Citi,
#         Dives Akuru, Nandinagari, Zanabazar Square, Soyombo, Pau Cin Hau,
#         Bhaiksuki, Marchen, Masaram Gondi, Gunjala Gondi, Makasar, Lisu
#         Supplement, Tamil Supplement, Cuneiform, Cuneiform Numbers and
#         Punctuation, Early Dynastic Cuneiform, Egyptian Hieroglyphs,
#         Egyptian Hieroglyph Format Controls, Anatolian Hieroglyphs,Bamum
#         Supplement, Mro, Bassa Vah, Pahawh Hmong, Medefaidrin, Miao,
#         Ideographic Symbols and Punctuation, Tangut, Tangut Components,
#         Khitan Small Script, Tangut Supplement, Kana Supplement, Kana
#         Extended-A, Small Kana Extension, Nushu, Duployan, Shorthand Format
#         Controls, Byzantine Musical Symbols, Musical Symbols, Ancient Greek
#         Musical Notation, Mayan Numerals, Tai Xuan Jing Symbols, Counting
#         Rod Numerals, Mathematical Alphanumeric Symbols, Sutton SignWriting,
#         Glagolitic Supplement, Nyiakeng Puachue Hmong, Wancho, Mende Kikakui,
#         Adlam, Indic Siyaq Numbers, Ottoman Siyaq Numbers, Arabic
#         Mathematical Alphabetic Symbols, Mahjong Tiles, Domino Tiles,
#         Playing Cards, Enclosed Alphanumeric Supplement, Enclosed Ideographic
#         Supplement, Miscellaneous Symbols and Pictographs, Emoticons (Emoji),
#         Ornamental Dingbats, Transport and Map Symbols, Alchemical Symbols,
#         Geometric Shapes Extended, Supplemental Arrows-C, Supplemental
#         Symbols and Pictographs, Chess Symbols, Symbols and Pictographs
#         Extended-A, Symbols for Legacy Computing, CJK Unified Ideographs
```

```
#         Extension B, CJK Unified Ideographs Extension C, CJK Unified
#         Ideographs Extension D, CJK Unified Ideographs Extension E, CJK
#         Unified Ideographs Extension F, CJK Compatibility Ideographs
#         Supplement, CJK Unified Ideographs Extension G, Tags, Variation
#         Selectors Supplement

#=============================================================================#
# Definition of a FUNCTION
#=============================================================================#
function = identifier "(" {argumentList} ")";

argumentList = argument [ { "," argument } ];

argument = characterClause
         | numericLiteral
         | temporalInstance
         | spatialInstance
         | array
         | arithmeticExpression
         | booleanExpression
         | propertyName
         | function;

#=============================================================================#
# Character expression
#=============================================================================#
characterExpression = characterClause
                    | propertyName
                    | function;

characterClause = "CASEI" "(" characterExpression ")"
                | "ACCENTI" "(" characterExpression ")"
                | characterLiteral;

#=============================================================================#
# Definition of CHARACTER literals
#=============================================================================#
characterLiteral = "'" [ {character} ] "'";

character = alpha | digit | whitespace | escapeQuote;

escapeQuote = "''" | "\\'";

# character & digit productions copied from:
# https://www.w3.org/TR/REC-xml/#charsets
#
alpha = "\x0007".."\x0008"     # bell, bs
      | "\x0021".."\x0026"     # !, ", #, $, %, &
      | "\x0028".."\x002F"     # (, ), *, +, comma, -, ., /
      | "\x003A".."\x0084"     # --+
      | "\x0086".."\x009F"     #   |
```

```
      | "\x00A1".."\x167F"     #   |
      | "\x1681".."\x1FFF"     #   |
      | "\x200B".."\x2027"     #   +-> :,;,<,=,>,?,@,A-Z,[,\,],^,_,`,a-z,...
      | "\x202A".."\x202E"     #   |
      | "\x2030".."\x205E"     #   |
      | "\x2060".."\x2FFF"     #   |
      | "\x3001".."\xD7FF"     # --+
      | "\xE000".."\xFFFD"     # See note 8.
      | "\x10000".."\x10FFFF"; # See note 9.

# Note 8: Private Use, CJK Compatibility Ideographs, Alphabetic Presentation
#         Forms, Arabic Presentation Forms-A, Combining Half Marks, CJK
#         Compatibility Forms, Small Form Variants, Arabic Presentation Forms-B,
#         Specials, Halfwidth and Fullwidth Forms, Specials
# Note 9: Linear B Syllabary, Linear B Ideograms, Aegean Numbers, Ancient Greek
#         Numbers, Ancient Symbols, Phaistos Disc, Lycian, Carian, Coptic
#         Epact Numbers, Old Italic, Gothic, Old Permic, Ugaritic, Old Persian,
#         Deseret, Shavian, Osmanya, Osage, Elbasan, Caucasian Albanian,
#         Vithkuqi, Linear A, Latin Extended-F, Cypriot Syllabary, Imperial
#         Aramaic, Palmyrene, Nabataean, Hatran, Phoenician, Lydian, Meroitic
#         Hieroglyphs, Meroitic Cursive, Kharoshthi, Old South Arabian, Old
#         North Arabian, Manichaean, Avestan, Inscriptional Parthian,
#         Inscriptional Pahlavi, Psalter Pahlavi, Old Turkic, Old Hungarian,
#         Hanifi Rohingya, Rumi Numeral Symbols, Yezidi, Arabic Extended-C,
#         Old Sogdian, Sogdian, Old Uyghur, Chorasmian, Elymaic, Brahmi,
#         Kaithi, Sora Sompeng, Chakma, Mahajani, Sharada, Sinhala Archaic
#         Numbers, Khojki, Multani, Khudawadi, Grantha, Newa, Tirhuta, Siddham,
#         Modi, Mongolian Supplement, Takri, Ahom, Dogra, Warang Citi, Dives
#         Akuru, Nandinagari, Zanabazar Square, Soyombo, Unified Canadian
#         Aboriginal Syllabics Extended-A, Pau Cin Hau, Devanagari Extended-A,
#         Bhaiksuki, Marchen, Masaram Gondi, Gunjala Gondi, Makasar, Kawi,
#         Lisu Supplement, Tamil Supplement, Cuneiform, Cuneiform Numbers and
#         Punctuation, Early Dynastic Cuneiform, Cypro-Minoan, Egyptian
#         Hieroglyphs, Egyptian Hieroglyph Format Controls, Anatolian
#         Hieroglyphs, Bamum Supplement, Mro, Tangsa, Bassa Vah, Pahawh Hmong,
#         Medefaidrin, Miao, Ideographic Symbols and Punctuation, Tangut,
#         Tangut Components, Khitan Small Script, Tangut Supplement, Kana
#         Extended-B, Kana Supplement, Kana Extended-A, Small Kana Extension,
#         Nushu, Duployan, Shorthand Format Controls, Znamenny Musical Notation,
#         Byzantine Musical Symbols, Musical Symbols, Ancient Greek Musical
#         Notation, Kaktovik Numerals, Mayan Numerals, Tai Xuan Jing Symbols,
#         Counting Rod Numerals, Mathematical Alphanumeric Symbols, Sutton
#         SignWriting, Latin Extended-G, Glagolitic Supplement, Cyrillic
#         Extended-D, Nyiakeng Puachue Hmong, Toto, Wancho, Nag Mundari,
#         Ethiopic Extended-B, Mende Kikakui, Adlam, Indic Siyaq Numbers,
#         Ottoman Siyaq Numbers, Arabic Mathematical Alphabetic Symbols,
#         Mahjong Tiles, Domino Tiles, Playing Cards, Enclosed Alphanumeric
#         Supplement, Enclosed Ideographic Supplement, Miscellaneous Symbols
#         and Pictographs, Emoticons, Ornamental Dingbats, Transport and Map
#         Symbols, Alchemical Symbols, Geometric Shapes Extended, Supplemental
#         Arrows-C, Supplemental Symbols and Pictographs, Chess Symbols, Symbols
```

```
#         and Pictographs Extended-A, Symbols for Legacy Computing, CJK Unified
#         Ideographs Extension B, CJK Unified Ideographs Extension C, CJK
#         Unified Ideographs Extension D, CJK Unified Ideographs Extension E,
#         CJK Unified Ideographs Extension F, CJK Compatibility Ideographs
#         Supplement, CJK Unified Ideographs Extension G, CJK Unified
#         Ideographs Extension H, Tags, Variation Selectors Supplement,
#         Supplementary Private Use Area-A, Supplementary Private Use Area-B

digit = "\x0030".."\x0039";

whitespace = "\x0009"  # Character tabulation
           | "\x000A"  # Line feed
           | "\x000B"  # Line tabulation
           | "\x000C"  # Form feed
           | "\x000D"  # Carriage return
           | "\x0020"  # Space
           | "\x0085"  # Next line
           | "\x00A0"  # No-break space
           | "\x1680"  # Ogham space mark
           | "\x2000"  # En quad
           | "\x2001"  # Em quad
           | "\x2002"  # En space
           | "\x2003"  # Em space
           | "\x2004"  # Three-per-em space
           | "\x2005"  # Four-per-em space
           | "\x2006"  # Six-per-em space
           | "\x2007"  # Figure space
           | "\x2008"  # Punctuation space
           | "\x2009"  # Thin space
           | "\x200A"  # Hair space
           | "\x2028"  # Line separator
           | "\x2029"  # Paragraph separator
           | "\x202F"  # Narrow no-break space
           | "\x205F"  # Medium mathematical space
           | "\x3000"; # Ideographic space

#=========================================================================#
# Definition of NUMERIC literals
#=========================================================================#
numericLiteral = unsignedNumericLiteral | signedNumericLiteral;

unsignedNumericLiteral = decimalNumericLiteral | scientificNumericLiteral;

signedNumericLiteral = [sign] unsignedNumericLiteral;

decimalNumericLiteral = unsignedInteger [ "." [ unsignedInteger ] ]
                      | "." unsignedInteger;

scientificNumericLiteral = mantissa "E" exponent;

mantissa = decimalNumericLiteral;
```

```
exponent = signedInteger;

signedInteger = [ sign ] unsignedInteger;

unsignedInteger = {digit};

sign = "+" | "-";

#============================================================================#
# Boolean literal
#============================================================================#
#
booleanLiteral = "TRUE" | "FALSE";

#============================================================================#
# Definition of GEOMETRIC literals
#
# NOTE: This is basically BNF that define WKT encoding. It would be nice
#       to instead reference some normative BNF for WKT.
#============================================================================#
spatialInstance = geometryLiteral
                | geometryCollectionTaggedText
                | bboxTaggedText;

geometryLiteral = pointTaggedText
                | linestringTaggedText
                | polygonTaggedText
                | multipointTaggedText
                | multilinestringTaggedText
                | multipolygonTaggedText;

pointTaggedText = "POINT" ["Z"] pointText;

linestringTaggedText = "LINESTRING" ["Z"] lineStringText;

polygonTaggedText = "POLYGON" ["Z"] polygonText;

multipointTaggedText = "MULTIPOINT" ["Z"] multiPointText;

multilinestringTaggedText = "MULTILINESTRING" ["Z"] multiLineStringText;

multipolygonTaggedText = "MULTIPOLYGON" ["Z"] multiPolygonText;

geometryCollectionTaggedText = "GEOMETRYCOLLECTION" ["Z"] geometryCollectionText;

pointText = "(" point ")";

point = xCoord yCoord [zCoord];

xCoord = signedNumericLiteral;
```

```
yCoord = signedNumericLiteral;

zCoord = signedNumericLiteral;

lineStringText = "(" point "," point {"," point} ")";

linearRingText = emptySet | "(" point "," point "," point "," point {"," point } ")";

polygonText =  "(" linearRingText {"," linearRingText} ")";

multiPointText = "(" pointText {"," pointText} ")";

multiLineStringText = "(" lineStringText {"," lineStringText} ")";

multiPolygonText = "(" polygonText {"," polygonText} ")";

geometryCollectionText = "(" geometryLiteral {"," geometryLiteral} ")";

bboxTaggedText = "BBOX" bboxText;

bboxText = "(" westBoundLon "," southBoundLat "," [minElev ","] eastBoundLon ","
northBoundLat ["," maxElev] ")";

westBoundLon = signedNumericLiteral;

eastBoundLon = signedNumericLiteral;

northBoundLat = signedNumericLiteral;

southBoundLat = signedNumericLiteral;

minElev = signedNumericLiteral;

maxElev = signedNumericLiteral;

temporalInstance = instantInstance | intervalInstance;

instantInstance = dateInstant | timestampInstant;

dateInstant = "DATE"
              "(" dateInstantString ")";

dateInstantString = "'" fullDate "'";

timestampInstant = "TIMESTAMP"
                   "(" timestampInstantString ")";

timestampInstantString = "'" fullDate "T" utcTime "'";

intervalInstance = "INTERVAL" "(" instantParameter "," instantParameter ")";
```

```
instantParameter = dateInstantString
                 | timestampInstantString
                 | "'..'"
                 | propertyName
                 | function;

fullDate   = dateYear "-" dateMonth "-" dateDay;

dateYear   = digit digit digit digit;

dateMonth  = digit digit;

dateDay    = digit digit;

utcTime  = timeHour ":" timeMinute ":" timeSecond "Z";

timeHour   = digit digit;

timeMinute = digit digit;

timeSecond = digit digit ["." digit {digit}];
```

Annex C: JSON schemas for CQL2 (Normative)

C.1. JSON Schema for CQL2

The following document specifies the schema for CQL2 according to JSON Schema version '2020-12':

```
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "$dynamicAnchor": "cql2expression",
  "oneOf": [
    {"$ref": "#/$defs/andOrExpression"    },
    {"$ref": "#/$defs/notExpression"      },
    {"$ref": "#/$defs/comparisonPredicate"},
    {"$ref": "#/$defs/spatialPredicate"   },
    {"$ref": "#/$defs/temporalPredicate"  },
    {"$ref": "#/$defs/arrayPredicate"     },
    {"$ref": "#/$defs/functionRef"        },
    {"type": "boolean"                    }
  ],
  "$defs": {
    "andOrExpression": {
      "type": "object",
      "required": ["op", "args"],
      "properties": {
        "op": { "type": "string", "enum": ["and", "or"] },
        "args": {
          "type": "array",
          "minItems": 2,
          "items": {"$dynamicRef": "#cql2expression"}
        }
      }
    },
    "notExpression": {
      "type": "object",
      "required": ["op", "args"],
      "properties": {
        "op": { "type": "string", "enum": ["not"] },
        "args": {
          "type": "array",
          "minItems": 1,
          "maxItems": 1,
          "items": {"$dynamicRef": "#cql2expression"}
        }
      }
    },
    "comparisonPredicate": {
      "oneOf": [
```

```
      {"$ref": "#/$defs/binaryComparisonPredicate"},
      {"$ref": "#/$defs/isLikePredicate"          },
      {"$ref": "#/$defs/isBetweenPredicate"       },
      {"$ref": "#/$defs/isInListPredicate"        },
      {"$ref": "#/$defs/isNullPredicate"          }
    ]
  },
  "binaryComparisonPredicate": {
    "type": "object",
    "required": ["op", "args"],
    "properties": {
      "op": { "type": "string", "enum": ["=", "<>", "<", ">", "<=", ">="] },
      "args": {"$ref": "#/$defs/scalarOperands"}
    }
  },
  "scalarOperands": {
    "type": "array",
    "minItems": 2,
    "maxItems": 2,
    "items": {"$ref": "#/$defs/scalarExpression"}
  },
  "scalarExpression": {
    "oneOf": [
      {"$ref": "#/$defs/characterExpression"},
      {"$ref": "#/$defs/numericExpression"}  ,
      {"type": "boolean"}  ,
      {"$ref": "#/$defs/instantInstance"}    ,
      {"$ref": "#/$defs/functionRef"},
      {"$ref": "#/$defs/propertyRef"}
    ]
  },
  "isLikePredicate": {
    "type": "object",
    "required": ["op", "args"],
    "properties": {
      "op"  : { "type": "string", "enum": ["like"] },
      "args": {"$ref": "#/$defs/isLikeOperands"}
    }
  },
  "isLikeOperands": {
    "type": "array",
    "minItems": 2,
    "maxItems": 2,
    "prefixItems": [
      {
        "oneOf": [
          {"$ref": "#/$defs/characterExpression"},
          {"$ref": "#/$defs/propertyRef"        },
          {"$ref": "#/$defs/functionRef"        }
        ]
      },
```

```
      {"$ref": "#/$defs/patternExpression"}
    ]
  },
  "patternExpression": {
    "oneOf": [
      {
        "type": "object",
        "required": ["op", "args"],
        "properties": {
          "op": { "type": "string", "enum": ["casei"] },
          "args": {
            "type": "array",
            "items": {"$ref": "#/$defs/patternExpression"},
            "minItems": 1,
            "maxItems": 1
          }
        }
      },
      {
        "type": "object",
        "required": ["op", "args"],
        "properties": {
          "op": { "type": "string", "enum": ["accenti"] },
          "args": {
            "type": "array",
            "items": {"$ref": "#/$defs/patternExpression"},
            "minItems": 1,
            "maxItems": 1
          }
        }
      },
      {"type": "string"}
    ]
  },
  "isBetweenPredicate": {
    "type": "object",
    "required": ["op", "args"],
    "properties": {
      "op"  : { "type": "string", "enum": ["between"] },
      "args": {"$ref": "#/$defs/isBetweenOperands"}
    }
  },
  "isBetweenOperands": {
    "type": "array",
    "minItems": 3,
    "maxItems": 3,
    "items": {
      "oneOf": [
        {"$ref": "#/$defs/numericExpression"},
        {"$ref": "#/$defs/propertyRef"      },
        {"$ref": "#/$defs/functionRef"      }
```

```
        ]
      }
    },
    "numericExpression": {
      "oneOf": [ {"$ref": "#/$defs/arithmeticExpression"}, {"type": "number"} ]
    },
    "isInListPredicate": {
      "type": "object",
      "required": ["op", "args"],
      "properties": {
        "op"  : { "type": "string", "enum": ["in"] },
        "args": {"$ref": "#/$defs/inListOperands"}
      }
    },
    "inListOperands": {
      "type": "array",
      "minItems": 2,
      "maxItems": 2,
      "prefixItems": [
        {"$ref": "#/$defs/scalarExpression"}                              ,
        { "type": "array", "items": {"$ref": "#/$defs/scalarExpression"} }
      ]
    },
    "isNullPredicate": {
      "type": "object",
      "required": ["op", "args"],
      "properties": {
        "op"  : { "type": "string", "enum": ["isNull"] },
        "args": {"$ref": "#/$defs/isNullOperand"}
      }
    },
    "isNullOperand": {
      "type": "array",
      "minItems": 1,
      "maxItems": 1,
      "items": {
        "oneOf": [
          {"$ref": "#/$defs/characterExpression"},
          {"$ref": "#/$defs/numericExpression"}  ,
          {"$dynamicRef": "#cql2expression"}     ,
          {"$ref": "#/$defs/spatialInstance"}    ,
          {"$ref": "#/$defs/temporalInstance"}   ,
          {"$ref": "#/$defs/propertyRef"}
        ]
      }
    },
    "spatialPredicate": {
      "type": "object",
      "required": ["op", "args"],
      "properties": {
        "op": {
```

```
        "type": "string",
        "enum": [
          "s_contains"  , "s_crosses"   , "s_disjoint"  , "s_equals"    ,
          "s_intersects", "s_overlaps"  , "s_touches"   , "s_within"
        ]
      },
      "args": {"$ref": "#/$defs/spatialOperands"}
    }
  },
  "spatialOperands": {
    "type": "array",
    "minItems": 2,
    "maxItems": 2,
    "items": {
      "oneOf": [
        {"$ref": "#/$defs/spatialInstance"},
        {"$ref": "#/$defs/propertyRef"    },
        {"$ref": "#/$defs/functionRef"    }
      ]
    }
  },
  "temporalPredicate": {
    "type": "object",
    "required": ["op", "args"],
    "properties": {
      "op": {
        "type": "string",
        "enum": [
          "t_after"       , "t_before"      , "t_contains"    ,
          "t_disjoint"    , "t_during"      , "t_equals"      ,
          "t_finishedBy"  , "t_finishes"    , "t_intersects"  ,
          "t_meets"       , "t_metBy"       , "t_overlappedBy",
          "t_overlaps"    , "t_startedBy"   , "t_starts"
        ]
      },
      "args": {"$ref": "#/$defs/temporalOperands"}
    }
  },
  "temporalOperands": {
    "type": "array",
    "minItems": 2,
    "maxItems": 2,
    "items": {
      "oneOf": [
        {"$ref": "#/$defs/temporalInstance"},
        {"$ref": "#/$defs/propertyRef"     },
        {"$ref": "#/$defs/functionRef"     }
      ]
    }
  },
  "arrayPredicate": {
```

```
      "type": "object",
      "required": ["op", "args"],
      "properties": {
        "op": {
          "type": "string",
          "enum": ["a_containedBy", "a_contains", "a_equals", "a_overlaps"]
        },
        "args": {"$ref": "#/$defs/arrayExpression"}
      }
    },
    "arrayExpression": {
      "type": "array",
      "minItems": 2,
      "maxItems": 2,
      "items": {
        "oneOf": [
          {"$ref": "#/$defs/array"      },
          {"$ref": "#/$defs/propertyRef"},
          {"$ref": "#/$defs/functionRef"}
        ]
      }
    },
    "array": {
      "type": "array",
      "items": {
        "oneOf": [
          {"$ref": "#/$defs/characterExpression"},
          {"$ref": "#/$defs/numericExpression"}  ,
          {"$dynamicRef": "#cql2expression"}     ,
          {"$ref": "#/$defs/spatialInstance"}    ,
          {"$ref": "#/$defs/temporalInstance"}   ,
          {"$ref": "#/$defs/array"}              ,
          {"$ref": "#/$defs/propertyRef"}
        ]
      }
    },
    "arithmeticExpression": {
      "type": "object",
      "required": ["op", "args"],
      "properties": {
        "op": {
          "type": "string",
          "enum": ["+", "-", "*", "/", "^", "%", "div"]
        },
        "args": {"$ref": "#/$defs/arithmeticOperands"}
      }
    },
    "arithmeticOperands": {
      "type": "array",
      "minItems": 2,
      "maxItems": 2,
```

```
      "items": {
        "oneOf": [
          {"$ref": "#/$defs/arithmeticExpression"                 },
          {"$ref": "#/$defs/propertyRef"                          },
          {"$ref": "#/$defs/functionRef"                          },
          {                                          "type": "number"}
        ]
      }
    },
    "propertyRef": {
      "type": "object",
      "required": ["property"],
      "properties": { "property": {"type": "string"} }
    },
    "casei": {
      "type": "object",
      "required": ["op", "args"],
      "properties": {
        "op": { "type": "string", "enum": ["casei"] },
        "args": {
          "type": "array",
          "items": {
            "oneOf": [
              {"$ref": "#/$defs/characterExpression"},
              {"$ref": "#/$defs/propertyRef"        },
              {"$ref": "#/$defs/functionRef"        }
            ]
          },
          "minItems": 1,
          "maxItems": 1
        }
      }
    },
    "accenti": {
      "type": "object",
      "required": ["op", "args"],
      "properties": {
        "op": { "type": "string", "enum": ["accenti"] },
        "args": {
          "type": "array",
          "items": {
            "oneOf": [
              {"$ref": "#/$defs/characterExpression"},
              {"$ref": "#/$defs/propertyRef"        },
              {"$ref": "#/$defs/functionRef"        }
            ]
          },
          "minItems": 1,
          "maxItems": 1
        }
      }
```

```
    },
    "characterExpression": {
      "oneOf": [
        {"$ref": "#/$defs/casei"                    },
        {"$ref": "#/$defs/accenti"                  },
        {                           "type": "string"}
      ]
    },
    "functionRef": {
      "type": "object",
      "required": ["op", "args"],
      "properties": {
        "op": {
          "type": "string",
          "not": {
            "enum": [
              "and"           , "or"            , "not"           ,
              "="             , "<>"            , "<"             ,
              ">"             , "<="            , ">="            ,
              "like"          , "between"       , "in"            ,
              "isNull"        , "casei"         , "accenti"       ,
              "s_contains"    , "s_crosses"     , "s_disjoint"    ,
              "s_equals"      , "s_intersects"  , "s_overlaps"    ,
              "s_touches"     , "s_within"      , "t_after"       ,
              "t_before"      , "t_contains"    , "t_disjoint"    ,
              "t_during"      , "t_equals"      , "t_finishedBy"  ,
              "t_finishes"    , "t_intersects"  , "t_meets"       ,
              "t_metBy"       , "t_overlappedBy", "t_overlaps"    ,
              "t_startedBy"   , "t_starts"      , "a_containedBy" ,
              "a_contains"    , "a_equals"      , "a_overlaps"    ,
              "+"             , "-"             , "*"             ,
              "/"             , "^"             , "%"             ,
              "div"
            ]
          }
        },
        "args": {
          "type": "array",
          "items": {
            "oneOf": [
              {"$ref": "#/$defs/characterExpression"},
              {"$ref": "#/$defs/numericExpression"}  ,
              {"$dynamicRef": "#cql2expression"}     ,
              {"$ref": "#/$defs/spatialInstance"}    ,
              {"$ref": "#/$defs/temporalInstance"}   ,
              {"$ref": "#/$defs/array"}              ,
              {"$ref": "#/$defs/propertyRef"}
            ]
          }
        }
      }
    }
```

```
    },
    "spatialInstance": {
      "oneOf": [
        {"$ref": "#/$defs/geometryLiteral"},
        {"$ref": "#/$defs/bboxLiteral"    }
      ]
    },
    "geometryLiteral": {
      "oneOf": [
        {"$ref": "#/$defs/point"             },
        {"$ref": "#/$defs/linestring"        },
        {"$ref": "#/$defs/polygon"           },
        {"$ref": "#/$defs/multipoint"        },
        {"$ref": "#/$defs/multilinestring"   },
        {"$ref": "#/$defs/multipolygon"      },
        {"$ref": "#/$defs/geometrycollection"}
      ]
    },
    "point": {
      "title": "GeoJSON Point",
      "type": "object",
      "required": ["type", "coordinates"],
      "properties": {
        "type": { "type": "string", "enum": ["Point"] },
        "coordinates": {
          "type": "array",
          "minItems": 2,
          "items": {"type": "number"}
        },
        "bbox": { "type": "array", "minItems": 4, "items": {"type": "number"} }
      }
    },
    "linestring": {
      "title": "GeoJSON LineString",
      "type": "object",
      "required": ["type", "coordinates"],
      "properties": {
        "type": { "type": "string", "enum": ["LineString"] },
        "coordinates": {
          "type": "array",
          "minItems": 2,
          "items": {
            "type": "array",
            "minItems": 2,
            "items": {"type": "number"}
          }
        },
        "bbox": { "type": "array", "minItems": 4, "items": {"type": "number"} }
      }
    },
    "polygon": {
```

```
    "title": "GeoJSON Polygon",
    "type": "object",
    "required": ["type", "coordinates"],
    "properties": {
      "type": { "type": "string", "enum": ["Polygon"] },
      "coordinates": {
        "type": "array",
        "items": {
          "type": "array",
          "minItems": 4,
          "items": {
            "type": "array",
            "minItems": 2,
            "items": {"type": "number"}
          }
        }
      },
      "bbox": { "type": "array", "minItems": 4, "items": {"type": "number"} }
    }
  },
  "multipoint": {
    "title": "GeoJSON MultiPoint",
    "type": "object",
    "required": ["type", "coordinates"],
    "properties": {
      "type": { "type": "string", "enum": ["MultiPoint"] },
      "coordinates": {
        "type": "array",
        "items": {
          "type": "array",
          "minItems": 2,
          "items": {"type": "number"}
        }
      },
      "bbox": { "type": "array", "minItems": 4, "items": {"type": "number"} }
    }
  },
  "multilinestring": {
    "title": "GeoJSON MultiLineString",
    "type": "object",
    "required": ["type", "coordinates"],
    "properties": {
      "type": { "type": "string", "enum": ["MultiLineString"] },
      "coordinates": {
        "type": "array",
        "items": {
          "type": "array",
          "minItems": 2,
          "items": {
            "type": "array",
            "minItems": 2,
```

```
            "items": {"type": "number"}
          }
        }
      },
      "bbox": { "type": "array", "minItems": 4, "items": {"type": "number"} }
    }
  },
  "multipolygon": {
    "title": "GeoJSON MultiPolygon",
    "type": "object",
    "required": ["type", "coordinates"],
    "properties": {
      "type": { "type": "string", "enum": ["MultiPolygon"] },
      "coordinates": {
        "type": "array",
        "items": {
          "type": "array",
          "items": {
            "type": "array",
            "minItems": 4,
            "items": {
              "type": "array",
              "minItems": 2,
              "items": {"type": "number"}
            }
          }
        }
      },
      "bbox": { "type": "array", "minItems": 4, "items": {"type": "number"} }
    }
  },
  "geometrycollection": {
    "title": "GeoJSON GeometryCollection",
    "type": "object",
    "required": ["type", "geometries"],
    "properties": {
      "type": { "type": "string", "enum": ["GeometryCollection"] },
      "geometries": {
        "type": "array",
        "minItems": 2,
        "items": {
          "oneOf": [
            {"$ref": "#/$defs/point"           },
            {"$ref": "#/$defs/linestring"      },
            {"$ref": "#/$defs/polygon"         },
            {"$ref": "#/$defs/multipoint"      },
            {"$ref": "#/$defs/multilinestring"},
            {"$ref": "#/$defs/multipolygon"    }
          ]
        }
      }
```

```
    }
  },
  "bboxLiteral": {
    "type": "object",
    "required": ["bbox"],
    "properties": { "bbox": {"$ref": "#/$defs/bbox"} }
  },
  "bbox": {
    "type": "array",
    "oneOf": [
      {"minItems": 4, "maxItems": 4},
      {"minItems": 6, "maxItems": 6}
    ],
    "items": {"type": "number"}
  },
  "temporalInstance": {
    "oneOf": [
      {"$ref": "#/$defs/instantInstance" },
      {"$ref": "#/$defs/intervalInstance"}
    ]
  },
  "instantInstance": {
    "oneOf": [
      {"$ref": "#/$defs/dateInstant"     },
      {"$ref": "#/$defs/timestampInstant"}
    ]
  },
  "dateInstant": {
    "type": "object",
    "required": ["date"],
    "properties": { "date": {"$ref": "#/$defs/dateString"} }
  },
  "timestampInstant": {
    "type": "object",
    "required": ["timestamp"],
    "properties": { "timestamp": {"$ref": "#/$defs/timestampString"} }
  },
  "instantString": {
    "oneOf": [
      {"$ref": "#/$defs/dateString"     },
      {"$ref": "#/$defs/timestampString"}
    ]
  },
  "dateString": {"type": "string", "pattern": "^\\d{4}-\\d{2}-\\d{2}$"},
  "timestampString": {
    "type"   : "string"                                                 ,
    "pattern": "^\\d{4}-\\d{2}-\\d{2}T\\d{2}:\\d{2}:\\d{2}(?:\\.\\d+)?Z$"
  },
  "intervalInstance": {
    "type": "object",
    "required": ["interval"],
```

```
      "properties": { "interval": {"$ref": "#/$defs/intervalArray"} }
    },
    "intervalArray": {
      "type": "array",
      "minItems": 2,
      "maxItems": 2,
      "items": {
        "oneOf": [
          {"$ref": "#/$defs/instantString"}   ,
          { "type": "string", "enum": [".."] },
          {"$ref": "#/$defs/propertyRef"}     ,
          {"$ref": "#/$defs/functionRef"}
        ]
      }
    }
  }
}
```

C.2. OpenAPI 3.0 schema for CQL2

The following document specifies the schema for CQL2 as an OpenAPI 3.0 schema in YAML:

```
---
openapi: 3.0.3
info:
  title: Schema of Common Query Language (CQL2)
  description: 'For use in OpenAPI 3.0 documents.'
  version: '1.0.0-SNAPSHOT'
paths: {}
components:
  schemas:
    booleanExpression:
      oneOf:
        - $ref: '#/components/schemas/andOrExpression'
        - $ref: '#/components/schemas/notExpression'
        - $ref: '#/components/schemas/comparisonPredicate'
        - $ref: '#/components/schemas/spatialPredicate'
        - $ref: '#/components/schemas/temporalPredicate'
        - $ref: '#/components/schemas/arrayPredicate'
        - $ref: '#/components/schemas/functionRef'
        - type: boolean
    andOrExpression:
      type: object
      required:
        - op
        - args
      properties:
        op:
          type: string
```

```
        enum:
          - and
          - or
      args:
        type: array
        minItems: 2
        items:
          $ref: '#/components/schemas/booleanExpression'
  notExpression:
    type: object
    required:
      - op
      - args
    properties:
      op:
        type: string
        enum:
          - not
      args:
        type: array
        minItems: 1
        maxItems: 1
        items:
          $ref: '#/components/schemas/booleanExpression'
  comparisonPredicate:
    oneOf:
      - $ref: '#/components/schemas/binaryComparisonPredicate'
      - $ref: '#/components/schemas/isLikePredicate'
      - $ref: '#/components/schemas/isBetweenPredicate'
      - $ref: '#/components/schemas/isInListPredicate'
      - $ref: '#/components/schemas/isNullPredicate'
  binaryComparisonPredicate:
    type: object
    required:
      - op
      - args
    properties:
      op:
        type: string
        enum:
          - '='
          - <>
          - <
          - '>'
          - <=
          - '>='
      args:
        $ref: '#/components/schemas/scalarOperands'
  scalarOperands:
    type: array
    minItems: 2
```

```
    maxItems: 2
    items:
      $ref: '#/components/schemas/scalarExpression'
  scalarExpression:
    oneOf:
      - $ref: '#/components/schemas/characterExpression'
      - $ref: '#/components/schemas/numericExpression'
      - type: 'boolean'
      - $ref: '#/components/schemas/instantInstance'
      - $ref: '#/components/schemas/functionRef'
      - $ref: '#/components/schemas/propertyRef'
  isLikePredicate:
    type: object
    required:
      - op
      - args
    properties:
      op:
        type: string
        enum:
          - like
      args:
        $ref: '#/components/schemas/isLikeOperands'
  isLikeOperands:
    type: array
    minItems: 2
    maxItems: 2
    items:
      oneOf:
        - oneOf:
            - $ref: '#/components/schemas/characterExpression'
            - $ref: '#/components/schemas/propertyRef'
            - $ref: '#/components/schemas/functionRef'
        - $ref: '#/components/schemas/patternExpression'
  patternExpression:
    oneOf:
      - type: object
        required:
          - op
          - args
        properties:
          op:
            type: string
            enum:
              - casei
          args:
            type: array
            items:
              $ref: '#/components/schemas/patternExpression'
            minItems: 1
            maxItems: 1
```

```
      - type: object
        required:
          - op
          - args
        properties:
          op:
            type: string
            enum:
              - accenti
          args:
            type: array
            items:
              $ref: '#/components/schemas/patternExpression'
            minItems: 1
            maxItems: 1
      - type: string
  isBetweenPredicate:
    type: object
    required:
      - op
      - args
    properties:
      op:
        type: string
        enum:
          - between
      args:
        $ref: '#/components/schemas/isBetweenOperands'
  isBetweenOperands:
    type: array
    minItems: 3
    maxItems: 3
    items:
      oneOf:
        - $ref: '#/components/schemas/numericExpression'
        - $ref: '#/components/schemas/propertyRef'
        - $ref: '#/components/schemas/functionRef'
  numericExpression:
    oneOf:
      - $ref: '#/components/schemas/arithmeticExpression'
      - type: number
  isInListPredicate:
    type: object
    required:
      - op
      - args
    properties:
      op:
        type: string
        enum:
          - in
```

```
      args:
        $ref: '#/components/schemas/inListOperands'
  inListOperands:
    type: array
    minItems: 2
    maxItems: 2
    items:
      oneOf:
        - $ref: '#/components/schemas/scalarExpression'
        - type: array
          items:
            $ref: '#/components/schemas/scalarExpression'
  isNullPredicate:
    type: object
    required:
      - op
      - args
    properties:
      op:
        type: string
        enum:
          - isNull
      args:
        $ref: '#/components/schemas/isNullOperand'
  isNullOperand:
    type: array
    minItems: 1
    maxItems: 1
    items:
      oneOf:
        - $ref: '#/components/schemas/characterExpression'
        - $ref: '#/components/schemas/numericExpression'
        - $ref: '#/components/schemas/booleanExpression'
        - $ref: '#/components/schemas/spatialInstance'
        - $ref: '#/components/schemas/temporalInstance'
        - $ref: '#/components/schemas/propertyRef'
  spatialPredicate:
    type: object
    required:
      - op
      - args
    properties:
      op:
        type: string
        enum:
          - s_contains
          - s_crosses
          - s_disjoint
          - s_equals
          - s_intersects
          - s_overlaps
```

```
          - s_touches
          - s_within
      args:
        $ref: '#/components/schemas/spatialOperands'
  spatialOperands:
    type: array
    minItems: 2
    maxItems: 2
    items:
      oneOf:
        - $ref: '#/components/schemas/spatialInstance'
        - $ref: '#/components/schemas/propertyRef'
        - $ref: '#/components/schemas/functionRef'
  temporalPredicate:
    type: object
    required:
      - op
      - args
    properties:
      op:
        type: string
        enum:
          - t_after
          - t_before
          - t_contains
          - t_disjoint
          - t_during
          - t_equals
          - t_finishedBy
          - t_finishes
          - t_intersects
          - t_meets
          - t_metBy
          - t_overlappedBy
          - t_overlaps
          - t_startedBy
          - t_starts
      args:
        $ref: '#/components/schemas/temporalOperands'
  temporalOperands:
    type: array
    minItems: 2
    maxItems: 2
    items:
      oneOf:
        - $ref: '#/components/schemas/temporalInstance'
        - $ref: '#/components/schemas/propertyRef'
        - $ref: '#/components/schemas/functionRef'
  arrayPredicate:
    type: object
    required:
```

```
      - op
      - args
    properties:
      op:
        type: string
        enum:
          - a_containedBy
          - a_contains
          - a_equals
          - a_overlaps
      args:
        $ref: '#/components/schemas/arrayExpression'
  arrayExpression:
    type: array
    minItems: 2
    maxItems: 2
    items:
      oneOf:
        - $ref: '#/components/schemas/array'
        - $ref: '#/components/schemas/propertyRef'
        - $ref: '#/components/schemas/functionRef'
  array:
    type: array
    items:
      oneOf:
        - $ref: '#/components/schemas/characterExpression'
        - $ref: '#/components/schemas/numericExpression'
        - $ref: '#/components/schemas/booleanExpression'
        - $ref: '#/components/schemas/spatialInstance'
        - $ref: '#/components/schemas/temporalInstance'
        - $ref: '#/components/schemas/array'
        - $ref: '#/components/schemas/propertyRef'
  arithmeticExpression:
    type: object
    required:
      - op
      - args
    properties:
      op:
        type: string
        enum:
          - +
          - '-'
          - '*'
          - /
          - ^
          - '%'
          - div
      args:
        $ref: '#/components/schemas/arithmeticOperands'
  arithmeticOperands:
```

```
      type: array
      minItems: 2
      maxItems: 2
      items:
        oneOf:
          - $ref: '#/components/schemas/arithmeticExpression'
          - $ref: '#/components/schemas/propertyRef'
          - $ref: '#/components/schemas/functionRef'
          - type: number
    propertyRef:
      type: object
      required:
        - property
      properties:
        property:
          type: string
    casei:
      type: object
      required:
        - op
        - args
      properties:
        op:
          type: string
          enum:
            - casei
        args:
          type: array
          items:
            oneOf:
              - $ref: '#/components/schemas/characterExpression'
              - $ref: '#/components/schemas/propertyRef'
              - $ref: '#/components/schemas/functionRef'
          minItems: 1
          maxItems: 1
    accenti:
      type: object
      required:
        - op
        - args
      properties:
        op:
          type: string
          enum:
            - accenti
        args:
          type: array
          items:
            oneOf:
              - $ref: '#/components/schemas/characterExpression'
              - $ref: '#/components/schemas/propertyRef'
```

```
            - $ref: '#/components/schemas/functionRef'
        minItems: 1
        maxItems: 1
  characterExpression:
    oneOf:
      - $ref: '#/components/schemas/casei'
      - $ref: '#/components/schemas/accenti'
      - type: string
  functionRef:
    type: object
    required:
      - op
      - args
    properties:
      op:
        type: string
        not:
          enum:
            - and
            - or
            - not
            - '='
            - <>
            - <
            - '>'
            - <=
            - '>='
            - like
            - between
            - in
            - isNull
            - casei
            - accenti
            - s_contains
            - s_crosses
            - s_disjoint
            - s_equals
            - s_intersects
            - s_overlaps
            - s_touches
            - s_within
            - t_after
            - t_before
            - t_contains
            - t_disjoint
            - t_during
            - t_equals
            - t_finishedBy
            - t_finishes
            - t_intersects
            - t_meets
```

```
            - t_metBy
            - t_overlappedBy
            - t_overlaps
            - t_startedBy
            - t_starts
            - a_containedBy
            - a_contains
            - a_equals
            - a_overlaps
            - +
            - '-'
            - '*'
            - /
            - ^
            - '%'
            - div
      args:
        type: array
        items:
          oneOf:
            - $ref: '#/components/schemas/characterExpression'
            - $ref: '#/components/schemas/numericExpression'
            - $ref: '#/components/schemas/booleanExpression'
            - $ref: '#/components/schemas/spatialInstance'
            - $ref: '#/components/schemas/temporalInstance'
            - $ref: '#/components/schemas/array'
            - $ref: '#/components/schemas/propertyRef'
  spatialInstance:
    oneOf:
      - $ref: '#/components/schemas/geometryLiteral'
      - $ref: '#/components/schemas/bboxLiteral'
  geometryLiteral:
    oneOf:
      - $ref: '#/components/schemas/point'
      - $ref: '#/components/schemas/linestring'
      - $ref: '#/components/schemas/polygon'
      - $ref: '#/components/schemas/multipoint'
      - $ref: '#/components/schemas/multilinestring'
      - $ref: '#/components/schemas/multipolygon'
      - $ref: '#/components/schemas/geometrycollection'
  point:
    title: GeoJSON Point
    type: object
    required:
      - type
      - coordinates
    properties:
      type:
        type: string
        enum:
          - Point
```

```
      coordinates:
        type: array
        minItems: 2
        items:
          type: number
      bbox:
        type: array
        minItems: 4
        items:
          type: number
  linestring:
    title: GeoJSON LineString
    type: object
    required:
      - type
      - coordinates
    properties:
      type:
        type: string
        enum:
          - LineString
      coordinates:
        type: array
        minItems: 2
        items:
          type: array
          minItems: 2
          items:
            type: number
      bbox:
        type: array
        minItems: 4
        items:
          type: number
  polygon:
    title: GeoJSON Polygon
    type: object
    required:
      - type
      - coordinates
    properties:
      type:
        type: string
        enum:
          - Polygon
      coordinates:
        type: array
        items:
          type: array
          minItems: 4
          items:
```

```
            type: array
            minItems: 2
            items:
              type: number
      bbox:
        type: array
        minItems: 4
        items:
          type: number
  multipoint:
    title: GeoJSON MultiPoint
    type: object
    required:
      - type
      - coordinates
    properties:
      type:
        type: string
        enum:
          - MultiPoint
      coordinates:
        type: array
        items:
          type: array
          minItems: 2
          items:
            type: number
      bbox:
        type: array
        minItems: 4
        items:
          type: number
  multilinestring:
    title: GeoJSON MultiLineString
    type: object
    required:
      - type
      - coordinates
    properties:
      type:
        type: string
        enum:
          - MultiLineString
      coordinates:
        type: array
        items:
          type: array
          minItems: 2
          items:
            type: array
            minItems: 2
```

```
            items:
              type: number
      bbox:
        type: array
        minItems: 4
        items:
          type: number
  multipolygon:
    title: GeoJSON MultiPolygon
    type: object
    required:
      - type
      - coordinates
    properties:
      type:
        type: string
        enum:
          - MultiPolygon
      coordinates:
        type: array
        items:
          type: array
          items:
            type: array
            minItems: 4
            items:
              type: array
              minItems: 2
              items:
                type: number
      bbox:
        type: array
        minItems: 4
        items:
          type: number
  geometrycollection:
    title: GeoJSON GeometryCollection
    type: object
    required:
      - type
      - geometries
    properties:
      type:
        type: string
        enum:
          - GeometryCollection
      geometries:
        type: array
        minItems: 2
        items:
          oneOf:
```

```
            - $ref: '#/components/schemas/point'
            - $ref: '#/components/schemas/linestring'
            - $ref: '#/components/schemas/polygon'
            - $ref: '#/components/schemas/multipoint'
            - $ref: '#/components/schemas/multilinestring'
            - $ref: '#/components/schemas/multipolygon'
    bboxLiteral:
      type: object
      required:
        - bbox
      properties:
        bbox:
          $ref: '#/components/schemas/bbox'
    bbox:
      type: array
      oneOf:
        - minItems: 4
          maxItems: 4
        - minItems: 6
          maxItems: 6
      items:
        type: number
    temporalInstance:
      oneOf:
        - $ref: '#/components/schemas/instantInstance'
        - $ref: '#/components/schemas/intervalInstance'
    instantInstance:
      oneOf:
        - $ref: '#/components/schemas/dateInstant'
        - $ref: '#/components/schemas/timestampInstant'
    dateInstant:
      type: object
      required:
        - date
      properties:
        date:
          $ref: '#/components/schemas/dateString'
    timestampInstant:
      type: object
      required:
        - timestamp
      properties:
        timestamp:
          $ref: '#/components/schemas/timestampString'
    instantString:
      oneOf:
        - $ref: '#/components/schemas/dateString'
        - $ref: '#/components/schemas/timestampString'
    dateString:
      type: string
      pattern: ^\d{4}-\d{2}-\d{2}$
```

```yaml
    timestampString:
      type: string
      pattern: ^\d{4}-\d{2}-\d{2}T\d{2}:\d{2}:\d{2}(?:\.\d+)?Z$
    intervalInstance:
      type: object
      required:
        - interval
      properties:
        interval:
          $ref: '#/components/schemas/intervalArray'
    intervalArray:
      type: array
      minItems: 2
      maxItems: 2
      items:
        oneOf:
          - $ref: '#/components/schemas/instantString'
          - type: string
            enum:
              - ..
          - $ref: '#/components/schemas/propertyRef'
          - $ref: '#/components/schemas/functionRef'
```

Annex D: Revision History

Date	Release	Editor	Primary clauses modified	Description
2021-09-27	1.0.0-SNAPSHOT	C. Portele	all	split from OGC API - Features - Part 3: Filtering
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 • #902 CQL2 JSON: fix schema and some examples; • #904 CQL2 Text: update spatialLiteral rule; • #915 simplify `scalarExpression`; • #907 update title of the `basic-spatial-functions-plus` requirements class.
2024-06-29	1.0.0	C. Portele, P. Vretanos, G. Hobona	all	prepare for publication

Annex E: Bibliography

- Internet Engineering Task Force (IETF). RFC 5234: **Augmented BNF for Syntax Specifications: ABNF** [online]. Edited by D. Crocker, P. Overell. 2008. Available at https://www.rfc-editor.org/rfc/rfc5234.html
- Internet Engineering Task Force (IETF). draft-bhutton-json-schema-validation-01: **JSON Schema Validation: A Vocabulary for Structural Validation of JSON** [online]. Edited by A. Wright, H. Andrews, B. Hutton. 2020. Available at https://json-schema.org/draft/2020-12/json-schema-validation
- 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). **OGC API - Features - Part 3: Filtering** [online]. Edited by P. Vretanos, C. Portele. Available at https://docs.ogc.org/is/19-079r2/19-079r2.html
- PostGIS. **Dimensionally Extended 9-Intersection Model** [online]. Available at https://postgis.net/workshops/postgis-intro/de9im.html.
- Wikipedia. **DE-9IM** [online]. Available at https://en.wikipedia.org/wiki/DE-9IM.