OGC GeoSPARQL - A Geographic Query Language for RDF Data

Table of Contents

i. Preface 5
ii. Submitting organizations 6
  iii. Submission contact points 6
iv. Revision history 7
  Major changes between versions 1.0 and 1.1 8
v. Changes to the OGC® Abstract Specification 11
Foreword 12
Introduction 13
  RDF 13
  SPARQL 13
  GeoSPARQL Standard structure 14
OGC GeoSPARQL – A Geographic Query Language for RDF Data 17
  1. Scope 18
  2. Conformance 19
  3. Normative references 21
  4. Terms and definitions 22
    4.1. Semantic Web 22
    4.2. Spatial 22
  5. Conventions 24
    5.1. Symbols and abbreviated terms 24
    5.2. Namespaces 24
    5.3. Placeholder IRIs 25
    5.4. RDF Serializations 25
  6. Core 26
    6.1. SPARQL 26
    6.2. Classes 27
    6.3. Standard Properties for geo:SpatialObject 29
    6.4. Standard Properties for geo:Feature 34
  7. Topology Vocabulary Extension (relation_family) 37
    7.1. Parameters 37
    7.2. Simple Features Relation Family (relation_family=Simple Features) 37
    7.3. Egenhofer Relation Family (relation_family=Egenhofer) 38
    7.4. RCC8 Relation Family (relation_family=RCC8) 39
    7.5. Equivalent RCC8, Egenhofer and Simple Features Topological Relations 40
  8. Geometry Extension (serialization, version) 41
    8.1. Rationale 41
    8.2. GeoSPARQL and Simple Features (SFA-CA) 41
    8.3. Recommendation for specification of units of measurement 42

8.4. Influence of Coordinate Reference Systems on geometric computations . . . 42
8.5. Parameters . . . 43
8.6. Geometry Class . . . 43
8.7. Standard Properties for geo:Geometry . . . 45
8.8. Geometry Serializations . . . 49
8.9. Non-topological Query Functions . . . 58
8.10. Spatial Aggregate Functions . . . 66
9. Geometry Topology Extension (relation_family, serialization, version) . . . 69
9.1. Parameters . . . 69
9.2. Common Query Functions . . . 69
9.3. Simple Features Relation Family (relation_family=Simple Features) . . . 70
9.4. Egenhofer Relation Family (relation_family=Egenhofer) . . . 71
9.5. Requirements for RCC8 Relation Family (relation_family=RCC8) . . . 71
10. RDFS Entailment Extension (relation_family, serialization, version) . . . 73
10.1. Parameters . . . 73
10.2. Common Requirements . . . 73
10.3. WKT Serialization (serialization=WKT) . . . 73
10.4. GML Serialization (serialization=GML) . . . 74
11. Query Rewrite Extension (relation_family, serialization, version) . . . 76
11.1. Parameters . . . 77
11.2. Simple Features Relation Family (relation_family=Simple Features) . . . 77
11.3. Egenhofer Relation Family (relation_family=Egenhofer) . . . 77
11.4. RCC8 Relation Family (relation_family=RCC8) . . . 78
11.5. Special Considerations . . . 78
12. Future Work . . . 80
Annex A - Abstract Test Suite (normative) . . . 81
A.0 Overview . . . 82
A.1 Conformance Class: Core . . . 83
A.1.1 SPARQL . . . 83
A.1.2 RDF Classes & Properties . . . 83
A.2 Conformance Class: Topology Vocabulary Extension (relation_family) . . . 85
A.2.1 relation_family = Simple Features . . . 85
A.2.2 relation_family = Egenhofer . . . 85
A.2.3 relation_family = RCC8 . . . 85
A.3 Conformance Class: Geometry Extension (serialization, version) . . . 87
A.3.1 Tests for all Serializations . . . 87
A.3.2 serialization = WKT . . . 89
A.3.3 serialization = GML . . . 91
A.3.4 serialization = GEOJSON . . . 92
A.3.5 serialization = KML . . . 94
A.3.6 serialization = DGGS . . . 95

A.4 Conformance Class: Geometry Topology Extension (relation_family, serialization, version) . 97
A.4.1 Tests for all relation families . 97
A.4.2 relation_family = Simple Features . 97
A.4.3 relation_family = Egenhofer . 97
A.4.4 relation_family = RCC8 . 98
A.5 Conformance Class: RDFS Entailment Extension (relation_family, serialization, version) . 99
A.5.1 Tests for all implementations . 99
A.5.2 serialization = WKT . 99
A.5.3 serialization = GML . 99
A.6 Conformance Class: Query Rewrite Extension (relation_family, serialization, version) . 101
A.6.1 relation_family = Simple Features . 101
A.6.2 relation_family = Egenhofer . 101
A.6.3 relation_family = RCC8 . 102
Annex B - Functions Summary (normative) . 103
B.0 Overview . 104
B.1 Functions Summary Table . 105
B.2 GeoSPARQL to SFA Functions Mapping . 112
Annex C - GeoSPARQL Examples (informative) . 114
C.0 Overview . 115
C.1 RDF Examples . 116
C.1.1 Classes . 116
C.1.2 Properties . 124
C.2 Example SPARQL Queries & Rules . 130
C.2.1 Example Data . 130
C.2.2 Example Queries . 132
C.2.3 Example Rule Application . 135
C.2.4 Example Geometry Serialization Conversion Functions . 137
Annex D - Usage of SHACL shapes (informative) . 138
D.0 Overview . 139
D.1 Tools . 140
D.2 Scope of SHACL Shapes provided with GeoSPARQL . 141
D.3 Table of SHACL Shapes . 142
Annex E - Alignments (informative) . 148
E.0 Overview . 149
E.1 ISA Programme Location Core Vocabulary (LOCN) . 150
E.2 WGS84 Geo Positioning: an RDF vocabulary (POS) . 152
E.3 Geonames Ontology (GN) . 153
E.4 NeoGeo Vocabulary . 154
E.5 Juso Ontology . 156
E.6 Time Ontology in OWL (TIME) . 157
E.7 schema.org . 159

E.8 Semantic Sensor Network Ontology (SSN) . . . 160
E.9 DCMI Metadata Terms (DCTERMS) . . . 161
E.10 The Provenance Ontology (PROV) . . . 162
E.11 WikiData . . . 164
E.12 OpenStreetMap Ontologies . . . 168
E.12.1 LinkedGeoData . . . 168
E.12.2 OpenStreetMap RDF (Sophox) . . . 168
E.12.3 Routable Tiles Ontology . . . 169
E.13 Ordnance Survey UK Spatial Ontology . . . 170
E.14 CIDOC CRM Geo . . . 171
E.15 Basic Formal Ontology (BFO) . . . 173
Annex F - CQL / GeoSPARQL Mapping (informative) . . . 175
F.0 Overview . . . 176
F.1 Accessing spatial Features in a SPARQL endpoint . . . 177
F.2 Mappings from CQL2 statements to GeoSPARQL queries . . . 179
F.2.1 Query Parameters . . . 179
F.2.2 Literal Values . . . 179
F.2.3 Property references . . . 180
F.2.4 Comparison Predicates . . . 180
F.2.5 Spatial Operators . . . 181
F.2.6 Temporal Operators . . . 181
F.3 Mappings from Simple Features for SQL . . . 183
Bibliography . . . 187

```
<style>
  .imageblock > .title {
    text-align: inherit;
  }
</style>
```

Open Geospatial Consortium

Submission Date: <yyyy-mm-dd>

Approval Date: <yyyy-mm-dd>

Publication Date: <yyyy-mm-dd>

External identifier of this OGC® document: http://www.opengis.net/doc/IS/geosparql/1.1

Internal reference number of this OGC® document: YY-DOC

Version: 1.1

Category: OGC® Implementation Specification

Editors: Nicholas J. Car, Timo Homburg, Matthew Perry, John Herring, Frans Knibbe, Simon J.D. Cox, Joseph Abhayaratna and Mathias Bonduel

Contributors: Paul J. Cripps, Krzysztof Janowicz

OGC GeoSPARQL - A Geographic Query Language for RDF Data

Warning

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

Document type: OGC® Standard
Document subtype: Encoding
Document stage: Draft
Document language: English

License Agreement

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

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

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

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

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

Except as contained in this notice, the name of LICENSOR or of any other holder of a copyright in all or part of the Intellectual Property shall not be used in advertising or otherwise to promote the sale, use or other dealings in this Intellectual Property without prior written authorization of LICENSOR or such copyright holder. LICENSOR is and shall at all times be the sole entity that may authorize you or any third party to use certification marks, trademarks or other special designations to indicate compliance with any LICENSOR standards or specifications.

This Agreement is governed by the laws of the Commonwealth of Massachusetts. The application to this Agreement of the United Nations Convention on Contracts for the International Sale of Goods is hereby expressly excluded. In the event any provision of this Agreement shall be deemed unenforceable, void or invalid, such provision shall be modified so as to make it valid and enforceable, and as so modified the entire Agreement shall remain in full force and effect. No decision, action or inaction by LICENSOR shall be construed to be a waiver of any rights or remedies available to it.

None of the Intellectual Property or underlying information or technology may be downloaded or otherwise exported or reexported in violation of U.S. export laws and regulations. In addition, you are responsible for complying with any local laws in your jurisdiction which may impact your right to import, export or use the

Intellectual Property, and you represent that you have complied with any regulations or registration procedures required by applicable law to make this license enforceable.

i. Preface

The GeoSPARQL standard defines:

- a formal *profile*
- **this specification document**
  - a core RDF/OWL ontology for geographic information representation
- a set of SPARQL extension functions
- a Functions & Rules vocabulary, derived from the ontology
- a Simple Features geometry types vocabulary
- a set of RIF rules, and
- SHACL shapes for RDF data validation

This document has the role of *specification* and authoritatively defines many of the standard's elements, including the ontology classes and properties, SPARQL functions and function and rule vocabulary concepts. Complete descriptions of the standard's parts and their roles are given in the Introduction in the section GeoSPARQL Standard structure.

ii. Submitting organizations

The following organizations submitted this Implementation Specification to the Open Geospatial Consortium Inc.:

a. Australian Bureau of Meteorology
b. Bentley Systems, Inc.
c. CSIRO
d. Defence Geospatial Information Working Group (DGIWG)
e. GeoConnections - Natural Resources Canada
f. Interactive Instruments GmbH
g. GeoScape Australia
h. Mainz University Of Applied Sciences
i. Oracle America
j. Ordnance Survey
k. Raytheon Company
l. SURROUND Australia Pty Ltd.
m. Traverse Technologies, Inc.
n. US Geological Survey (USGS)

iii. Submission contact points

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

Contact	**Company**
Matthew Perry	Oracle America
John Herring	Oracle America
Nicholas J. Car	SURROUND Australia Pty Ltd.
Timo Homburg	Mainz University Of Applied Sciences
Joseph Abhayaratna	GeoScape Australia
Simon J.D. Cox	CSIRO

iv. Revision history

Date	Release	Author	Paragraph modified	Description
27 Oct. 2009	Draft	Matthew Perry	Clause 6	Technical Draft
11 Nov. 2009	Draft	John R. Herring	All	Creation
06 Jan. 2010	Draft	John R. Herring	All	Comment responses
30 March 2010	Draft	Matthew Perry	All	Comment responses
26 Oct. 2010	Draft	Matthew Perry	All	Revision based on working group discussion
28 Jan. 2011	Draft	Matthew Perry	All	Revision based on working group discussion
18 April 2011	Draft	Matthew Perry	All	Restructure with multiple conformance classes
02 May 2011	Draft	Matthew Perry	Clause 6 and Clause 8	Move Geometry Class from core to geometryExtension
05 May 2011	Draft	Matthew Perry	All	Update URIs
13 Jan. 2012	Draft	Matthew Perry	All	Revision based on Public RFC
16 April 2012	Draft	Matthew Perry	All	Revision based on adoption vote comments
19 July 2012	1.0	Matthew Perry	All	Revision of URIs based on OGC Naming Authority recommendations
09 Oct. 2020	1.1 Draft	Joseph Abhayaratna	All	Establishment of the 1.1 Specification

Date	Release	Author	Paragraph modified	Description
10 Oct. 2020 to 15 Aug. 2021	1.1 Draft	GeoSPARQL 1.1 SWG	All	Addition of GeoSPARQL 1.1 elements

Major changes between versions 1.0 and 1.1

Version 1.1 of GeoSPARQL was released approximately 9 years after version 1.0. It contains no breaking changes to 1.0, but does contain additions: whole new profile resources, new ontology elements and new functions. The major changes are given in the tables below.

These new profile resources are resources - documents - that are separate from this specification. The new *profile defintion* lists all the GeoSPARQL 1.1 resources.

New resource	Location
Profile definition	http://www.opengis.net/def/geosparql
GeoSPARQL Rules in RIF	http://www.opengis.net/def/geosparql-rifrules
RDF validation file	http://www.opengis.net/def/geosparql-shapes

These new ontology elements and new functions are normatively defined in this specification document.

New element	Section
Classes	
Spatial Object Collection class	Section 6.2.3
Feature Collection class	Section 6.2.4
Geometry Collection class	Section 8.6.2
Spatial Object Properties	
hasSize	Section 6.3.1
hasMetricSize	Section 6.3.2
hasLength	Section 6.3.3
hasMetricLength	Section 6.3.4
hasPerimeterLength	Section 6.3.5
hasMetricPerimeterLength	Section 6.3.6
hasArea	Section 6.3.7
hasMetricArea	Section 6.3.8
hasVolume	Section 6.3.9

New element	Section
hasMetricVolume	Section 6.3.10
Feature Properties	
hasBoundingBox	Section 6.4.3
hasCentroid	Section 6.4.4
Geometry Serializations	
geoJSONLiteral	Section 8.8.3.1
asGeoJSON	Section 8.8.3.2
asGeoJSON function	Section 8.8.3.3
kmlLiteral	Section 8.8.4.1
asKML	Section 8.8.4.2
asKML function	Section 8.8.4.3
dggsLiteral	Section 8.8.5.1
asDGGS	Section 8.8.5.2
asDGGS function	Section 8.8.5.3
Non-topological Query Functions	
area	Section 8.9.2
coordinateDimension	Section 8.9.7
dimension	Section 8.9.9
geometryN	Section 8.9.13
geometryType	Section 8.9.14
is3D	Section 8.9.17
isEmpty	Section 8.9.18
isMeasured	Section 8.9.19
isSimple	Section 8.9.20
length	Section 8.9.21
maxX	Section 8.9.22
maxY	Section 8.9.23
maxZ	Section 8.9.24
minX	Section 8.9.25
minY	Section 8.9.26
minZ	Section 8.9.27
numGeometries	[Function: geof:]
projectTo	Section 8.9.29

New element	Section
spatialDimension	Section 8.9.30
transform	Section 8.9.32
Spatial Aggregate Functions	
aggBoundingBox	Section 8.10.1
aggBoundingCircle	Section 8.10.2
aggCentroid	Section 8.10.3
aggConcatLines	Section 8.10.4
aggConcaveHull	Section 8.10.5
aggUnion	Section 8.10.7

v. Changes to the OGC® Abstract Specification

The OGC® Abstract Specification does not require changes to accommodate this OGC® standard.

Foreword

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. Open Geospatial Consortium shall not be held responsible for identifying any or all such patent rights. However, to date, no such rights have been claimed or identified.

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 specification set forth in this document, and to provide supporting documentation.

Introduction

The W3C Semantic Web Activity is defining a collection of technologies that enables a “web of data” where information is easily shared and reused across applications. Some key pieces of this technology stack are the RDF (Resource Description Framework) data model [RDF], [RDFS], the OWL Web Ontology Language [OWL2] and the SPARQL protocol and RDF query language [SPARQL].

RDF

RDF is, among other things, a data model built on edge-node "graphs." Each link in a graph consists of three things (with many aliases depending on the mapping from other types of data models):

- Subject (start node, instance, entity, feature)
- Predicate (verb, property, attribute, relation, member, link, reference)
- Object (value, end node, non-literal values can be used as a Subject)

Any of the three values in a triple can be represented with a Internationalized Resource Identifier (IRI) [IETF3987], which globally and uniqueliy identifies the resource referenced. IRIs are an extension to Universal Resource Identifiers (URIs) that allow for non-ASCII characters. In addition to functioning as identifiers, IRIs are usually, but not necissarily, resolvable which means a person or machine can "dereference" them (*click on them* or otherwise action them) and be taken to more information about the resource, perhaps in a web browser.

Subjects and objects within an RDF triple are called nodes and can also be be represented with a blank node (a local identifier with meaning outside the graph it is defined within). Objects can further be represented with a literal value. Basic literal values in RDF are those used in XML [XSD2] but the basic types can be extended for specialised purposes and in this specification are, for geometry data. The figure below shows a basic triple.

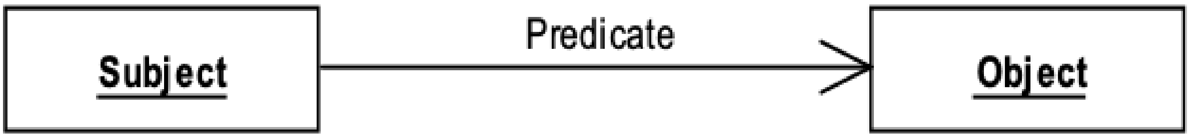

Figure 1. An RDF Triple

Note that the same node may be a subject in some triples, and an object in others.

Almost all data can be presented or represented in RDF. In particular, it is an easy match to the (feature-instance-by-id, attribute, value) tuples of the General Feature Model [ISO19109], and for the relational model as (table primary key, column, value).

SPARQL

From [SPARQL]:

> SPARQL … is a set of specifications that provide languages and protocols to query and manipulate RDF graph content on the Web or in an RDF store.

and, from Wikipedia[1]:

> SPARQL (pronounced "sparkle" /□sp□□k□l/, a recursive acronym for SPARQL Protocol and RDF Query Language) is an RDF query language - that is, a semantic query language for databases — able to retrieve and manipulate data stored in Resource Description Framework (RDF) format. It was made a standard by the RDF Data Access Working Group (DAWG) of the World Wide Web Consortium, and is recognized as one of the key technologies of the semantic web. On 15 January 2008, SPARQL 1.0 was acknowledged by W3C as an official recommendation, and SPARQL 1.1 in March, 2013.

SPARQL queries work on RDF representations of data by finding patterns that match templates in the query, in effect finding information graphs in the RDF data based on the templates and filters (constraints on nodes and edges) expressed in the query. This query template is represented in the SPARQL query by a set of parameterized “query variables” appearing in a sequence of RDF triples and filters. If the query processor finds a set of triples in the data (converted to an RDF graph in some predetermined standard manner) then the values that the “query variables” take on in those triples become a solution to the query request. The values of the variables are returned in the query result in a format based on the “SELECT” clause of the query (similar to SQL).

In addition to predicates defined in this manner, the SPARQL query may contain filter functions that can be used to further constrain the query. Several mechanisms are available to extend filter functions to allow for predicates calculated directly on data values. The SPARQL specification [SPARQL] in section 17.6[2] describes the mechanism for invocation of such a filter function.

The OGC GeoSPARQL standard supports representing and querying geospatial data on the Semantic Web. GeoSPARQL defines a vocabulary for representing geospatial data in RDF, and it defines extensions to the SPARQL query language for processing geospatial data.

GeoSPARQL Standard structure

The GeoSPARQL standard comprises multiple parts, or *profile resources*. The comprehensive listing of them is given not here but in the standard’s *profile definition*, see http://www.opengis.net/def/geosparql. Here is an overview of the major parts:

1. *profile definition*
    - http://www.opengis.net/def/geosparql
    - formally defined as an ontology, defined according to the *Profiles Vocabulary* [PROF]
    - this relates the parts in the standard together, provides access to them, and declares dependencies on other standards
2. **specification document**
    - **http://www.opengis.net/doc/IS/geosparql/1.1**
    - **defines many of the standard’s parts**
    - **includes normative RDF/OWL [RDF],[OWL2] ontology element definitions,**

conformance requirements and function signatures

    - based on the General Feature Model [ISO19109], Simple Features [ISO19125-1] and SQL MM [ISO13249]

  - **also includes non-normative examples and mappings to other modelling and function systems**

3. domain model RDF/OWL [RDF],[OWL2] ontology
   - http://www.opengis.net/ont/geosparql
   - for geographic information representation
   - based on the General Feature Model [ISO19109], Simple Features Access [ISO19125-1], Feature Geometry [ISO19107] and SQL MM [ISO13249]
   - defined within the specification document and delivered in RDF also
4. Functions & Rules vocabulary
   - http://www.opengis.net/def/geosparql/funcsrules
   - derived from the ontology
   - presented as a [SKOS] taxonomy
5. Simple Features vocabulary
   - http://www.opengis.net/ont/sf
   - derived from Simple Features Access [ISO19125-1]'s class model
   - presented as an Section 4.1.3 ontology
6. SPARQL [SPARQL] extension functions
   - defined within this specification document
7. RIF [RIFCORE] rules
   - http://www.opengis.net/def/geosparql/rifrules
   - templated within the specification document
   - also delivered as a RIF document also
8. RDF data validator
   - http://www.opengis.net/def/geosparql/validator
   - pefined using SHACL [SHACL]
   - presented within a single RDF file

This specification document follows a modular design and contains the following components:

- a *core* component defining the top-level RDFS/OWL classes for spatial objects
- a *topology vocabulary* component defining the RDF properties for asserting and querying topological relations between spatial objects
- a *geometry* component defines RDFS data types for serializing geometry data, geometry-related RDF properties, and non-topological spatial query functions for geometry objects

- a *geometry topology* component defining topological query functions
- an *RDFS entailment* component defining mechanisms for matching implicit RDF triples that are derived based on RDF and RDFS semantics
- a *query rewrite* component defining rules for transforming a simple triple pattern that tests a topological relation between two features into an equivalent query involving concrete geometries and topological query functions

Each of these specification components forms a *requirements class* (a set of requirements) for GeoSPARQL. Implementations can provide various levels of functionality by choosing which requirements classes to support. For example, a system based purely on qualitative spatial reasoning may support only the core and topological vocabulary components.

In addition, GeoSPARQL is designed to accommodate systems based on qualitative spatial reasoning and systems based on quantitative spatial computations. Systems based on qualitative spatial reasoning, (e.g. those based on the Region Connection Calculus [QUAL], [LOGIC]) do not usually model explicit geometries, so queries in such systems will likely test for binary spatial relationships between features rather than between explicit geometries. To allow queries for spatial relations between features in quantitative systems, GeoSPARQL defines a series of query transformation rules that expand a feature-only query into a geometry-based query. With these transformation rules, queries about spatial relations between features will have the same specification in both qualitative systems and quantitative systems. The qualitative system will likely evaluate the query with a backward-chaining spatial “reasoner”, and the quantitative system can transform the query into a geometry-based query that can be evaluated with computational geometry.

[1] https://en.wikipedia.org/wiki/SPARQL

[2] https://www.w3.org/TR/sparql11-query/#extensionFunctions

OGC GeoSPARQL – A Geographic Query Language for RDF Data

1. Scope

This is the specification document for GeoSPARQL which, as a whole, comprises multiple parts. See the Introduction section GeoSPARQL Standard structure for details of the parts.

GeoSPARQL does not define a comprehensive vocabulary for representing spatial information. It instead defines a core set of classes, properties and datatypes that can be used to construct query patterns. Many useful extensions to this vocabulary are possible, and we intend for the Semantic Web and Geographic Information System (GIS) communities to develop additional vocabulary for describing spatial information.

2. Conformance

Conformance with this specification shall be checked using all the relevant tests specified in *Annex A — Abstract Test Suite*. The framework, concepts, and methodology for testing, and the criteria to be achieved to claim conformance are specified in *ISO 19105: Geographic information — Conformance and Testing* [ISO19105].

This document establishes several requirements classes and corresponding conformance classes (a conformance class is a set of tests for each requirement in a requirements class). Any GeoSPARQL implementation claiming conformance with one of the conformance classes shall pass all the tests in the associated abstract test suite.

Requirements and conformance tests have IRIs that are relative to versioned namespace IRIs. Requirements and conformance test that are defined in GeoSPARQL 1.0 have IRIs relative to `http://www.opengis.net/spec/geosparql/1.0/`, requirements and conformance test that are added in GeoSPARQL 1.1 have IRIs relative to `http://www.opengis.net/spec/geosparql/1.1/`.

Many conformance classes are parameterized. For parameterized conformance classes, the list of parameters is given within parenthesis.

Table 1. Conformance Classes

Conformance class	**Description**	**Subclause of the abstract test suite**
Core	Defines top-level spatial vocabulary components	A.1
Topology Vocabulary Extension (relation_family)	Defines topological relation vocabulary	A.2
Geometry Extension (serialization, version)	Defines geometry vocabulary and non- topological query functions	A.3
Geometry Topology Extension (serialization, version, relation_family)	Defines topological query functions for geometry objects	A.4
RDFS Entailment Extension (serialization, version , relation_family)	Defines a mechanism for matching implicit RDF triples that are derived based on RDF and RDFS semantics	A.5
Query Rewrite Extension (serialization, version, relation_family)	Defines query transformation rules for computing spatial relations between spatial objects based on their associated geometries	A.6

Dependencies between each GeoSPARQL requirements class are shown below in Figure 2. To support a requirements class for a given set of parameter values, an implementation must support each dependent requirements class with the same set of parameter values.

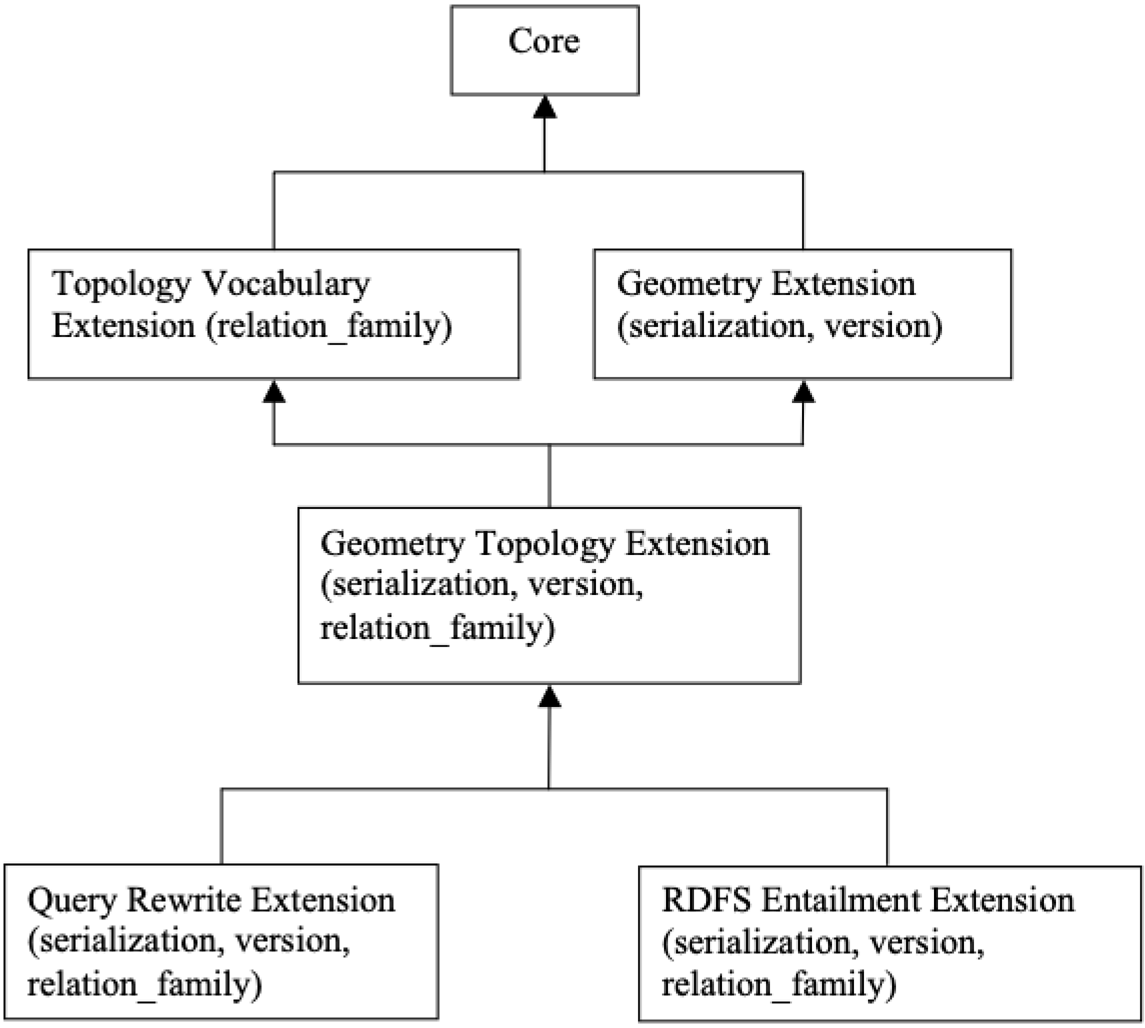

Figure 2. Requirements Class Dependency Graph

3. Normative references

The following normative documents contain provisions which, 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. However, parties to agreements based on this document are encouraged to investigate the possibility of applying the most recent editions of the normative documents indicated below. For undated references, the latest edition of the normative document referred to applies.

Items in this list are liked to their full citation Bibliography.

- [ISO19125-1], *ISO 19125-1: Geographic information — Simple feature access — Part 1: Common architecture*
- [ISO19156], *ISO 19156: Geographic information — Observations and measurements*
- [OGC07-036], *OGC 07-036: Geography Markup Language (GML) Encoding Standard*
- [IETF3987], Internet Engineering Task Force, *RFC 3987: Internationalized Resource Identifiers (IRIs)*
- [OWL2] *OWL 2 Web Ontology Language Document Overview (Second Edition)*
- [RDF], *RDF 1.1 Concepts and Abstract Syntax*
- [RDFS] *RDF Schema 1.1*
- [RIFCORE], *RIF Core Dialect (Second Edition)*
- [SPARQL], *SPARQL 1.1 Query Language*
- [SPARQLENT], *SPARQL 1.1 Entailment Regimes*
- [SPARQLPROT], *SPARQL 1.1 Protocol*
- [SPARQLRESX], *SPARQL Query Results XML Format (Second Edition)*
- [SPARQLRESJ], *SPARQL 1.1 Query Results JSON Format*

4. Terms and definitions

For the purposes of this document, the terms and definitions given in the above normative references apply, as well as those reproduced or created in this section.

4.1. Semantic Web

The following terms and their definitions relate to Semantic Web models, tools and methods.

4.1.1. RDF

The Resource Description Framework (RDF) is a framework for representing information in the Web. RDF graphs are sets of subject-predicate-object triples, where the elements may be IRIs, blank nodes, or datatyped literals. They are used to express descriptions of resources. [RDF]

4.1.2. RDFS

RDF Schema provides a data-modelling vocabulary for RDF data. RDF Schema is an extension of the basic RDF vocabulary. [RDFS]

4.1.3. OWL

The OWL 2 Web Ontology Language, informally OWL 2, is an ontology language for the Semantic Web with formally defined meaning. OWL 2 ontologies provide classes, properties, individuals, and data values and are stored as Semantic Web documents. OWL 2 ontologies can be used along with information written in RDF, and OWL 2 ontologies themselves are primarily exchanged as RDF documents. [OWL2]

4.1.4. SPARQL

SPARQL is a query language for RDF. The results of SPARQL queries can be result sets or RDF graphs. [SPARQL]

4.2. Spatial

The following terms and their definitions relate to spatial science and data.

4.2.1. coordinate system

A coordinate system is a set of mathematical rules for specifying how coordinates are to be assigned to points.

4.2.2. coordinate reference system

A coordinate reference system (CRS) is a coordinate system that is related to an object by a datum.

4.2.3. datum

A datum is a parameter or set of parameters that define the position of the origin, the scale, and the orientation of a coordinate system.

4.2.4. discrete global grid system

A discrete global grid system (DGGS) is a spatial reference system that represents the Earth, or any other globe-like object, with a tessellation of nested cells. Generally, a DGGS will exhaustively partition the globe in closely packed hierarchical tessellations, each cell representing a homogenous value, with a unique identifier or indexing that allows for linear ordering, parent-child operations, and nearest neighbour algebraic operations.

4.2.5. spatial reference system

A spatial reference system (SRS) is a system for establishing spatial position. A spatial reference system can use geographic identifiers (place names, for example), coordinates (in which case it is a coordinate reference system), or identifiers with structured geometry (in which case it is a discrete global grid system).

5. Conventions

5.1. Symbols and abbreviated terms

In this specification, the following common acronyms are used:

CRS	Coordinate Reference System
DGGS	Discrete Global Grid System
GeoJSON	Geographic JavaScript Object Notation
GFM	General Feature Model (as defined in ISO 19109)
GML	Geography Markup Language
KML	Keyhole Markup Language
OWL	OWL 2 Web Ontology Language
RCC	Region Connection Calculus
RDF	Resource Description Framework
RDFS	RDF Schema
RIF	Rule Interchange Format
SPARQL	SPARQL Protocol and RDF Query Language
SRS	Spatial Reference System
WKT	Well Known Text (as defined by Simple Features or ISO 19125)
W3C	World Wide Web Consortium (http://www.w3.org/)
XML	Extensible Markup Language

5.2. Namespaces

The following IRI namespace prefixes are used throughout this document:

ex:	http://example.com/
geo:	http://www.opengis.net/ont/geosparql#
geof:	http://www.opengis.net/def/function/geosparql/
geor:	http://www.opengis.net/def/rule/geosparql/
gml:	http://www.opengis.net/ont/gml#
my:	http://example.org/ApplicationSchema#
ogc:	http://www.opengis.net/
owl:	http://www.w3.org/2002/07/owl#
rdf:	http://www.w3.org/1999/02/22-rdf-syntax-ns#
rdfs:	http://www.w3.org/2000/01/rdf-schema#

sf:	http://www.opengis.net/ont/sf#
skos:	http://www.w3.org/2004/02/skos/core#
xsd:	http://www.w3.org/2001/XMLSchema#

5.3. Placeholder IRIs

All of these namespace prefixes in the previous section resolve to resources that contain their namespace content except for eg: (http://example.com/), which is used just for examples, and ogc: (http://www.opengis.net/), which is used in requirement specifications as a placeholder for the geometry literal serialization used in a fully-qualified conformance class, e.g. <http://www.opengis.net/ont/geosparql#wktLiteral>.

5.4. RDF Serializations

Three RDF serializations are used in this document. Terse RDF Triple Language (turtle) [TURTLE] is used for RDF snippets placed within the main body of the document, and turtle, JSON-LD [JSON-LD] & RDF/XML [RDFXML] is used for the examples in *Annex B — GeoSPARQL Examples*.

6. Core

This clause establishes the **core** requirements class, with IRI `/req/core`, which has a single corresponding conformance class, **core**, with IRI `/conf/core`. This requirements class defines a set of classes and properties for representing geospatial data. The resulting vocabulary - an ontology - can be used to construct SPARQL graph patterns for querying appropriately modeled geospatial data. RDFS and OWL vocabulary have both been used so that the vocabulary can be understood by systems that support only RDFS entailment and by systems that support OWL-based reasoning.

The figure below overviews the classes and properties defined by GeoSPARQL in the **core**, **Topology Vocabulary Extension** and **Geometry Extension**, **Geometry Topology Extension** and **RDFS Entailment Extension** requirements classes.

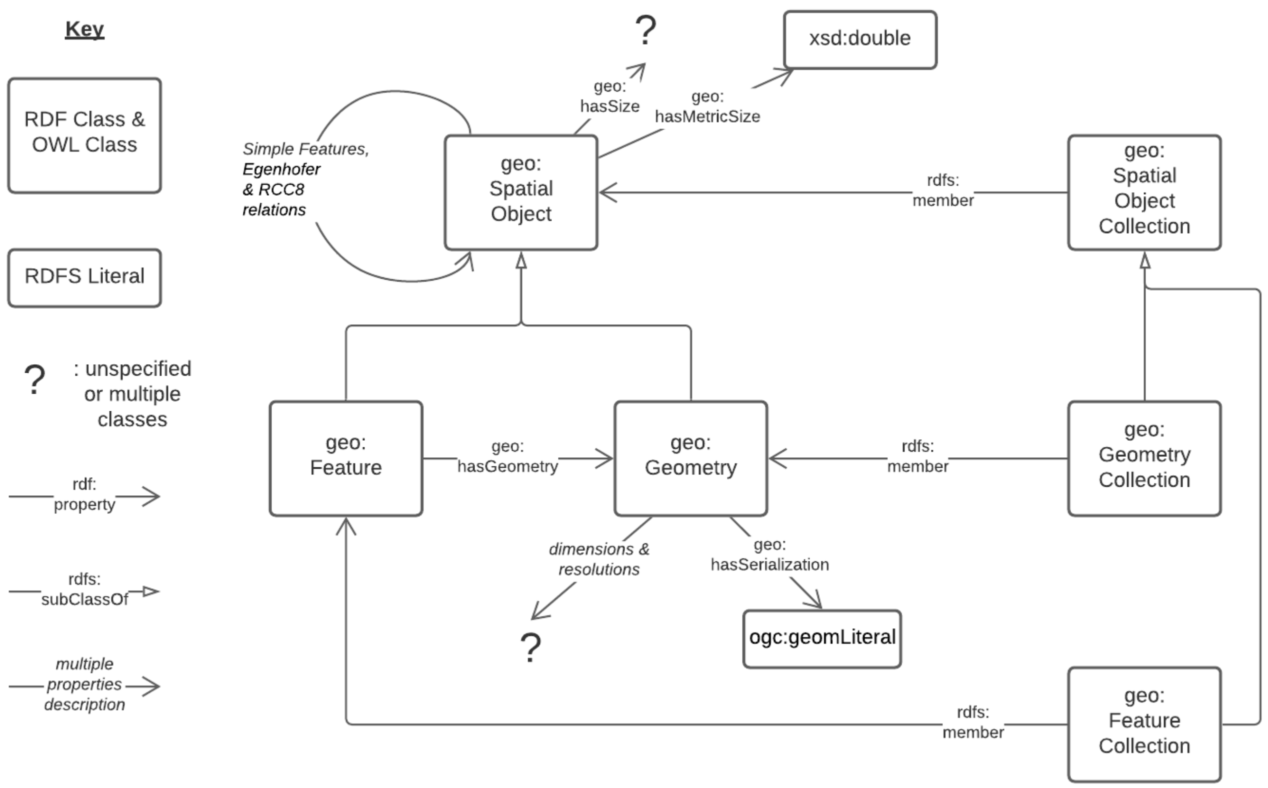

Figure 3. An overview of the Classes and Properties defined in GeoSPARQL. Where specific Classes and Properties are indicated, their prefixed forms of their ontology identifiers (IRIs) are given. Where types or collections of properties are given, they are described in italics. Where unspecified Classes are given, they are represented with a question mark. For cardinalities and other ontology restrictions, see the ontology document. Subproperties of `geo:hasSize`, its metric equivalent and `geo:hasSerialization` are not shown for clarity.

6.1. SPARQL

Req 1 Implementations shall support the SPARQL Query Language for RDF [SPARQL], the SPARQL Protocol [SPARQLPROT] and the SPARQL Query Results XML [SPARQLRESX] and JSON [SPARQLRESJ] Formats.
`http://www.opengis.net/spec/geosparql/1.0/req/core/sparql-protocol`

6.2. Classes

Two main classes are defined: `geo:SpatialObject` and `geo:Feature`. The class `geo:Feature` is equivalent to the UML class `Feature` defined in [ISO19109].

Two container classes are defined: Spatial Object Collection and Feature Collection.

6.2.1. Class: geo:SpatialObject

The class `geo:SpatialObject` is defined by the following:

```
geo:SpatialObject
    a rdfs:Class, owl:Class ;
    rdfs:isDefinedBy geo: ;
    skos:prefLabel "Spatial Object"@en ;
    skos:definition "The class Spatial Object represents everything that can
                    have a spatial representation. It is superclass of feature
                    and geometry"@en .
```

Req 2 Implementations shall allow the RDFS class `geo:SpatialObject` to be used in SPARQL graph patterns.	
`http://www.opengis.net/spec/geosparql/1.0/req/core/spatial-object-class`	

Example:

```
eg:x
    a geo:SpatialObject ;
     skos:prefLabel "Object X";
.
```

6.2.2. Class: geo:Feature

The class `geo:Feature` is equivalent to the class `GFI_Feature` [ISO19156] and is defined by the following:

```
geo:Feature
    a rdfs:Class, owl:Class ;
    rdfs:isDefinedBy geo: ;
    skos:prefLabel "Feature"@en ;
    rdfs:subClassOf geo:SpatialObject ;
    owl:disjointWith geo:Geometry ;
    skos:definition "This class represents the top-level feature type. This
                    class is equivalent to GFI_Feature defined in ISO 19156,
                    and it is superclass of all feature types."@en .
```

Req 3 Implementations shall allow the RDFS class `geo:Feature` to be used in SPARQL graph patterns.
`http://www.opengis.net/spec/geosparql/1.0/req/core/feature-class`

6.2.3. Class: geo:SpatialObjectCollection

The class `geo:SpatialObjectCollection` is defined by the following:

```
geo:SpatialObjectCollection
  a owl:Class ;
  rdfs:isDefinedBy geo: ;
  skos:prefLabel "Spatial Object Collection" ;
  skos:definition "The class Spatial Object Collection represents any collection of
                  individual Spatial Objects. It is superclass of Feature Collection
                  and Geometry Collection."@en ;
  rdfs:subClassOf rdfs:Container ;
  rdfs:subClassOf [
      a owl:Restriction ;
      owl:allValuesFrom geo:SpatialObject ;
      owl:onProperty rdfs:member ;
    ] ;
.
```

The restriction imposed on the generic `rdfs:Container` that defines this class is that only instances of Spatial Object are allowed to be members of it and these are indicated with the `rdfs:member` property.

Req 5 Implementations shall allow the RDFS class `geo:SpatialObjectCollection` to be used in SPARQL graph patterns.
`http://www.opengis.net/spec/geosparql/1.1/req/core/spatial-object-collection-class`

6.2.4. Class: geo:FeatureCollection

The class `geo:FeatureCollection` is defined by the following:

```
geo:FeatureCollection
  a owl:Class ;
  rdfs:isDefinedBy geo: ;
  skos:prefLabel "Feature Collection" ;
  skos:definition "The class Feature Collection represents
                  any collection of individual Features."@en ;
  rdfs:subClassOf geo:SpatialObjectCollection ;
  rdfs:subClassOf [
      a owl:Restriction ;
      owl:allValuesFrom :Feature ;
      owl:onProperty rdfs:member ;
    ] ;
.
```

The restriction imposed on the more general Spatial Object Collection that defines this class is that only instances of Feature are allowed to be members of it and these are to be indicated with the `rdfs:member` property.

Req 6 Implementations shall allow the RDFS class `geo:FeatureCollection` to be used in SPARQL graph patterns.	
`http://www.opengis.net/spec/geosparql/1.1/req/core/feature-collection-class`	

6.3. Standard Properties for geo:SpatialObject

Properties are defined for associating Spatial Objects with scalar spatial measurements (sizes) .

Req 7 Implementations shall allow the properties `geo:hasSize`, `geo:hasMetricSize`, `geo:hasLength`, `geo:hasMetricLength`, `geo:hasPerimeterLength`, `geo:hasMetricPerimeterLength`, `geo:hasArea`, `geo:hasMetricArea`, `geo:hasVolume` and `geo:hasMetricVolume`. to be used in SPARQL graph patterns.	
`http://www.opengis.net/spec/geosparql/1.1/req/core/spatial-object-properties`	

6.3.1. Property: geo:hasSize

The property `geo:hasSize` is the superproperty of all properties that can be used to indicate the size of a Spatial Object in case (only) metric units (meter, square meter or cubic meter) can not be used. If it is possible to express size in metric units, subproperties of `geo:hasMetricSize` should be used. This property has not range specification. This makes it possible to use other vocabularies for expressions of size, for example vocabularies for units of measurment or vocabularies for specifying measurement quality.

GeoSPARQL 1.1 defines the following subproperties of this property: `geo:hasLength`, `geo:hasPerimterLength`, `geo:hasArea` and `geo:hasVolume`.

```
geo:hasSize
    a rdf:Property, owl:ObjectProperty ;
    rdfs:isDefinedBy geo: ;
    rdfs:domain geo:SpatialObject ;
    skos:definition "Subproperties of this property are used to indicate the size of a
                    Spatial Object as a measurement or estimate of one or more
dimensions
                    of the Spatial Object's spatial presence."@en ;
    skos:prefLabel "has size"@en ;
.
```

6.3.2. Property: geo:hasMetricSize

The property `geo:hasMetricSize` is the superproperty of all properties that can be used to indicate the size of a Spatial Object using metric units (meter, square meter or cubic meter). Using a subproperty of this property is the recommended way to specify size, because using a standard unit of length (meter) benefits data interoperability and simplicity. Subproperties of `geo:hasSize` can be used if more complex expressions are necessary, for example if the unit of length can not be converted to meter, or if additional data are needed to describe the measurement or estimate of size.

GeoSPARQL 1.1 defines the following subproperties of this property: `geo:hasMetricLength`, `geo:hasMetricPerimeterLength`, `geo:hasMetricArea` and `geo:hasMetricVolume`.

```
geo:hasMetricSize
    a rdf:Property, owl:DatatypeProperty ;
    rdfs:isDefinedBy geo: ;
    rdfs:domain geo:SpatialObject ;
    rdfs:range xsd:double ;
    skos:definition "Subproperties of this property are used to indicate the size of a
                    Spatial Object, as a measurement or estimate of one or more
dimensions
                    of the Spatial Object's spatial presence. Units are always metric
                    (meter, square meter or cubic meter)."@en ;
    skos:prefLabel "has metric size"@en ;
.
```

6.3.3. Property: geo:hasLength

The property `geo:hasLength` can be used to indicate the length of a Spatial Object if it is not possible to use the property `geo:hasMetricLength`. It is a subproperty of `geo:hasSize`.

```
geo:hasLength
    a rdf:Property, owl:ObjectProperty ;
    rdfs:isDefinedBy geo: ;
    rdfs:subPropertyOf geo:hasSize ;
    rdfs:domain geo:SpatialObject ;
    skos:definition "The length of a Spatial Object."@en ;
    skos:prefLabel "has length"@en ;
.
```

6.3.4. Property: geo:hasMetricLength

The property `geo:hasMetricLength` can be used to indicate the length of a Spatial Object in meters (m). It is a subproperty of `geo:hasMetricSize`. This property can be used for Spatial Objects having one, two, or three dimensions.

```
geo:hasMetricLength
    a rdf:Property, owl:DatatypeProperty ;
    rdfs:isDefinedBy geo: ;
    rdfs:subPropertyOf geo:hasMetricSize ;
    rdfs:domain geo:SpatialObject ;
    rdfs:range xsd:double ;
    skos:definition "The length of a Spatial Object in meters."@en ;
    skos:prefLabel "has length in meters"@en ;
.
```

6.3.5. Property: geo:hasPerimeterLength

The property `geo:hasPerimeterLength` can be used to indicate the length of the outer boundary of a Spatial Object if it is not possible to use the property `geo:hasMetricPerimeterLength`. It is a subproperty of `geo:hasSize`.

```
geo:hasPerimeterLength
    a rdf:Property, owl:ObjectProperty ;
    rdfs:isDefinedBy geo: ;
    rdfs:subPropertyOf geo:hasSize ;
    skos:definition "The length of the perimeter of a Spatial Object."@en ;
    skos:prefLabel "has perimeter length"@en ;
.
```

6.3.6. Property: geo:hasMetricPerimeterLength

The property `geo:hasMetricPerimeterLength` can be used to indicate the length of the outer boundary of a Spatial Object in meters (m). It is a subproperty of `geo:hasMetricSize`. Circumference is considered a type of perimeter, so this property can be used for circular or curved objects too. This property can be used for Spatial Objects having two or three dimensions.

```
geo:hasMetricPerimeterLength
    a rdf:Property, owl:DatatypeProperty ;
    rdfs:isDefinedBy geo: ;
    rdfs:subPropertyOf geo:hasMetricSize ;
    rdfs:domain geo:SpatialObject ;
    rdfs:range xsd:double ;
    skos:definition "The length of the perimeter of a Spatial Object in meters."@en ;
    skos:prefLabel "has perimeter length in meters"@en ;
.
```

TIP A consistency check can be applied to Geometry instances indicating both this property and the property `geo:dimension`: if supplied, the `geo:dimension` property's range value must be the literal integer 2 or 3. The following SPARQL query will return `true` if applied to a graph where this is not the case for all Geometries:

```
    PREFIX geo: <http://www.opengis.net/ont/geosparql#>
    ASK
    WHERE {
        ?g geo:hasMetricPerimeterLength ?p ;
           geo:dimension ?d .

        FILTER (?d < 2)
    }
```

6.3.7. Property: geo:hasArea

The property `geo:hasArea` can be used to indicate the area of a Spatial Object if it is not possible to use the property `geo:hasMetricArea`. It is a subproperty of `geo:hasSize`.

```
geo:hasArea
    a rdf:Property, owl:ObjectProperty ;
    rdfs:isDefinedBy geo: ;
    rdfs:subPropertyOf geo:hasSize ;
    rdfs:domain geo:SpatialObject ;
    skos:definition "The area of a Spatial Object."@en ;
    skos:prefLabel "has area"@en ;
.
```

6.3.8. Property: geo:hasMetricArea

The property `geo:hasMetricArea` can be used to indicate the area of a Spatial Object in square meters (m^2). It is a subproperty of `geo:hasMetricSize`. This property can be used for Spatial Objects having two or three dimensions.

```
geo:hasMetricArea
    a rdf:Property, owl:DatatypeProperty ;
    rdfs:isDefinedBy geo: ;
    rdfs:subPropertyOf geo:hasMetricSize ;
    rdfs:domain geo:SpatialObject ;
    rdfs:range xsd:double ;
    skos:definition "The area of a Spatial Object in square meters."@en ;
    skos:prefLabel "has area in meters"@en ;
.
```

TIP A consistency check can be applied to Geometry instances indicating both this property and the property `geo:dimension`: if supplied, the `geo:dimension` property's range value must be the literal integer 2 or 3. The following SPARQL query will return `true` if applied to a graph where this is not the case for all Geometries:

```
    PREFIX geo: <http://www.opengis.net/ont/geosparql#>

    ASK
    WHERE {
        ?g geo:hasMetricArea ?a ;
           geo:dimension ?d .

        FILTER (?d < 2)
    }
```

6.3.9. Property: geo:hasVolume

The property `geo:hasVolume` can be used to indicate the volume of a Spatial Object if it is not possible to use the property `geo:hasMetricVolume`. It is a subproperty of `geo:hasSize`.

```
geo:hasVolume
    a rdf:Property, owl:ObjectProperty ;
    rdfs:isDefinedBy geo: ;
    rdfs:subPropertyOf geo:hasSize ;
    rdfs:domain geo:SpatialObject ;
    skos:definition "The volume of a three-dimensional Spatial Object."@en ;
    skos:prefLabel "has volume"@en ;
.
```

6.3.10. Property: geo:hasMetricVolume

The property `geo:hasMetricVolume` can be used to indicate the volume of a Spatial Object in cubic meters (m^3). It is a subproperty of `geo:hasMetricSize`. This property can be used for Spatial Objects having three dimensions.

```
geo:hasMetricVolume
    a rdf:Property, owl:DatatypeProperty ;
    rdfs:isDefinedBy geo: ;
    rdfs:subPropertyOf :hasMetricSize ;
    rdfs:domain geo:SpatialObject ;
    rdfs:range xsd:double ;
    skos:definition "The volume of a Spatial Object in cubic meters."@en ;
    skos:prefLabel "has area in meters"@en ;
.
```

TIP	A consistency check can be applied to Geometries indicating both this property and the property `geo:dimension`: if supplied, the property `geo:dimension` property's range value must be the literal integer 3. The following SPARQL query will return `true` if applied to a graph where this is not the case for all Geometries:

```
    PREFIX geo: <http://www.opengis.net/ont/geosparql#>

    ASK
    WHERE {
        ?g geo:hasMetricVolume ?v ;
           geo:dimension ?d .

        FILTER (?d != 3)
    }
```

6.4. Standard Properties for geo:Feature

Properties are defined for associating `geo:Feature` instances with `geo:Geometry` instances.

Req 8 Implementations shall allow the properties `geo:hasGeometry`, `geo:hasDefaultGeometry`, `geo:hasCentroid` and `geo:hasBoundingBox` to be used in SPARQL graph patterns.
http://www.opengis.net/spec/geosparql/1.1/req/geometry-extension/feature-properties

6.4.1. Property: geo:hasGeometry

The property `geo:hasGeometry` is used to link a Feature with a Geometry that represents its spatial extent. A given Feature may have many associated geometries.

```
geo:hasGeometry
    a rdf:Property, owl:ObjectProperty ;
    rdfs:isDefinedBy geo: ;
    rdfs:domain geo:Feature ;
    rdfs:range geo:Geometry ;
    skos:prefLabel "has Geometry"@en ;
    skos:definition "A spatial representation for a given Feature."@en ;
.
```

6.4.2. Property: geo:hasDefaultGeometry

The property `geo:hasDefaultGeometry` is used to link a Feature with its default Geometry. The default geometry is the Geometry that should be used for spatial calculations in the absence of a request for a specific geometry (e.g. in the case of query rewrite).

```
geo:hasDefaultGeometry
    a rdf:Property, owl:ObjectProperty ;
    rdfs:isDefinedBy geo: ;
    rdfs:domain geo:Feature ;
    rdfs:range geo:Geometry ;
    skos:prefLabel "has Default Geometry"@en ;
    skos:definition "The default geometry to be used in spatial calculations,
                    usually the most detailed geometry."@en ;
    rdfs:subPropertyOf geo:hasGeometry ;
.
```

GeoSPARQL does not restrict the cardinality of the has default geometry property. It is thus possible for a Feature to have more than one distinct default geometry or to have no default geometry. This situation does not result in a query processing error; SPARQL graph pattern matching simply proceeds as normal. Certain queries may, however, give logically inconsistent results. For example, if a Feature `my:f1` has two asserted default geometries, and those two geometries are disjoint polygons, the query below could return a non-zero count on a system supporting the GeoSPARQL Query Rewrite Extension (rule `geor:sfDisjoint`).

```
PREFIX geo: <http://www.opengis.net/ont/geosparql#>

SELECT (COUNT(*) AS ?cnt)
WHERE { :f1 geo:sfDisjoint :f1 }
```

Such cases are application-specific data modeling errors and are therefore outside of the scope of the GeoSPARQL specification., however it is recommended that multiple geometries indicated with `geo:hasDefaultGeometry` should be differentiated by `Geometry` class properties, perhaps relating to precision, SRS etc.

6.4.3. Property: geo:hasBoundingBox

The property `geo:hasBoundingBox` is used to link a Feature with a simplified geometry-representation corresponding to the envelope of its geometry. Bounding-boxes are typically uses in indexing and discovery.

```
geo:hasBoundingBox
    a rdf:Property, owl:ObjectProperty ;
    rdfs:isDefinedBy geo: ;
    rdfs:subPropertyOf geo:hasGeometry ;
    rdfs:domain geo:Feature ;
    rdfs:range geo:Geometry ;
    skos:prefLabel "has bounding box"@en ;
    skos:definition "The minimum or smallest bounding or enclosing box of a given
Feature."@en ;
    skos:scopeNote "The target is a geometry that defines a rectilinear region whose
edges are
                    aligned with the axes of the coordinate reference system, which
exactly
                    contains the geometry or Feature e.g. sf:Envelope"@en ;
.
```

GeoSPARQL does not restrict the cardinality of the `geo:hasBoundingBox` property. A Feature may be associated with more than one bounding-box, for example in different coordinate reference systems.

6.4.4. Property: geo:hasCentroid

The property `geo:hasCentroid` is used to link a Feature with a point geometry corresponding with the centroid of its geometry. The centroid is typically used to show location on a low-resolution image, and for some indexing and discovery functions.

```
geo:hasCentroid
    a rdf:Property, owl:ObjectProperty ;
    rdfs:isDefinedBy geo: ;
    rdfs:subPropertyOf geo:hasGeometry ;
    rdfs:domain geo:Feature ;
    rdfs:range geo:Geometry ;
    skos:prefLabel "has centroid"@en ;
    skos:definition "The arithmetic mean position of all the geometry points
                    of a given Feature."@en ;
    skos:scopeNote "The target geometry shall describe a point, e.g. sf:Point"@en ;
.
```

GeoSPARQL does not restrict the cardinality of the `geo:hasCentroid` property. A Feature may be associated with more than one centroid, for example computed using different rules or in different coordinate reference systems.

7. Topology Vocabulary Extension (relation_family)

This clause establishes the *Topology Vocabulary Extension (relation_family)* parameterized requirements class, with IRI `/req/topology-vocab-extension`, which has a single corresponding conformance class *Topology Vocabulary Extension (relation_family)*, with IRI `/conf/topology-vocab-extension`. This requirements class defines a vocabulary for asserting and querying topological relations between spatial objects. The class is parameterized so that different families of topological relations may be used, e.g. RCC8, Egenhofer. These relations are generalized so that they may connect features as well as geometries.

A Dimensionally Extended 9-Intersection Model (DE-9IM) pattern, which specifies the spatial dimension of the intersections of the interiors, boundaries and exteriors of two geometric objects, is used to describe each spatial relation. Possible pattern values are `-1 (empty)`, `0, 1, 2, T (true) = {0, 1, 2}`, `F (false) = {-1}`, `* (don't care) = {-1, 0, 1, 2}`. In the following descriptions, the notation `X/Y` is used denote applying a spatial relation to geometry types `X` and `Y` (i.e., `x` *relation* `y` where `x` is of type `X` and `y` is of type `Y`). The symbol `P` is used for 0- dimensional geometries (e.g. points). The symbol `L` is used for 1-dimensional geometries (e.g. lines), and the symbol `A` is used for 2-dimensional geometries (e.g. polygons). Consult the Simple Features specification [ISO19125-1] for a more detailed description of DE-9IM intersection patterns.

7.1. Parameters

The following parameter is defined for the *Topology Vocabulary Extension* requirements class.

relation_family: Specifies the set of topological spatial relations to support.

7.2. Simple Features Relation Family (relation_family=Simple Features)

This clause defines requirements for the *Simple Features* relation family.

Req 9 Implementations shall allow the properties `geo:sfEquals`, `geo:sfDisjoint`, `geo:sfIntersects`, `geo:sfTouches`, `geo:sfCrosses`, `geo:sfWithin`, `geo:sfContains` and `geo:sfOverlaps` to be used in SPARQL graph patterns.
`http://www.opengis.net/spec/geosparql/1.0/req/topology-vocab-extension/sf-spatial-relations`

Topological relations in the *Simple Features* family are summarized in Table 2. Multi-row intersection patterns should be interpreted as a logical `OR` of each row.

Table 2. Simple Features Topological Relations

Relation Name	Relation IRI	Domain/Range	Applies To Geometry Types	DE-9IM Intersection Pattern
equals	`geo:sfEquals`	`geo:SpatialObject`	`All`	`(TFFFTFFFT)`

Relation Name	Relation IRI	Domain/Range	Applies To Geometry Types	DE-9IM Intersection Pattern
disjoint	`geo:sfDisjoint`	`geo:SpatialObject`	`All`	`(FF**FF****)`
intersects	`geo:sfIntersects`	`geo:SpatialObject`	`All`	`(T******** *T******* ***T***** ****T****)`
touches	`geo:sfTouches`	`geo:SpatialObject`	`All except P/P`	`(FT******* F**T***** F***T****)`
within	`geo:sfWithin`	`geo:SpatialObject`	`All`	`(T*F**F***)`
contains	`geo:sfContains`	`geo:SpatialObject`	`All`	`(T*****FF*)`
overlaps	`geo:sfOverlaps`	`geo:SpatialObject`	`A/A, P/P, L/L`	`(T*T***T**) for A/A, P/P`; `(1*T***T**) for L/L`
crosses	`geo:sfCrosses`	`geo:SpatialObject`	`P/L, P/A, L/A, L/L`	`(T*T***T**) for P/L, P/A, L/A`; `(0********) for L/L`

7.3. Egenhofer Relation Family (relation_family=Egenhofer)

This clause defines requirements for the 9-intersection model for binary topological relations (*Egenhofer*) relation family. Consult references [FORMAL] and [CATEG] for a more detailed discussion of *Egenhofer* relations.

Req 10 Implementations shall allow the properties `geo:ehEquals`, `geo:ehDisjoint`, `geo:ehMeet`, `geo:ehOverlap`, `geo:ehCovers`, `geo:ehCoveredBy`, `geo:ehInside` and `geo:ehContains` to be used in SPARQL graph patterns.
`http://www.opengis.net/spec/geosparql/1.0/req/topology-vocab-extension/eh-spatial-relations`

Topological relations in the *Egenhofer* family are summarized in Table 3. Multi-row intersection patterns should be interpreted as a logical `OR` of each row.

Table 3. Egenhofer Topological Relations

Relation Name	Relation IRI	Domain/Range	Applies To Geometry Types	DE-9IM Intersection Pattern
equals	`geo:ehEquals`	`geo:SpatialObject`	`All`	`(TFFFTFFFT)`
disjoint	`geo:ehDisjoint`	`geo:SpatialObject`	`All`	`(FF*FF****)`

Relation Name	Relation IRI	Domain/Range	Applies To Geometry Types	DE-9IM Intersection Pattern
meet	`geo:ehMeet`	`geo:SpatialObject`	`All except P/P`	`(FT******* F**T***** F***T****)`
overlap	`geo:ehOverlap`	`geo:SpatialObject`	`All`	`(T*T***T**)`
covers	`geo:ehCovers`	`geo:SpatialObject`	`A/A, A/L, L/L`	`(T*TFT*FF*)`
covered by	`geo:ehCoveredBy`	`geo:SpatialObject`	`A/A, L/A, L/L`	`(TFF*TFT**)`
inside	`geo:ehInside`	`geo:SpatialObject`	All	`(TFF*FFT**)`
contains	`geo:ehContains`	`geo:SpatialObject`	All	`(T*TFF*FF*)`

7.4. RCC8 Relation Family (relation_family=RCC8)

This clause defines requirements for the region connection calculus basic 8 (*RCC8*) relation family. Consult references [QUAL] and [LOGIC] for a more detailed discussion of *RCC8* relations.

Req 11 Implementations shall allow the properties `geo:rcc8eq`, `geo:rcc8dc`, `geo:rcc8ec`, `geo:rcc8po`, `geo:rcc8tppi`, `geo:rcc8tpp`, `geo:rcc8ntpp`, `geo:rcc8ntppi` to be used in SPARQL graph patterns.
`http://www.opengis.net/spec/geosparql/1.0/req/topology-vocab-extension/rcc8-spatial-relations`

Topological relations in the *RCC8* family are summarized in Table 4.

Table 4. RCC8 Topological Relations

Relation Name	Relation IRI	Domain/Range	Applies To Geometry Types	DE-9IM Intersection Pattern
equals	`geo:rcc8eq`	`geo:SpatialObject`	`A/A`	`(TFFFTFFFT)`
disconnected	`geo:rcc8dc`	`geo:SpatialObject`	`A/A`	`(FFTFFTTTT)`
externally connected	`geo:rcc8ec`	`geo:SpatialObject`	`A/A`	`(FFTFTTTTT)`
partially overlapping	`geo:rcc8po`	`geo:SpatialObject`	`A/A`	`(TTTTTTTTT)`
tangential proper part inverse	`geo:rcc8tppi`	`geo:SpatialObject`	`A/A`	`(TTTFTTFFT)`
tangential proper part	`geo:rcc8tpp`	`geo:SpatialObject`	`A/A`	`(TFFTTFTTT)`
non-tangential proper part	`geo:rcc8ntpp`	`geo:SpatialObject`	`A/A`	`(TFFTFFTTT)`
non-tangential proper part inverse	`geo:rcc8ntppi`	`geo:SpatialObject`	`A/A`	`(TTTFFTFFT)`

7.5. Equivalent RCC8, Egenhofer and Simple Features Topological Relations

Table 5 summarizes the equivalences between *Egenhofer*, *RCC8* and *Simple Features* spatial relations for closed, non-empty regions. The symbol + denotes logical `OR`, and the symbol ¬ denotes negation.

Table 5. Equivalent Simple Features, RCC8 and Egenhofer relations

Simple Features	**RCC8**	**Egenhofer**
equals	equals	equals
disjoint	disconnected	disjoint
intersects	¬ disconnected	¬ disjoint
touches	externally connected	meet
within	non-tangential proper part + tangential proper part	inside + coveredBy
contains	non-tangential proper part inverse + tangential proper part inverse	contains + covers
overlaps	partially overlapping	overlap

8. Geometry Extension (serialization, version)

This clause establishes the *Geometry Extension (serialization, version)* parameterized requirements class, with IRI `/req/geometry-extension`, which has a single corresponding conformance class *Geometry Extension (serialization, version)*, with IRI `/conf/geometry-extension`. This requirements class defines a vocabulary for asserting and querying information about geometry data, and it defines query functions for operating on geometry data.

As part of the vocabulary, RDFS datatypes are defined for encoding detailed geometry information as a literal value. A literal representation of a geometry is needed so that geometric values may be treated as a single unit. Such a representation allows geometries to be passed to external functions for computations and to be returned from a query.

8.1. Rationale

Other schemes for encoding simple geometry data in RDF have been implemented. The W3C Basic Geo vocabulary[3] was an early (2003) RDF vocabulary for "representing lat(itude), long(itude) and other information about spatially-located things, using WGS84 as a reference datum" and many widely used Semantic Web vocabularies contain some spatial data support. For example, *Dublin Core Terms* provides a *Location* class[4] for "A spatial region or named place." and *schema.org* provides a number of spatial object and geometry classes, such as `GeoCoordinates` [5] and `GeoShape` [6].

Many vocabularies, such as these two, provide little specific support for detailed geometries and only support the WGS84 Coordinate Reference System (CRS).

Since 2012 and the first version of GeoSPARQL, many ontologies have imported GeoSPARQL, for example, the *ISA Programme Location Core Vocabulary* [7] whose usage notes provide examples containing GeoSPARQL literals and the use of GeoSPARQL's "geometry class". The W3C's more recent *Data Catalog Vocabulary, Version 2* (DCAT2) standard[8] similarly contains usage notes for `geometry`, `bbox` and other properties that suggest the use of GeoSPARQL literals.

Some of the properties defined in these vocabularies, such as DCAT2's `spatialResolution` have motivated the inclusion of new properties in this version of GeoSPARQL. In this case the equivalent property is `geo:hasSpatialResolution`. The GeoSPARQL 1.1 Standards Working Group charter [CHARTER] contains references to a number of vocabularies/ontologies that were influential in the generation of this version of GeoSPARQL.

8.2. GeoSPARQL and Simple Features (SFA-CA)

The GeoSPARQL Geometry Extension is largely based on the the specification Simple Features Access - Common Architecture (SFA-CA) [OGCSFACA]. Contrary to what the name may imply, SFA-CA is about Geometry, not about Features. SFA-CA describes simple geometry, meaning that geometric shapes are based on points and straight lines (lineair interpolations) between points. Within a single Geometry, these lines may not cross.

Neither GeoSPARQL nor SFA-CA support full three dimensional geometry. Coordinates may be three-dimensional, which means that points may have a Z-coordinate next to an X- and Y-coordinate. The Z-coordinate then holds the value of height or depth. However, lines or surfaces can only have one Z value for any explicit or interpolated X,Y pair. This is often referred to as 2.5 dimensional geometry. Geometric functions working with Geometries that have Z values will ignore Z values in calculations and first project geometry onto the Z=0 level.

SFA-CA also describes M coordinate values that may be part of geometry encodings. The M value represents a measure, a value that can be used in information systems that support linear referencing. GeoSPARQL at the moment does not support linear referencing. Like Z values in coordinates, M values are to be ignored.

SFA-CA specifies a class hierarchy for Geometry. Although these classes are not part of the GeoSPARQL ontology, the GeoSPARQL DWG does publish a vocabulary of Simple Features geometry: http://www.opengis.net/ont/sf. Geometry types defined in this vocabulary can be considered safe to use with GeoSPARQL. The two Geometry serializations that were specified in GeoSPARQL 1.0, WKT and GML, fully support all SFA-CA geometry types. However, the two Geometry serializations that were introduced in GeoSPARQL 1.1 do not. Some SFA-CA geometry types are not supported by the KML or GeoJSON format. For example, neither KML nor GeoJSON support the TIN or Triangle geometry types.

8.3. Recommendation for specification of units of measurement

For geometric data to be interpreted and used correctly, the unit of distance should be known. Typically, the particular Coordinate Reference System (CRS) that is associated with a Geometry instance will specify a unit of measurement. However, some elements of GeoSPARQL allow arbitrary units of distance to be used, for example the property `geo:hasSpatialResolution` or the function `geof:buffer`. In those cases it is advisable to make use of a well known web vocabulary for units of measurement. That will improve data interoperability. The recommended vocabulary for units of measurement for GeoSPARQL is the *Quantities, Units, Dimensions and Types (QUDT)* ontology[9].

8.4. Influence of Coordinate Reference Systems on geometric computations

Geometric computations must always be mindful of the kind of space that is described by a Coordinate Reference System (CRS). A Geometry object consists of a set of coordinates and a specification on how the coordinates should be interpreted: the CRS. Taken together, coordinates and CRS allow performing computations on Geometry objects. For example, sizes can be calculated or new Geometry objects can be created. Some Coordinate Reference Systems describe a two-dimensional flat space. In that case, coordinates are understood to be Cartesian, and Cartesian geometric computations can be performed. But Coordinate Reference Systems can describe other types of spaces, to which Cartesian computations are not applicable. For example, if CRS `<http://www.opengis.net/def/crs/OGC/1.3/CRS84>` is used, coordinates are to be interpreted as degrees of latitude and longitude, designating positions on a spheroid. The distance between two points using this CRS is different from the distance between two points that have the same coordinates but are based on a Cartesian CRS.

To avoid erroreneous computations involving Geometry, data publishers are recommended to clearly make the type of space that is described by the CRS known.

8.5. Parameters

The following parameters are defined for the *Geometry Extension* requirements class.

serialization

Specifies the serialization standard to use when generating geometry literals and also the supported geometry types.

NOTE	a serialization strongly affects the geometry conceptualization. The WKT serialization aligns the geometry types with *ISO 19125 Simple Features* [ISO19125-1], and the GML serialization aligns the geometry types with *ISO 19107 Spatial Schema* [ISO19107].

version

Specifies the version of the serialization format used.

8.6. Geometry Class

A single root geometry class is defined: `geo:Geometry`. In addition, properties are defined for describing geometry data and for associating geometries with features.

One container class is defined: Geometry Collection.

8.6.1. Class: geo:Geometry

The class `geo:Geometry` is equivalent to `GM_Object` [ISO19107] and is defined by the following:

```
geo:Geometry
    a rdfs:Class, owl:Class ;
    rdfs:isDefinedBy geo: ;
    skos:prefLabel "Geometry"@en ;
    rdfs:subClassOf geo:SpatialObject ;
    owl:disjointWith geo:Feature;
    skos:definition "The class represents the top-level geometry type. This class
                    is equivalent to the UML class GM_Object defined in ISO 19107,
                    and it is superclass of all geometry types."@en ;
.
```

Req 12 Implementations shall allow the RDFS class `geo:Geometry` to be used in SPARQL graph patterns.
`http://www.opengis.net/spec/geosparql/1.0/req/geometry-extension/geometry-class`

8.6.2. Class: geo:GeometryCollection

The class Geometry Collection is defined by the following:

```
geo:GeometryCollection
  a owl:Class ;
  rdfs:isDefinedBy geo: ;
  skos:prefLabel "Geometry Collection"@en ;
  skos:definition "The class Geometry Collection represents any collection of
individual Geometries."@en ;
  rdfs:subClassOf geo:SpatialObjectCollection ;
  rdfs:subClassOf [
      a owl:Restriction ;
      owl:allValuesFrom geo:Geometry ;
      owl:onProperty rdfs:member ;
    ] ;
.
```

The restriction imposed on the more general Spatial Object Collection that defines this class is that only instances of Geometry are allowed to be members of it and these are to be indicated with the `rdfs:member` property.

NOTE

There is no RDF/ontology relationship between this `geo:GeometryCollection` class and the Simple Features Vocabulary's `sf:GeometryCollection` class since the former is a collection of `geo:Geometry` objects and the latter is to be used for compound geometry literals.

`sf:GeometryCollection` instances can act as input or output of GeoSPARQL functions whereas `geo:GeometryCollection` instances are more likely to be used for grouping `geo:Geometry` objects for other purposes.

Many geometry literal formats also have the ability to represent multiple geometries. GML & KML use a *MultiGeometry* type and WKT & GeoJSON use a *GeometryCollection* type. While the names of some of these objects is the same as this class' and all the concepts are similar, there is also no RDF/ontology relationship between this class and these literals. This class contains whole `geo:Geometry` instances, which may have more information within them than just a geometry serialization.

As per the expected use of `sf:GeometryCollection` instances mentioned above: the uses of multi-geometry literals and `geo:GeometryCollection` instances is expected to be different too.

Req 13 Implementations shall allow the RDFS class `geo:GeometryCollection` to be used in SPARQL graph patterns.
`http://www.opengis.net/spec/geosparql/1.1/req/core/geometry-collection-class`

8.7. Standard Properties for geo:Geometry

Properties are defined for describing geometry metadata.

Req 14 Implementations shall allow the properties `geo:dimension`, `geo:coordinateDimension`, `geo:spatialDimension`, `geo:hasSpatialResolution`, `geo:hasMetricSpatialResolution`, `geo:hasSpatialAccuracy`, `geo:hasMetricSpatialAccuracy`, `geo:isEmpty`, `geo:isSimple` and `geo:hasSerialization` to be used in SPARQL graph patterns.
`http://www.opengis.net/spec/geosparql/1.1/req/geometry-extension/geometry-properties`

8.7.1. Property: geo:dimension

The property `geo:dimension` is used to link the a Geometry object to its topological dimension, which must be less than or equal to the coordinate dimension. In non-homogeneous collections, this will return the largest topological dimension of the contained objects.

```
geo:dimension
    a rdf:Property, owl:DatatypeProperty ;
    rdfs:isDefinedBy geo: ;
    skos:prefLabel "dimension"@en ;
    skos:definition "The topological dimension of this geometric object, which
                    must be less than or equal to the coordinate dimension. In
                    non-homogeneous collections, this is the largest
                    topological dimension of the contained objects."@en ;
    rdfs:domain geo:Geometry ;
    rdfs:range xsd:integer ;
.
```

8.7.2. Property: geo:coordinateDimension

The property `geo:coordinateDimension` is defined to link a Geometry object to the dimension of direct positions (coordinate tuples) used in the Geometry's definition.

```
geo:coordinateDimension
    a rdf:Property, owl:DatatypeProperty;
    rdfs:isDefinedBy geo: ;
    skos:prefLabel "coordinate dimension"@en ;
    skos:definition "The number of measurements or axes needed to describe the
                    position of this Geometry in a coordinate system."@en ;
    rdfs:domain geo:Geometry ;
    rdfs:range xsd:integer ;
.
```

8.7.3. Property: geo:spatialDimension

The property `geo:spatialDimension` is defined to link a Geometry object to the dimension of the spatial portion of the direct positions (coordinate tuples) used in its serializations. If the direct positions do not carry a measure coordinate, this will be equal to the coordinate dimension.

```
geo:spatialDimension
    a rdf:Property, owl:DatatypeProperty;
    rdfs:isDefinedBy geo: ;
    skos:prefLabel "spatial dimension"@en ;
    skos:definition "The number of measurements or axes needed to describe the
                    spatial position of this Geometry in a coordinate system."@en ;
    rdfs:domain geo:Geometry ;
    rdfs:range xsd:integer ;
.
```

8.7.4. Property: geo:hasSpatialResolution

The property `geo:hasSpatialResolution` is defined to indicate spatial resolution of the elements within a Geometry. Spatial resolution specifies the level of detail of a Geometry. It the smallest dinstinghuishable distance between adjacent coordinate sets. Therefore this property is not applicable to a point Geometry, because it consists of a single coordinate set.

Since this property is defined for a `geo:Geometry`, all literal representations of that Geometry instance must have the same spatial resolution.

```
geo:hasSpatialResolution
    a rdf:Property, owl:ObjectProperty;
    rdfs:isDefinedBy geo: ;
    skos:prefLabel "has spatial resolution"@en ;
    skos:definition "The spatial resolution of a Geometry"@en ;
    rdfs:domain geo:Geometry ;
.
```

NOTE See the Section 8.3.

8.7.5. Property: geo:hasMetricSpatialResolution

The property `geo:hasMetricSpatialResolution` is similar to `geo:hasSpatialResolution`, specifies that the unit of resolution is always meter (the standard distance unit of the International System of Units).

```
geo:hasMetricSpatialResolution
    a rdf:Property, owl:ObjectProperty;
    rdfs:isDefinedBy geo: ;
    skos:prefLabel "has spatial resolution in meters"@en ;
    skos:definition "The spatial resolution of a Geometry in meters."@en ;
    rdfs:domain geo:Geometry ;
    rdfs:range xsd:double ;
.
```

8.7.6. Property: geo:hasSpatialAccuracy

The property `geo:hasSpatialAccuracy` is applicable when a Geometry is used to represent a Feature. It is expressed as a distance that indicates the truthfullness of the positions (coordinates) that define the Geometry. In this case accuracy defines a zone surrounding each coordinate within wich the real positions are known to be. The accuracy value defines this zone as a distance from the coordinate(s) in all directions (e.g. a line, a circle or a sphere, depending on spatial dimension).

```
geo:hasSpatialAccuracy
    a rdf:Property, owl:ObjectProperty;
    rdfs:isDefinedBy geo: ;
    skos:prefLabel "has spatial accuracy"@en ;
    skos:definition "The positional accuracy of the coordinates of a Geometry."@en ;
    rdfs:domain geo:Geometry ;
.
```

NOTE See the Section 8.3.

8.7.7. Property: geo:hasMetricSpatialAccuracy

The property `geo:hasMetricSpatialAccuracy` is similar to has spatial accuracy, but it is easier to specify and use because the unit of distance is always meter (the standard distance unit of the International System of Units).

```
geo:hasMetricSpatialAccuracy
    a rdf:Property, owl:ObjectProperty;
    rdfs:isDefinedBy geo: ;
    skos:prefLabel "has spatial accuracy in meters"@en ;
    skos:definition "The positional accuracy of the coordinates of a Geometry in
meters."@en ;
    rdfs:domain geo:Geometry ;
    rdfs:range xsd:double ;
.
```

8.7.8. Property: geo:isEmpty

The property `geo:isEmpty` will indicate a Boolean object set to `true` if and only if the Geometry contains no information.

```
geo:isEmpty
    a rdf:Property, owl:DatatypeProperty ;
    rdfs:isDefinedBy geo: ;
    skos:prefLabel "is empty"@en ;
    skos:definition "(true) if this geometric object is the empty Geometry. If
                    true, then this geometric object represents the empty point
                    set for the coordinate space."@en ;
    rdfs:domain geo:Geometry ;
    rdfs:range xsd:boolean ;
.
```

8.7.9. Property: geo:isSimple

The property `geo:isSimple` will indicate a Boolean object set to `true`, only if the Geometry contains no self-intersections, with the possible exception of its boundary.

```
geo:isSimple
    a rdf:Property, owl:DatatypeProperty ;
    rdfs:isDefinedBy geo: ;
    skos:prefLabel "is simple"@en ;
    skos:definition "(true) if this geometric object has no anomalous geometric
                    points, such as self intersection or self tangency."@en ;
    rdfs:domain geo:Geometry ;
    rdfs:range xsd:boolean ;
.
```

8.7.10. Property: geo:hasSerialization

The property `geo:hasSerialization` is defined to connect a Geometry with its text-based serialization (e.g., its WKT serialization).

```
geo:hasSerialization
    a rdf:Property, owl:DatatypeProperty ;
    rdfs:isDefinedBy geo: ;
    skos:prefLabel "has serialization"@en ;
    skos:definition "Connects a Geometry object with its text-based serialization."@en
;
    rdfs:domain geo:Geometry ;
    rdfs:range rdfs:Literal ;
.
```

NOTE this property is the generic property used to connect a Geometry with its serialization. GeoSPARQL also contains a number of sub properties of this one for connecting serializations of common types with geometries, for example as GeoJSON which can be used for GeoJSON [GEOJSON] literals.

8.8. Geometry Serializations

This section establishes the requirements for representing Geometry data in RDF literals, according to different non-RDF systems.

GeoSPARQL presents specializations of the `geo:hasSerialization` property for indicating particular serializations and specialized datatype literals for containing them but does not provide comprehensive definitions of their content since these are given in standards external to GeoSPARQL, all of which are referenced.

GeoSPARQL does present some requirements for literal structure which extend the serialization-defining standards, for example the requirement to allow indications of spatial reference systems within WKT geometry representations.

GeoSPARQL's expectation of RDF literal representations of geometry data is that it is related to the *Simple Features Access* (SFA) [ISO19125-1] standard's conceptualization of geometry which defines classes such as `Point`, `Curve` and `Surface` and specialised variants of them which it presents in a hierarchy. All SFA classes are represented in OWL in the *Simple Features Vocabulary* presented within GeoSPARQL as an independent profile element, see GeoSPARQL Standard structure.

Some geometry represenation systems given here do not use the same terminology as SFA, in particular the Discrete Global Grid System representation. To know the extent to which geometry literal representations listed here support SFA, or map to SFA, please see their definitions.

8.8.1. Well-Known Text (serialization=WKT)

This section establishes the requirements for representing Geometry data in RDF based on Well-Known Text (WKT) as defined by *Simple Features Access* [ISO19125-1]. It defines one RDFS Datatype: WKT Literal and one property, as WKT.

8.8.1.1. RDFS Datatype: geo:wktLiteral

The datatype `geo:wktLiteral` is used to contain the Well-Known Text (WKT) serialization of a Geometry.

```
geo:wktLiteral
    a rdfs:Datatype ;
    rdfs:isDefinedBy geo: ;
    skos:prefLabel "Well-known Text literal"@en ;
    skos:definition "A Well-known Text serialization of a Geometry object."@en ;
.
```

Req 15 All RDFS Literals of type `geo:wktLiteral` shall consist of an optional IRI identifying the coordinate reference system and a required Well Known Text (WKT) description of a geometric value. Valid `geo:wktLiteral` instances are formed by either a WKT string as defined in [ISO13249] or by concatenating a valid absolute IRI, as defined in [IETF3987], enclose in angled brackets (`<` & `>`) followed by a single space (Unicode U+0020 character) as a separator, and a WKT string as defined in [ISO13249].
`http://www.opengis.net/spec/geosparql/1.0/req/geometry-extension/wkt-literal`

The following *ABNF* [IETF5234] syntax specification formally defines this literal:

```
wktLiteral ::= opt-iri-and-space geometric-data

opt-iri-and-space = "<" IRI ">" LWSP / ""
```

The token `opt-iri-and-space` may be either an IRI and space or nothing (`""`), the token `IRI` (Internationalized Resource Identifier) is essentially a web address and is defined in [IETF3987] and the token `LWSP`, is one or more white space characters, as defined in [IETF5234]. `geometric-data` is the Well-Known Text representation of the Geometry, defined in [ISO13249].

In the absence of a leading spatial reference system IRI, the following spatial reference system IRI will be assumed: `<http://www.opengis.net/def/crs/OGC/1.3/CRS84>`. This IRI denotes WGS 84 longitude-latitude.

Req 16 The IRI `<http://www.opengis.net/def/crs/OGC/1.3/CRS84>` shall be assumed as the spatial reference system for `geo:wktLiteral` instances that do not specify an explicit spatial reference system IRI.
`http://www.opengis.net/spec/geosparql/1.0/req/geometry-extension/wkt-literal-default-srs`

The OGC maintains a set of SRS IRIs under the `http://www.opengis.net/def/crs/` namespace and IRIs from this set are recommended for use, however others may also be used, as long as they are valid IRIs.

Req 17 Coordinate tuples within `geo:wktLiteral` shall be interpreted using the axis order defined in the spatial reference system used.
`http://www.opengis.net/spec/geosparql/1.0/req/geometry-extension/wkt-axis-order`

The example WKT Literal below encodes a point Geometry using the default WGS84 geodetic longitude-latitude spatial reference system:

```
"Point(-83.38 33.95)"^^<http://www.opengis.net/ont/geosparql#wktLiteral>
```

A second example below encodes the same point as encoded in the example above but using a SRS identified by `http://www.opengis.net/def/SRS/EPSG/0/4326`: a WGS 84 geodetic latitude-longitude spatial reference system (note that this spatial reference system defines a different axis order):

```
"<http://www.opengis.net/def/crs/EPSG/0/4326> Point(33.95
-83.38)"^^<http://www.opengis.net/ont/geosparql#wktLiteral>
```

Req 18 An empty RDFS Literal of type `geo:wktLiteral` shall be interpreted as an empty Geometry.	
`http://www.opengis.net/spec/geosparql/1.0/req/geometry-extension/wkt-literal-empty`	

8.8.1.2. Property: geo:asWKT

The property `geo:asWKT` is defined to link a Geometry with its WKT serialization.

Req 19 Implementations shall allow the RDF property `geo:asWKT` to be used in SPARQL graph patterns.	
`http://www.opengis.net/spec/geosparql/1.0/req/geometry-extension/geometry-as-wkt-literal`	

```
geo:asWKT
    a rdf:Property, owl:DatatypeProperty ;
    rdfs:subPropertyOf geo:hasSerialization ;
    rdfs:isDefinedBy geo: ;
    skos:prefLabel "as WKT"@en ;
    skos:definition "The WKT serialization of a Geometry."@en ;
    rdfs:domain geo:Geometry ;
    rdfs:range geo:wktLiteral ;
.
```

8.8.1.3. Function: geof:asWKT

```
geof:asWKT (geom: ogc:geomLiteral): geo:wktLiteral
```

The function `geof:asWKT` converts `geom` to an equivalent WKT representation preserving the coordinate reference system.

Req 20 Implementations shall support `geo:asWKT` as a SPARQL extension function.	
`http://www.opengis.net/spec/geosparql/1.1/req/geometry-extension/asWKT-function`	

8.8.2. Geography Markup Language (serialization=GML)

This section establishes requirements for representing Geometry data in RDF based on GML as defined by Geography Markup Language Encoding Standard [OGC07-036]. It defines one RDFS

Datatype: GML Literal and one property, as GML.

8.8.2.1. RDFS Datatype: geo:gmlLiteral

The datatype `geo:gmlLiteral` is used to contain the Geography Markup Language (GML) serialization of a Geometry.

```
geo:gmlLiteral
    a rdfs:Datatype ;
    rdfs:isDefinedBy geo: ;
    skos:prefLabel "GML literal"@en ;
    skos:definition "The datatype of GML literal values"@en ;
.
```

Valid GML Literal instances are formed by encoding Geometry information as a valid element from the GML schema that implements a subtype of `GM_Object`. For example, in GML 3.2.1 this is every element directly or indirectly in the substitution group of the element `{http://www.opengis.net/ont/gml/3.2}AbstractGeometry`. In GML 3.1.1 and GML 2.1.2 this is every element directly or indirectly in the substitution group of the element `{http://www.opengis.net/ont/gml}_Geometry`.

Req 21 All `geo:gmlLiteral` instances shall consist of a valid element from the GML schema that implements a subtype of `GM_Object` as defined in [OGC07-036].	
`http://www.opengis.net/spec/geosparql/1.0/req/geometry-extension/gml-literal`	

The example GML Literal below encodes a point Geometry in the WGS 84 geodetic longitude-latitude spatial reference system using GML version 3.2:

```
"""
<gml:Point
        srsName=\"http://www.opengis.net/def/crs/OGC/1.3/CRS84\"
        xmlns:gml=\"http://www.opengis.net/ont/gml\">
    <gml:pos>-83.38 33.95</gml:pos>
</gml:Point>
"""^^<http://www.opengis.net/ont/geosparql#gmlLiteral>
```

Req 22 An empty `geo:gmlLiteral` shall be interpreted as an empty Geometry.
`http://www.opengis.net/spec/geosparql/1.0/req/geometry-extension/gml-literal-empty`

Req 23 Implementations shall document supported GML profiles.
`http://www.opengis.net/spec/geosparql/1.0/req/geometry-extension/gml-profile`

8.8.2.2. Property: geo:asGML

The property `geo:asGML` is defined to link a Geometry with its GML serialization.

Req 24 Implementations shall allow the RDF property `geo:asGML` to be used in SPARQL graph patterns.
`http://www.opengis.net/spec/geosparql/1.0/req/geometry-extension/geometry-as-gml-literal`

```
geo:asGML
    a rdf:Property ;
    rdfs:subPropertyOf geo:hasSerialization ;
    rdfs:isDefinedBy geo: ;
    skos:prefLabel "as GML"@en ;
    skos:definition "The GML serialization of a Geometry."@en ;
    rdfs:domain geo:Geometry ;
    rdfs:range geo:gmlLiteral ;
.
```

8.8.2.3. Function: geof:asGML

```
geof:asGML (geom: ogc:geomLiteral, gmlProfile: xsd:string): geo:gmlLiteral
```

The function `geof:asGML` converts `geom` to an equivalent GML representation defined by a gmlProfile version string preserving the coordinate reference system.

Req 25 Implementations shall support `geof:asGML` as a SPARQL extension function.
`http://www.opengis.net/spec/geosparql/1.1/req/geometry-extension/asGML-function`

8.8.3. GeoJSON (serialization=GEOJSON)

This section establishes requirements for representing Geometry data in RDF based on GeoJSON as defined by [GeoJSON]. It defines one RDFS Datatype: GeoJSON Literal and one property, as GeoJSON.

8.8.3.1. RDFS Datatype: geo:geoJSONLiteral

The datatype `geo:geoJSONLiteral` is used to contain the Geo JavaScript Object Notation (GeoJSON) serialization of a Geometry.

```
geo:geoJSONLiteral a rdfs:Datatype ;
    rdfs:isDefinedBy geo: ;
    skos:prefLabel "GeoJSON Literal"@en ;
    skos:definition "A GeoJSON serialization of a Geometry object."@en .
```

Valid GeoJSON Literal instances are formed by encoding Geometry information as a Geometry object as defined in the GeoJSON specification [GEOJSON].

Req 26 All `geo:geoJSONLiteral` instances shall consist of the Geometry objects as defined in the GeoJSON specification [GEOJSON].
`http://www.opengis.net/spec/geosparql/1.1/req/geometry-extension/geojson-literal`

Req 27 RDFS Literals of type `geo:geoJSONLiteral` do not contain a SRS definition. All literals of this type shall, according to the GeoJSON specification, be encoded only in, and be assumed to use, the WGS84 geodetic longitude-latitude spatial reference system (`http://www.opengis.net/def/crs/OGC/1.3/CRS84`).
`http://www.opengis.net/spec/geosparql/1.1/req/geometry-extension/geojson-literal-srs`

The example GeoJSON Literal below encodes a point Geometry using the default WGS84 geodetic longitude-latitude spatial reference system for Simple Features 1.0:

```
"""
{"type": "Point", "coordinates": [-83.38,33.95]}
"""^^<http://www.opengis.net/ont/geosparql#geoJSONLiteral>
```

Req 28 An empty RDFS Literal of type `geo:geoJSONLiteral` shall be interpreted as an empty Geometry, i.e. `{"geometry": null}` in GeoJSON .
`http://www.opengis.net/spec/geosparql/1.1/req/geometry-extension/geojson-literal-empty`

8.8.3.2. Property: geo:asGeoJSON

The property `geo:asGeoJSON` is defined to link a Geometry with its GeoJSON serialization.

Req 29 Implementations shall allow the RDF property `geo:asGeoJSON` to be used in SPARQL graph patterns.
`http://www.opengis.net/spec/geosparql/1.1/req/geometry-extension/geometry-as-geojson-literal`

```
geo:asGeoJSON
    a rdf:Property, owl:DatatypeProperty ;
    rdfs:subPropertyOf geo:hasSerialization ;
    rdfs:isDefinedBy geo: ;
    skos:prefLabel "as GeoJSON"@en ;
    skos:definition "The GeoJSON serialization of a Geometry."@en ;
    rdfs:domain geo:Geometry ;
    rdfs:range geo:geoJSONLiteral ;
.
```

8.8.3.3. Function: geof:asGeoJSON

```
geof:asGeoJSON (geom: ogc:geomLiteral): geo:geoJSONLiteral
```

The function `geof:asGeoJSON` converts `geom` to an equivalent GeoJSON representation. Coordinates are converted to the CRS84 coordinate system, the only valid coordinate system to be used in a GeoJSON literal.

Req 30 Implementations shall support `geof:asGeoJSON` as a SPARQL extension function.

http://www.opengis.net/spec/geosparql/1.1/req/geometry-extension/asGeoJSON-function

8.8.4. Keyhole Markup Language (serialization=KML)

This section establishes requirements for representing Geometry data in RDF based on KML as defined by [OGCKML]. It defines one RDFS Datatype: KML Literal and one property, as KML.

8.8.4.1. RDFS Datatype: geo:kmlLiteral

The datatype `geo:kmlLiteral` is used to contain the Keyhole Markup Language (KML) serialization of a Geometry.

```
geo:kmlLiteral
    a rdfs:Datatype ;
    rdfs:isDefinedBy geo: ;
    skos:prefLabel "KML Literal"@en ;
    skos:definition "A KML serialization of a Geometry object."@en ;
.
```

Valid KML Literal instances are formed by encoding Geometry information as a Geometry object as defined in the KML specification [OGCKML].

Req 31 All `geo:kmlLiteral` instances shall consist of the Geometry objects as defined in the KML specification [OGCKML].
http://www.opengis.net/spec/geosparql/1.1/req/geometry-extension/kml-literal

Req 32 RDFS Literals of type `geo:kmlLiteral` do not contain a SRS definition. All literals of this type shall according to the KML specification only be encoded in and assumed to use the WGS84 geodetic longitude-latitude spatial reference system (http://www.opengis.net/def/crs/OGC/1.3/CRS84).
http://www.opengis.net/spec/geosparql/1.1/req/geometry-extension/kml-literal-srs

The example KML Literal below encodes a point Geometry using the default WGS84 geodetic longitude-latitude spatial reference system for Simple Features 1.0:

```
"""
<Point xmlns=\"http://www.opengis.net/kml/2.2\">
    <coordinates>-83.38,33.95</coordinates>
</Point>
"""^^<http://www.opengis.net/ont/geosparql#kmlLiteral>
```

Req 33 An empty RDFS Literal of type `geo:kmlLiteral` shall be interpreted as an empty Geometry .
http://www.opengis.net/spec/geosparql/1.1/req/geometry-extension/kml-literal-empty

8.8.4.2. Property: geo:asKML

The property `geo:asKML` is defined to link a Geometry with its KML serialization.

Req 34 Implementations shall allow the RDF property `geo:asKML` to be used in SPARQL graph patterns.	
`http://www.opengis.net/spec/geosparql/1.1/req/geometry-extension/geometry-as-kml-literal`	

The property as KML is used to link a geometric element with its KML serialization.

```
geo:asKML
    a rdf:Property, owl:DatatypeProperty;
    rdfs:subPropertyOf geo:hasSerialization ;
    rdfs:isDefinedBy geo: ;
    skos:prefLabel "as KML"@en ;
    skos:definition "The KML serialization of a Geometry."@en ;
    rdfs:domain geo:Geometry ;
    rdfs:range geo:kmlLiteral ;
.
```

8.8.4.3. Function: geof:asKML

```
geof:asKML (geom: ogc:geomLiteral): geo:kmlLiteral
```

The function `geof:asKML` converts `geom` to an equivalent KML representation. Coordinates are converted to the CRS84 coordinate system, the only valid coordinate system to be used in a KML literal.

Req 35 Implementations shall support `geof:asKML` as a SPARQL extension function.	
`http://www.opengis.net/spec/geosparql/1.1/req/geometry-extension/asKML-function`	

8.8.5. Discrete Global Grid System (serialization=DGGS)

This section establishes the requirements for representing Discrete Global Grid System (DGGS) Geometry data as RDF literals. The form of representation is specific to individual DGGS implementations: known DGGSes are not compatible or even very similar.

Here is defined one RDFS Datatypes: `http://www.opengis.net/ont/geosparql#dggsLiteral` and one property, `http://www.opengis.net/ont/geosparql#asDGGS`.

NOTE The datatype defined here is for an abstract DGGS implementation (DGGS Literal) but concrete ones should be used in real implementations. For example, the AusPIX DGGS [AUSPIX] might implement something similar to `ex:auspixDggsLiteral`.

8.8.5.1. RDFS Datatype: geo:dggsLiteral

The datatype `geo:dggsLiteral` is used to contain the Discrete Global Grid System (DGGS) serialization of a Geometry.

```
geo:dggsLiteral
    a rdfs:Datatype ;
    rdfs:isDefinedBy geo: ;
    skos:prefLabel "DGGS Literal"@en ;
    skos:definition "A textual serialization of a Discrete Global Grid System (DGGS)
Geometry object."@en
.
```

Valid DGGS Literal instances are formed by encoding Geometry information according to specific DGGS implementation. The specific implementation should be indicated by use of a subclass of the `geo:dggsLiteral` datatype.

Req 36 All RDFS Literals of type `geo:dggsLiteral` shall consist of a DGGS Geometry serialization formulated according to a specific DGGS.
`http://www.opengis.net/spec/geosparql/1.1/req/geometry-extension/dggs-literal`

Req 37 An empty RDFS Literal of type `geo:dggsLiteral`, or one of its data subtypes, shall be interpreted as an empty `geo:Geometry`.
`http://www.opengis.net/spec/geosparql/1.1/req/geometry-extension/dggs-literal-empty`

An example of a literal for concrete DGGS, AusPIX, could be

```
ex:auspixDggsLiteral
    a rdfs:Datatype ;
    skos:prefLabel "AusPIX DGGS Literal"@en ;
    skos:definition "A textual serialization of an AusPIX Discrete Global Grid System
(DGGS) Geometry object."@en ;
.
```

A single *Cell* Geometry encoded according to the AusPIX DGGS using the example literal above is given below. The single cell value of *R3234* is analogous to either a `Point` or simple `Polygon` in WKT geometries.

```
"CellList (R3234)"^^<http://example.com#auspixDggsLiteral>
```

NOTE What `R3234` means, or the meaning of any other element within a concrete DGGS literal is not handled by GeoSPARQL but is expected to be handled by that DGGS' specification, just as GeoPSARQL does not delve into the internals of other Geometry formats such as WKT or GeoJSON.

8.8.5.2. Property: geo:asDGGS

The property `geo:asDGGS` is defined to link a Geometry with its DGGS serialization.

Req 38 Implementations shall allow the RDF property `geo:asDGGS` to be used in SPARQL graph patterns.
`http://www.opengis.net/spec/geosparql/1.1/req/geometry-extension/geometry-as-dggs-literal`

```
geo:asDGGS
    a rdf:Property, owl:DatatypeProperty ;
    rdfs:subPropertyOf geo:hasSerialization ;
    rdfs:isDefinedBy geo: ;
    skos:prefLabel "as DGGS"@en ;
    skos:definition "A DGGS serialization of a Geometry."@en ;
    rdfs:domain geo:Geometry ;
    rdfs:range geo:dggsLiteral ;
.
```

NOTE It is expected that this property will be used to indicate specific DGGS data types, such as the example `ex:auspixDggsLiteral`, described above, as opposed to the generic DGGS Literal.

8.8.5.3. Function: geof:asDGGS

```
geof:asDGGS (geom: ogc:geomLiteral, specificDggsDatatype: xsd:anyURI): geo:DggsLiteral
```

The function `geof:asDGGS` converts `geom` to an equivalent DGGS representation, formulated according to the specific DGGS literal indicated using the `specificDggsDatatype` parameter.

Req 39 Implementations shall support `geof:asDGGS` as a SPARQL extension function.
`http://www.opengis.net/spec/geosparql/1.1/req/geometry-extension/asDGGS-function`

8.9. Non-topological Query Functions

This clause defines SPARQL functions for performing non-topological spatial operations.

Req 40 Implementations shall support the functions `geof:boundary` `geof:metricBuffer`, `geof:buffer`, `geof:convexHull`, `geof:coordinateDimension`, `geof:difference`, `geof:dimension`, `geof:metricDistance`, `geof:distance`, `geof:envelope`, `geof:geometryType`, `geof:getSRID`, `geof:intersection`, `geof:is3D`, `geof:isEmpty`, `geof:isMeasured`, `geof:isSimple`, `geof:spatialDimension`, `geof:symDifference`, `geof:transform` and `geof:union` as SPARQL extension functions, consistent with definitions of these functions in Simple Features [ISO19125-1], for non-DGGS geometry literals
`http://www.opengis.net/spec/geosparql/1.1/req/geometry-extension/query-functions`

Req 41 Implementations shall support the functions `geof:area`, `geof:geometryN`, `geof:length`, `geof:maxX`, `geof:maxY`, `geof:maxZ`, `geof:minX`, `geof:minY`, `geof:minZ`, `geof:numGeometries` and `geof:projectTo`, as SPARQL extension functions which are defined in this standard, for non-DGGS geometry literals
`http://www.opengis.net/spec/geosparql/1.1/req/geometry-extension/query-functions-non-sf`

NOTE The requirements to support non-topological query functions for DGGS geometry literals are separated from the requirements to support them for traditional geometry literals as it is expected that implementing these functions for DGGS literals will be significantly more difficult. This is due to the novelty of DGGS literals and thus the lack of existing software libraries for their manipulation.

Req 42 Implementations shall support the functions of Requirement 40 for DGGS geometry literals as SPARQL extension functions, consistent with definitions of these functions in Simple Features [ISO19125-1], for non-DGGS geometry literals
`http://www.opengis.net/spec/geosparql/1.1/req/geometry-extension/query-functions-dggs`

Req 43 Implementations shall support the functions of Requirement 41 for DGGS geometry literals as SPARQL extension functions which are defined in this standard, for non-DGGS geometry literals
`http://www.opengis.net/spec/geosparql/1.1/req/geometry-extension/query-functions-non-sf-dggs`

Functions from both requirements above are listed below, alphabetically.

8.9.1. Function notes

These notes apply to all of the following functions in this section.

An invocation of any of the following functions with invalid arguments produces an error. An invalid argument includes any of the following:

- An argument of an unexpected type
- An invalid geometry literal value
- An non-fitting geometry type for the given function
- A geometry literal from a spatial reference system that is incompatible with the spatial reference system used for calculations
- An invalid units IRI

A more detailed description of expected inputs and expected outputs of the given functions is shown in Annex B.

Unless otherwise stated in the function definition, the following behaviors should be followed by all SPARQL extension functions defined in the GeoSPARQL standard:

- Functions returning a new geometry literal should follow the literal format of the first geometry literal input parameter. If no geometry literal input parameter is present, a WKT literal shall be returned

* Functions returning a new geometry literal should follow the SRS defined in the literal format of the first geometry literal input parameter. If no geometry literal input parameter is present, a geometry result should be returned in the CRS84 coordinate reference system

For further discussion of the effects of errors during FILTER evaluation, consult Section 17[10] of the SPARQL specification [SPARQL].

Note that returning values instead of raising an error serves as an extension mechanism of SPARQL.

From Section 17.3.1[11] of the SPARQL specification [SPARQL]:

> SPARQL language extensions may provide additional associations between operators and operator functions; ... No additional operator may yield a result that replaces any result other ... . The consequence of this rule is that SPARQL `FILTER` s will produce at least the same intermediate bindings after applying a `FILTER` as an unextended implementation.

This extension mechanism enables GeoSPARQL implementations to simultaneously support multiple geometry serializations. For example, a system that supports WKT Literal serializations may also support GML Literal serializations and consequently would not raise an error if it encounters multiple geometry datatypes while processing a given query.

NOTE	Several non-topological query functions use a unit of measure IRI. The OGC has recommended units of measure vocabularies for use, see the OGC Definitions Server[12].

8.9.2. Function: geof:area

```
geof:area (geom1: ogc:geomLiteral): xsd:double
```

Returns the area of `geom1` in square meters. Must return zero for all geometry types other than Polygon.

8.9.3. Function: geof:boundary

```
geof:boundary (geom1: ogc:geomLiteral): ogc:geomLiteral
```

This function returns the closure of the boundary of `geom1`. Calculations are in the spatial reference system of `geom1`.

8.9.4. Function: geof:metricBuffer

```
geof:metricBuffer (geom: ogc:geomLiteral,
                   radius: xsd:double): ogc:geomLiteral
```

Returns a geometric object that represents all Points whose distance from `geom` is less than or equal to the `radius` measured in meters. Calculations are in the coordinate reference system of `geom`. This function is similar to `geof:buffer`, but does not need a specification of distance unit.

8.9.5. Function: geof:buffer

```
geof:buffer (geom: ogc:geomLiteral,
             radius: xsd:double,
             units: xsd:anyURI): ogc:geomLiteral
```

Returns a geometric object that represents all Points whose distance from `geom` is less than or equal to the `radius` measured in `units`. Calculations are in the spatial reference system of `geom`. This function is similar to `geof:metricBuffer`, which does not need a specification of distance unit.

NOTE See the Recommendation for specification of units of measurement.

8.9.6. Function: geof:convexHull

```
geof:convexHull (geom: ogc:geomLiteral): ogc:geomLiteral
```

The function `geof:convexHull` returns a geometric object that represents all Points in the convex hull of `geom1`. Calculations are in the spatial reference system of `geom`.

8.9.7. Function: geof:coordinateDimension

```
geof:coordinateDimension (geom1: ogc:geomLiteral): xsd:integer
```

Returns the coordinate dimension of `geom1`.

8.9.8. Function: geof:difference

```
geof:difference (geom1: ogc:geomLiteral,
                 geom2: ogc:geomLiteral): ogc:geomLiteral
```

This function returns a geometric object that represents all Points in the set difference of `geom1` with `geom2`. Calculations are in the spatial reference system of `geom1`.

8.9.9. Function: geof:dimension

```
geof:dimension (geom1: ogc:geomLiteral): xsd:integer
```

Returns the dimension of `geom1`. In non-homogeneous geometry collections, this will return the largest topological dimension of the contained objects.

8.9.10. Function: geof:metricDistance

```
geof:metricDistance (geom1: ogc:geomLiteral,
                     geom2: ogc:geomLiteral): xsd:double
```

Returns the shortest distance in meters between any two Points in the two geometric objects. Calculations are in coordinate reference system of `geom1`. This function is similar to `geof:distance`, but does not need a specification of distance unit.

8.9.11. Function: geof:distance

```
geof:distance (geom1: ogc:geomLiteral,
               geom2: ogc:geomLiteral,
               units: xsd:anyURI): xsd:double
```

Returns the shortest distance in `units` between any two Points in the two geometric objects. Calculations are in spatial reference system of `geom1`. This function is similar to `geof:metricDistance`, which does not need a specification of distance unit.

NOTE See the Recommendation for specification of units of measurement.

8.9.12. Function: geof:envelope

```
geof:envelope (geom1: ogc:geomLiteral): ogc:geomLiteral
```

This function returns the minimum bounding box - a rectangle - of `geom1`. Calculations are in the spatial reference system of `geom1`.

8.9.13. Function: geof:geometryN

```
geof:geometryN (geom1: ogc:geomLiteral): xsd:integer
```

Returns the nth geometry of `geom1` if it is a GeometryCollection or `geom1` if it is a Geometry.

8.9.14. Function: geof:geometryType

```
geof:geometryType (geom1: ogc:geomLiteral): xsd:string
```

Returns the name of the subtype of Geometry of which this geometric object is an member. The name of the subtype of Geometry is returned as a string. No attempt to reconcile different geometry subtypes across all support literals need be made.

8.9.15. Function: geof:getSRID

```
geof:getSRID (geom: ogc:geomLiteral): xsd:anyURI
```

Returns the spatial reference system IRI for `geom`.

8.9.16. Function: geof:intersection

```
geof:intersection (geom1: ogc:geomLiteral,
                   geom2: ogc:geomLiteral): ogc:geomLiteral
```

Returns a geometric object that represents all Points in the intersection of `geom1` with `geom2`. Calculations are in the spatial reference system of `geom1`.

8.9.17. Function: geof:is3D

```
geof:is3D (geom1: ogc:geomLiteral): xsd:boolean
```

Returns true if `geom1` has z coordinate values.

8.9.18. Function: geof:isEmpty

```
geof:isEmpty (geom1: ogc:geomLiteral): xsd:boolean
```

Returns true if `geom1` is an empty geometry, i.e. contains no coordinates.

8.9.19. Function: geof:isMeasured

```
geof:isMeasured (geom1: ogc:geomLiteral): xsd:boolean
```

Returns true if `geom1` has m coordinate values.

8.9.20. Function: geof:isSimple

```
geof:isSimple (geom1: ogc:geomLiteral): xsd:boolean
```

Returns true if `geom1` is a simple geometry, i.e. has no anomalous geometric points, such as self intersection or self tangency.

8.9.21. Function: geof:length

```
geof:length (geom1: ogc:geomLiteral): xsd:double
```

Returns the length of `geom1` in meters. The longest length from any one dimension is returned.

8.9.22. Function: geof:maxX

```
geof:maxX (geom: ogc:geomLiteral): xsd:double
```

The function `geof:maxX` returns the maximum X coordinate for `geom`.

8.9.23. Function: geof:maxY

```
geof:maxY (geom: ogc:geomLiteral): xsd:double
```

The function `geof:maxY` returns the maximum Y coordinate for `geom`.

8.9.24. Function: geof:maxZ

```
geof:maxZ (geom: ogc:geomLiteral): xsd:double
```

The function `geof:maxZ` returns the maximum Z coordinate for `geom`.

8.9.25. Function: geof:minX

```
geof:minX (geom: ogc:geomLiteral): xsd:double
```

The function `geof:minX` returns the minimum X coordinate for `geom`.

8.9.26. Function: geof:minY

```
geof:minY (geom: ogc:geomLiteral): xsd:double
```

The function `geof:minY` returns the minimum Y coordinate for `geom`.

8.9.27. Function: geof:minZ

```
geof:minZ (geom: ogc:geomLiteral): xsd:double
```

The function `geof:minZ` returns the minimum Z coordinate for `geom`.

8.9.28. Function: geof:numGeometries

```
geof:numGeometries (geom1: ogc:geomLiteral): xsd:integer
```

Returns the number of geometries of `geom1`.

8.9.29. Function: geof:projectTo

```
geof:projectTo (geom1:  ogc:geomLiteral,
                dimVec: xsd:integer): ogc:geomLiteral
```

Projects the elements of geometry `geom1` to the dimensions specified by the `dimVec` "dimensional vector".

`dimVec` is a binary number indicating which dimensions to project to (retain) from the `geom`, for example, using dimensions x, y & z:

- "010" projects to the y dimension, only
- "110" projects to the x and y dimensions

A projection may only occur from a geometry to a geometry of equal or fewer dimensions and, so far, GeoSPARQL only allos 1-, 2- & 3-dimensional objects, thus `projectTo` can only be specified from 3 to 2, 2 to 1 or 1 to 0 dimensions.

8.9.30. Function: geof:spatialDimension

```
geof:spatialDimension (geom1: ogc:geomLiteral): xsd:integer
```

Returns the spatial dimension of `geom1`.

8.9.31. Function: geof:symDifference

```
geof:symDifference (geom1: ogc:geomLiteral,
                    geom2: ogc:geomLiteral): ogc:geomLiteral
```

This function returns a geometric object that represents all Points in the set symmetric difference of `geom1` with `geom2`. Calculations are in the spatial reference system of `geom1`.

8.9.32. Function: geof:transform

```
geof:transform (geom: ogc:geomLiteral, srsIRI: xsd:anyURI): ogc:geomLiteral
```

geof:transform converts `geom` to a spatial reference system defined by srsIRI. The function raises an error if a transformation is not mathematically possible.

NOTE	We recommend that implementers use the same literal type as a result of this function that is passed as a parameter to this function.

8.9.33. Function: geof:union

```
geof:union (geom1: ogc:geomLiteral,
            geom2: ogc:geomLiteral): ogc:geomLiteral
```

This function returns a geometric object that represents all Points in the union of `geom1` with `geom2`. Calculations are in the spatial reference system of `geom1`.

Req 44 Implementations shall support `geof:getSRID` as a SPARQL extension function.
`http://www.opengis.net/spec/geosparql/1.0/req/geometry-extension/srid-function`

8.10. Spatial Aggregate Functions

This clause defines SPARQL functions for performing spatial aggregations of data.

Req 45 Implementations shall support `geof:aggBoundingBox`, `geof:aggBoundingCircle`, `geof:aggCentroid`, `geof:aggConcatLines`, `geof:aggConcaveHull`, `geof:aggConvexHull` and `geof:aggUnion` as a SPARQL extension functions.
`http://www.opengis.net/spec/geosparql/1.1/req/geometry-extension/sa-functions`

8.10.1. Function: geof:aggBoundingBox

```
geof:aggBoundingBox (geom: ogc:geomLiteral): ogc:geomLiteral
```

The function `geof:aggBoundingBox` calculates a minimum bounding box - rectangle - of the set of given geometries.

8.10.2. Function: geof:aggBoundingCircle

```
geof:aggBoundingCircle (geom: ogc:geomLiteral): ogc:geomLiteral
```

The function `geof:aggBoundingCircle` calculates a minimum bounding circle of the set of given geometries.

8.10.3. Function: geof:aggCentroid

```
geof:aggCentroid (geom: ogc:geomLiteral): ogc:geomLiteral
```

The function `geof:aggCentroid` valculates the centroid of the set of given geometries.

8.10.4. Function: geof:aggConcatLines

```
geof:aggConcatLines (geom: ogc:geomLiteral): ogc:geomLiteral
```

The function `geof:aggConcatLines` Concatenates a set of LineStrings.

8.10.5. Function: geof:aggConcaveHull

```
geof:aggConcaveHull (geom: ogc:geomLiteral, targetPercent: xsd:double):
ogc:geomLiteral
```

The function `geof:aggConcaveHull` calculates the concave hull of the set of given geometries.

8.10.6. Function: geof:aggConvexHull

```
geof:aggConvexHull (geom: ogc:geomLiteral): ogc:geomLiteral
```

The function `geof:aggConvexHull` calculates the convex hull of the set of given geometries.

NOTE This function is similar in name to geof:aggConvexHull used to calculate the convex hull of just one geometry.

8.10.7. Function: geof:aggUnion

```
geof:aggUnion (geom: ogc:geomLiteral): ogc:geomLiteral
```

The function `geof:aggUnion` calculates the union of the set of given geometries.

NOTE This function is similar in name to geof:aggUnion used to calculate the union of just two geometries.

[3] http://www.w3.org/2003/01/geo/

[4] http://purl.org/dc/terms/Location

[5] https://schema.org/GeoCoordinates

[6] https://schema.org/GeoShape

[7] https://www.w3.org/ns/locn

[8] https://www.w3.org/TR/vocab-dcat/#spatial-properties

[9] http://www.qudt.org

[10] https://www.w3.org/TR/sparql11-query/#expressions

[11] https://www.w3.org/TR/sparql11-query/#operatorExtensibility

[12] https://www.ogc.org/def-server

9. Geometry Topology Extension (relation_family, serialization, version)

This clause establishes the *Geometry Topology Extension (relation_family, serialization, version)* parameterized requirements class, with IRI `/req/geometry-topology-extension`, which defines a collection of topological query functions that operate on geometry literals. This class is parameterized to give implementations flexibility in the topological relation families and geometry serializations that they choose to support. This requirements class has a single corresponding conformance class *Geometry Topology Extension (relation_family, serialization, version)*, with IRI `/conf/geometry-topology-extension`.

The Dimensionally Extended Nine Intersection Model (DE-9IM) [DE-9IM] has been used to define the relation tested by the query functions introduced in this section. Each query function is associated with a defining DE-9IM intersection pattern. Possible pattern values are:

- `-1` (empty)
- `0`, `1`, `2`, `T` (true) = `{0, 1, 2}`
- `F` (false) = `{-1}`
- `*` (don't care) = `{-1, 0, 1, 2}`

In the following descriptions, the notation `X/Y` is used denote applying a spatial relation to geometry types `X` and `Y` (i.e., `x` *relation* `y` where `x` is of type `X` and `y` is of type `Y`). The symbol `P` is used for 0-dimensional geometries (e.g. points). The symbol `L` is used for 1- dimensional geometries (e.g. lines), and the symbol `A` is used for 2-dimensional geometries (e.g. polygons). Consult the Simple Features specification [ISO19125-1] for a more detailed description of DE-9IM intersection patterns.

9.1. Parameters

- **relation_family**: Specifies the set of topological spatial relations to support.
- **serialization**: Specifies the serialization standard to use for geometry literals.
- **version**: Specifies the version of the serialization format used.

9.2. Common Query Functions

Req 46 Implementations shall support `geof:relate` as a SPARQL extension function, consistent with the relate operator defined in Simple Features [ISO19125-1].
`http://www.opengis.net/spec/geosparql/1.0/req/geometry-topology-extension/relate-query-function`

```
geof:relate (geom1: ogc:geomLiteral,
             geom2: ogc:geomLiteral,
             pattern-matrix: xsd:string): xsd:boolean
```

Returns `true` if the spatial relationship between `geom1` and `geom2` corresponds to one with acceptable values for the specified pattern-matrix. Otherwise, this function returns `false`. `pattern-matrix` represents a DE-9IM intersection pattern consisting of `T` (true) and `F` (false) values. The spatial reference system for `geom1` is used for spatial calculations.

9.3. Simple Features Relation Family (relation_family=Simple Features)

This clause establishes requirements for the *Simple Features* relation family.

Req 47 Implementations shall support `geof:sfEquals`, `geof:sfDisjoint`, `geof:sfIntersects`, `geof:sfTouches`, `geof:sfCrosses`, `geof:sfWithin`, `geof:sfContains` and `geof:sfOverlaps` as SPARQL extension functions, consistent with their corresponding DE-9IM intersection patterns, as defined by Simple Features [ISO19125-1].	
`http://www.opengis.net/spec/geosparql/1.0/req/geometry-topology-extension/sf-query-functions`	

Boolean query functions defined for the Simple Features relation family, along with their associated DE-9IM intersection patterns, are shown in Table 6 below. Multi-row intersection patterns should be interpreted as a logical OR of each row. Each function accepts two arguments (`geom1` and `geom2`) of the geometry literal *serialization* type *specified* by serialization and version. Each function returns an `xsd:boolean` value of `true` if the specified relation exists between `geom1` and `geom2` and returns false otherwise. In each case, the spatial reference system of `geom1` is used for spatial calculations.

Table 6. Simple Features Query Functions

Query Function	Defining DE-9IM Intersection Pattern
`geof:sfEquals(geom1: ogc:geomLiteral, geom2: ogc:geomLiteral): xsd:boolean`	`(TFFFTFFFT)`
`geof:sfDisjoint(geom1: ogc:geomLiteral, geom2: ogc:geomLiteral): xsd:boolean`	`(FF*FF****)`
`geof:sfIntersects(geom1: ogc:geomLiteral, geom2: ogc:geomLiteral): xsd:boolean`	`(FT******* F**T***** F***T****)`
`geof:sfTouches(geom1: ogc:geomLiteral, geom2: ogc:geomLiteral): xsd:boolean`	`(FT******* F**T***** F***T****)`
`geof:sfCrosses(geom1: ogc:geomLiteral, geom2: ogc:geomLiteral): xsd:boolean`	`(T*T***T**) for P/L, P/A, L/A; (0*T***T**) for L/L`
`geof:sfWithin(geom1: ogc:geomLiteral, geom2: ogc:geomLiteral): xsd:boolean`	`(T*F**F***)`
`geof:sfContains(geom1: ogc:geomLiteral, geom2: ogc:geomLiteral): xsd:boolean`	`(T*****FF*)`
`geof:sfOverlaps(geom1: ogc:geomLiteral, geom2: ogc:geomLiteral): xsd:boolean`	`(T*T***T**) for A/A, P/P; (1*T***T**) for L/L`

9.4. Egenhofer Relation Family (relation_family=Egenhofer)

This clause establishes requirements for the *Egenhofer* relation family. Consult references [FORMAL] and [CATEG] for a more detailed discussion of *Egenhofer* relations.

Req 48 Implementations shall support `geof:ehEquals`, `geof:ehDisjoint`, `geof:ehMeet`, `geof:ehOverlap`, `geof:ehCovers`, `geof:ehCoveredBy`, `geof:ehInside` and `geof:ehContains` as SPARQL extension functions, consistent with their corresponding DE-9IM intersection patterns, as defined by Simple Features [ISO19125-1].
`http://www.opengis.net/spec/geosparql/1.0/req/geometry-topology-extension/eh-query-functions`

Boolean query functions defined for the *Egenhofer* relation family, along with their associated DE-9IM intersection patterns, are shown in Table 7 below. Multi-row intersection patterns should be interpreted as a logical OR of each row. Each function accepts two arguments (`geom1` and `geom2`) of the geometry literal serialization type specified by *serialization* and *version*. Each function returns an `xsd:boolean` value of `true` if the specified relation exists between `geom1` and `geom2` and returns `false` otherwise. In each case, the spatial reference system of geom1 is used for spatial calculations.

Table 7. Egenhofer Query Functions

Query Function	Defining DE-9IM Intersection Pattern
`geof:ehEquals(geom1: ogc:geomLiteral, geom2: ogc:geomLiteral): xsd:boolean`	`(TFFFTFFFT)`
`geof:ehDisjoint(geom1: ogc:geomLiteral, geom2: ogc:geomLiteral): xsd:boolean`	`(FF*FF****)`
`geof:ehMeet(geom1: ogc:geomLiteral, geom2: ogc:geomLiteral): xsd:boolean`	`(FT******* F**T***** F***T****)`
`geof:ehOverlap(geom1: ogc:geomLiteral, geom2: ogc:geomLiteral): xsd:boolean`	`(T*T***T**)`
`geof:ehCovers(geom1: ogc:geomLiteral, geom2: ogc:geomLiteral): xsd:boolean`	`(T*TFT*FF*)`
`geof:ehCoveredBy(geom1: ogc:geomLiteral, geom2: ogc:geomLiteral): xsd:boolean`	`(TFF*TFT**)`
`geof:ehInside(geom1: ogc:geomLiteral, geom2: ogc:geomLiteral): xsd:boolean`	`(TFF*FFT**)`
`geof:ehContains(geom1: ogc:geomLiteral, geom2: ogc:geomLiteral): xsd:boolean`	`(T*TFF*FF*)`

9.5. Requirements for RCC8 Relation Family (relation_family=RCC8)

This clause establishes requirements for the *RCC8* relation family. Consult references [QUAL] and [LOGIC] for a more detailed discussion of *RCC8* relations.

Req 49 Implementations shall support `geof:rcc8eq`, `geof:rcc8dc`, `geof:rcc8ec`, `geof:rcc8po`, `geof:rcc8tppi`, `geof:rcc8tpp`, `geof:rcc8ntpp` and `geof:rcc8ntppi` as SPARQL extension functions, consistent with their corresponding DE-9IM intersection patterns, as defined by Simple Features [ISO19125-1].
`http://www.opengis.net/spec/geosparql/1.0/req/geometry-topology-extension/rcc8-query-functions`

Boolean query functions defined for the *RCC8* relation family, along with their associated DE-9IM intersection patterns, are shown in Table 8 below. Each function accepts two arguments (`geom1` and `geom2`) of the geometry literal serialization type specified by *serialization* and *version*. Each function returns an `xsd:boolean` value of `true` if the specified relation exists between `geom1` and `geom2` and returns `false` otherwise. In each case, the spatial reference system of geom1 is used for spatial calculations.

Table 8. RCC8 Query Functions

Query Function	Defining DE-9IM Intersection Pattern
`geof:rcc8eq(geom1: ogc:geomLiteral, geom2: ogc:geomLiteral): xsd:boolean`	`(TFFFTFFFT)`
`geof:rcc8dc(geom1: ogc:geomLiteral, geom2: ogc:geomLiteral): xsd:boolean`	`(FFTFFTTTT)`
`geof:rcc8ec(geom1: ogc:geomLiteral, geom2: ogc:geomLiteral): xsd:boolean`	`(FFTFTTTTT)`
`geof:rcc8po(geom1: ogc:geomLiteral, geom2: ogc:geomLiteral): xsd:boolean`	`(TTTTTTTTT)`
`geof:rcc8tppi(geom1: ogc:geomLiteral, geom2: ogc:geomLiteral): xsd:boolean`	`(TTTFTTFFT)`
`geof:rcc8tpp(geom1: ogc:geomLiteral, geom2: ogc:geomLiteral): xsd:boolean`	`(TFFTTFTTT)`
`geof:rcc8ntpp(geom1: ogc:geomLiteral, geom2: ogc:geomLiteral): xsd:boolean`	`(TFFTFFTTT)`
`geof:rcc8ntppi(geom1: ogc:geomLiteral, geom2: ogc:geomLiteral): xsd:boolean`	`(TTTFFTFFT)`

10. RDFS Entailment Extension (relation_family, serialization, version)

This clause establishes the *RDFS Entailment Extension (relation_family, serialization, version)* parameterized requirements class, with IRI `/req/rdfs-entailment-extension`, which defines a mechanism for matching implicitly-derived RDF triples in GeoSPARQL queries. This class is parameterized to give implementations flexibility in the topological relation families and geometry types that they choose to support. This requirements class has a single corresponding conformance class *RDFS Entailment Extension (relation_family, serialization, version)*, `with IRI /conf/rdfs-entailment-extension`.

10.1. Parameters

- **relation_family**: Specifies the set of topological spatial relations to support.
- **serialization**: Specifies the serialization standard to use for geometry literals.
- **version**: Specifies the version of the serialization format used.

10.2. Common Requirements

The basic mechanism for supporting RDFS entailment has been defined by the W3C SPARQL 1.1 RDFS Entailment Regime [SPARQLENT].

Req 50 Basic graph pattern matching shall use the semantics defined by the RDFS Entailment Regime [SPARQLENT].
`http://www.opengis.net/spec/geosparql/1.0/req/rdfs-entailment-extension/bgp-rdfs-ent`

10.3. WKT Serialization (serialization=WKT)

This section establishes the requirements for representing geometry data in RDF based on WKT as defined by Simple Features [ISO19125-1].

10.3.1. Geometry Class Hierarchy

The Simple Features specification presents a geometry class hierarchy. It is straightforward to represent this class hierarchy in RDFS and OWL by constructing IRIs for geometry classes using the following pattern: `http://www.opengis.net/ont/sf#{geometry class}` and by asserting appropriate `rdfs:subClassOf` statements. The *Simple Features Vocabulary* resource within GeoSPARQL 1.1 (simpling resource to this specification) does this. The following list gived the class hierarchy with each indented item being a subclass of the item in the line above. The class heirarchy starts with GeoSPARQL's `geo:Geometry` class of which `sf:Geometry` is a subclass:

```
geo:Geometry
    sf:Geometry
        sf:Curve
            sf:LineString
                sf:Line
                sf:LinearRing
        sf:GeometryCollection
            sf:MultiCurve
                sf:MultiLineString
            sf:MultiPoint
            sf:MultiSurface
                sf:MultiPolygon
        sf:Point
        sf:Surface
            sf:Polygon
                sf:Envelope
                sf:Triangle
            sf:PolyhedralSurface
                sf:TIN
```

The following example RDF snippet below encodes the *Simple Features vocabulary* Polygon class:

```
sf:Polygon
    a rdfs:Class, owl:Class ;
    rdfs:isDefinedBy <http://www.opengis.net/ont/sf> ;
    skos:prefLabel "Polygon"@en ;
    rdfs:subClassOf sf:Surface ;
    skos:definition "A planar surface defined by 1 exterior boundary and 0 or
                    more interior boundaries"@en ;
.
```

Req 51 Implementations shall support graph patterns involving terms from an RDFS/OWL class hierarchy of geometry types consistent with the one in the specified version of Simple Features [ISO19125-1].
`http://www.opengis.net/spec/geosparql/1.0/req/rdfs-entailment-extension/wkt-geometry-types`

10.4. GML Serialization (serialization=GML)

This section establishes requirements for representing geometry data in RDF based on GML as defined by Geography Markup Language Encoding Standard [OGC07-036].

10.4.1. Geometry Class Hierarchy

An RDF/OWL class hierarchy can be generated from the GML schema that implements `GM_Object` by constructing IRIs for geometry classes using the following pattern: `http://www.opengis.net/ont/gml#{GML Element}` and by asserting appropriate `rdfs:subClassOf` statements.

The example RDF snippet below encodes the Polygon class from GML 3.2.

```
gml:Polygon
    a rdfs:Class, owl:Class ;
    skos:prefLabel "Polygon"@en ;
    rdfs:subClassOf gml:SurfacePatch ;
    skos:definition "A planar surface defined by 1 exterior boundary and 0 or
                    more interior boundaries."@en ;
.
```

Req 52 Implementations shall support graph patterns involving terms from an RDFS/OWL class hierarchy of geometry types consistent with the GML schema that implements `GM_Object` using the specified *version* of GML [OGC07-036].
`http://www.opengis.net/spec/geosparql/1.0/req/rdfs-entailment-extension/gml-geometry-types`

11. Query Rewrite Extension (relation_family, serialization, version)

This clause establishes the *Query Rewrite Extension (relation_family, serialization, version)* parameterized requirements class, with IRI `/req/query-rewrite-extension`, which has a single corresponding conformance class *Query Rewrite Extension (relation_family, serialization, version)*, with IRI `/conf/query-rewrite-extension`. This requirements class defines a set of RIF rules [RIF] that use topological extension functions defined in Clause 9 to establish the existence of direct topological predicates defined in Clause 7. One possible implementation strategy is to transform a given query by expanding a triple pattern involving a direct spatial predicate into a series of triple patterns and an invocation of the corresponding extension function as specified in the RIF rule.

The following rule specified using the RIF Core Dialect [RIFCORE] is used as a template to describe rules in the remainder of this clause. `ogc:relation` is used as a placeholder for the spatial relation IRIs defined in Clause 7, and `ogc:function` is used as a placeholder for the spatial functions defined in Clause 9.

```
Forall ?f1 ?f2 ?g1 ?g2 ?g1Serial ?g2Serial
    (?f1[ogc:relation->?f2] :-
        Or(
            And
                # feature □ feature rule
                (?f1[geo:hasDefaultGeometry->?g1]
                 ?f2[geo:hasDefaultGeometry->?g2]
                 ?g1[ogc:asGeomLiteral->?g1Serial]
                 ?g2[ogc:asGeomLiteral->?g2Serial]
                 External(ogc:function (?g1Serial,?g2Serial)))
            And
                # feature □ geometry rule
                (?f1[geo:hasDefaultGeometry->?g1]
                 ?g1[ogc:asGeomLiteral->?g1Serial]
                 ?f2[ogc:asGeomLiteral->?g2Serial]
                 External(ogc:function (?g1Serial,?g2Serial)))
            And
                # geometry - feature rule
                (?f2[geo:hasDefaultGeometry->?g2]
                 ?f1[ogc:asGeomLiteral->?g1Serial]
                 ?g2[ogc:asGeomLiteral->?g2Serial]
                 External(ogc:function (?g1Serial,?g2Serial)))
            And
                # geometry - geometry rule
                (?f1[ogc:asGeomLiteral->?g1Serial]
                 ?f2[ogc:asGeomLiteral->?g2Serial]
                 External(ogc:function (?g1Serial,?g2Serial)))
    )
)
```

NOTE The GeoSPARQL 1.1 Standard contains a RIF rules artefact expanded for all function generated from this template and Python software for re-issuing the expanded artefact. See GeoSPARQL Standard structure.

11.1. Parameters

- **relation_family**: Specifies the set of topological spatial relations to support.
- **serialization**: Specifies the serialization standard to use for geometry literals.
- **version**: Specifies the version of the serialization format used.

11.2. Simple Features Relation Family (relation_family=Simple Features)

This clause defines requirements for the *Simple Features* relation family. Table 9 specifies the function and property substitutions for each rule in the *Simple Features* relation family.

Req 53 Basic graph pattern matching shall use the semantics defined by the RIF Core Entailment Regime [SPARQLENT] for the RIF rules [RIFCORE] `geor:sfEquals`, `geor:sfDisjoint`, `geor:sfIntersects`, `geor:sfTouches`, `geor:sfCrosses`, `geor:sfWithin`, `geor:sfContains` and `geor:sfOverlaps`.
`http://www.opengis.net/spec/geosparql/1.0/req/query-rewrite-extension/sf-query-rewrite`

Table 9. Simple Features Query Transformation Rules

Rule	ogc:relation	ogc:function
`geor:sfEquals`	`geo:sfEquals`	`geof:sfEquals`
`geor:sfDisjoint`	`geo:sfDisjoint`	`geof:sfDisjoint`
`geor:sfIntersects`	`geo:sfIntersects`	`geof:sfIntersects`
`geor:sfTouches`	`geo:sfTouches`	`geof:sfTouches`
`geor:sfCrosses`	`geo:sfCrosses`	`geof:sfCrosses`
`geor:sfWithin`	`geo:sfWithin`	`geof:sfWithin`
`geor:sfContains`	`geo:sfContains`	`geof:sfContains`
`geor:sfOverlaps`	`geo:sfOverlaps`	`geof:sfOverlaps`

11.3. Egenhofer Relation Family (relation_family=Egenhofer)

This clause defines requirements for the *Egenhofer* relation family. Table 10 specifies the function and property substitutions for each rule in the *Egenhofer* relation family.

Req 54 Basic graph pattern matching shall use the semantics defined by the RIF Core Entailment Regime [SPARQLENT] for the RIF rules [RIFCORE] `geor:ehEquals`, `geor:ehDisjoint`, `geor:ehMeet`, `geor:ehOverlap`, `geor:ehCovers`, `geor:ehCoveredBy`, `geor:ehInside` and `geor:ehContains`.

http://www.opengis.net/spec/geosparql/1.0/req/query-rewrite-extension/eh-query-rewrite

Table 10. Egenhofer Query Transformation Rules

Rule	ogc:relation	ogc:function
`geor:ehEquals`	`geo:ehEquals`	`geof:ehEquals`
`geor:ehDisjoint`	`geo:ehDisjoint`	`geof:ehDisjoint`
`geor:ehMeet`	`geo:ehMeet`	`geof:ehMeet`
`geor:ehOverlap`	`geo:ehOverlap`	`geof:ehOverlap`
`geor:ehCovers`	`geo:ehCovers`	`geof:ehCovers`
`geor:ehCoveredBy`	`geo:ehCoveredBy`	`geof:ehCoveredBy`
`geor:ehInside`	`geo:ehInside`	`geof:ehInside`
`geor:ehContains`	`geo:ehContains`	`geof:ehContains`

11.4. RCC8 Relation Family (relation_family=RCC8)

This clause defines requirements for the *RCC8* relation family. Table 11 specifies the function and property substitutions for each rule in the *RCC8* relation family.

Req 55 Basic graph pattern matching shall use the semantics defined by the RIF Core Entailment Regime [SPARQLENT] for the RIF rules [RIFCORE] `geor:rcc8eq`, `geor:rcc8dc`, `geor:rcc8ec`, `geor:rcc8po`, `geor:rcc8tppi`, `geor:rcc8tpp`, `geor:rcc8ntpp` and `geor:rcc8ntppi`.
http://www.opengis.net/spec/geosparql/1.0/req/query-rewrite-extension/rcc8-query-rewrite

Table 11. RCC8 Query Transformation Rules

Rule	ogc:relation	ogc:function
`geor:rcc8eq`	`geo:rcc8eq`	`geof:rcc8eq`
`geor:rcc8dc`	`geo:rcc8dc`	`geof:rcc8dc`
`geor:rcc8ec`	`geo:rcc8ec`	`geof:rcc8ec`
`geor:rcc8po`	`geo:rcc8po`	`geof:rcc8po`
`geor:rcc8tppi`	`geo:rcc8tppi`	`geof:rcc8tppi`
`geor:rcc8tpp`	`geo:rcc8tpp`	`geof:rcc8tpp`
`geor:rcc8ntpp`	`geo:rcc8ntpp`	`geof:rcc8ntpp`
`geor:rcc8ntppi`	`geo:rcc8ntppi`	`geof:rcc8ntppi`

11.5. Special Considerations

The applicability of GeoSPARQL rules in certain circumstances has intentionally been left undefined.

The first situation arises for triple patterns with unbound predicates. Consider the query pattern below:

```
{ my:feature1 ?p my:feature2 }
```

When using a query transformation strategy, this triple pattern could invoke none of the GeoSPARQL rules or all of the rules. Implementations are free to support either of these alternatives.

The second situation arises when supporting GeoSPARQL rules in the presence of RDFS Entailment. The existence of a topological relation (possibly derived from a GeoSPARQL rule) can entail other RDF triples. For example, if `geo:sfOverlaps` has been defined as an `rdfs:subPropertyOf` the property `my:overlaps`, and the RDF triple `my:feature1 geo:sfOverlaps my:feature2` has been derived from a GeoSPARQL rule, then the RDF triple `my:feature1 my:overlaps my:feature2` can be entailed. Implementations may support such entailments but are not required to.

12. Future Work

Many future extensions of this standard are possible and, since the release of GeoSPARQL 1.0, many extensions have been made.

The GeoSPARQL 1.1 release incorporates many additions requested of the GeoSPARQL 1.0 Standard, including the use of particular new serializations: where GeoSPARQL 1.0 supported GML & WKT, GeoSPARQL 1.1 also supports GeoJSON, KML and a generic DGGS literal. GeoSPARQL 1.1 also supports spatial scalar properties.

Plans for future GeoSPARQL releases have been mooted but won’t be articulated here, instead they will be discussed and decided apon by the OGC GeoSPARQL Standards Working Group and related groups. Readers of this document are encouraged to seek out those groups' lists of issues and standards change requests rather than looking for ideas here that will surely age badly.

Annex A - Abstract Test Suite (normative)

A.0 Overview

This Annex lists tests for the Conformance Classes defined in the main body sections of this Specification with links to their Requirements and test purpose method and type.

A.1 Conformance Class: Core

Conformance Class IRI: `/conf/core`

A.1.1 SPARQL

A.1.1.1 `/conf/core/sparql-protocol`

Requirement: `/req/core/sparql-protocol`

Implementations shall support the SPARQL Query Language for RDF [SPARQL], the SPARQL Protocol for RDF [SPARQLPROT] and the SPARQL Query Results XML Format [SPARQLRESX].

a. **Test purpose**: Check conformance with this requirement

b. **Test method**: Verify that the implementation accepts SPARQL queries and returns the correct results in the correct format, according to the SPARQL Query Language for RDF, the SPARQL Protocol for RDF and SPARQL Query Results XML Format W3C specifications.

c. **Reference**: Section 4.1.4

d. **Test Type**: Capabilities

A.1.2 RDF Classes & Properties

A.1.2.1 `/conf/core/spatial-object-class`

Requirement: `/req/core/spatial-object-class`

Implementations shall allow the RDFS class `geo:SpatialObject` to be used in SPARQL graph patterns.

a. **Test purpose**: Check conformance with this requirement

b. **Test method**: Verify that queries involving `geo:SpatialObject` return the correct result on a test dataset.

c. **Reference**: Section 6.2.1

d. **Test Type**: Capabilities

A.1.2.2 `/conf/core/feature-class`

Requirement: `/req/core/feature-class` Implementations shall allow the RDFS class `geo:Feature` to be used in SPARQL graph patterns.

a. **Test purpose**: check conformance with this requirement

b. **Test method**: verify that queries involving `geo:Feature` return the correct result on a test dataset.

c. **Reference**: Section 6.2.2

d. **Test Type**: Capabilities

A.1.2.3 `/conf/core/spatial-object-collection-class`

Requirement: `/req/core/spatial-object-collection-class`

Implementations shall allow the RDFS class `geo:SpatialObjectCollection` to be used in SPARQL graph patterns.

a. **Test purpose**: check conformance with this requirement

b. **Test method**: verify that queries involving `geo:SpatialObjectCollection` return the correct result on a test dataset.

c. **Reference**: Section 6.2.3

d. **Test Type**: Capabilities

A.1.2.4 `/conf/core/feature-collection-class`

Requirement: `/req/core/feature-collection-class`

Implementations shall allow the RDFS class `geo:FeatureCollection` to be used in SPARQL graph patterns.

a. **Test purpose**: check conformance with this requirement

b. **Test method**: verify that queries involving `geo:FeatureCollection` return the correct result on a test dataset.

c. **Reference**: Section 6.2.4

d. **Test Type**: Capabilities

A.1.2.5 `/conf/core/spatial-object-properties`

Implementations shall allow the properties `geo:hasSize`, `geo:hasMetricSize`, `geo:hasLength`, `geo:hasMetricLength`, `geo:hasPerimeterLength`, `geo:hasMetricPerimeterLength`, `geo:hasArea`, `geo:hasMetricArea`, `geo:hasVolume` and `geo:hasMetricVolume`. to be used in SPARQL graph patterns.

a. **Test purpose**: Check conformance with this requirement

b. **Test method**: Verify that queries involving these properties return the correct result for a test dataset.

c. **Reference**: Section 6.3

d. **Test Type**: Capabilities

A.2 Conformance Class: Topology Vocabulary Extension (relation_family)

Conformance Class IRI: `/conf/topology-vocab-extension`

A.2.1 relation_family = Simple Features

A.2.1.1 `/conf/topology-vocab-extension/sf-spatial-relations`

Requirement: `/req/topology-vocab-extension/sf-spatial-relations`

Implementations shall allow the properties `geo:sfEquals`, `geo:sfDisjoint`, `geo:sfIntersects`, `geo:sfTouches`, `geo:sfCrosses`, `geo:sfWithin`, `geo:sfContains` and `geo:sfOverlaps` to be used in SPARQL graph patterns.

a. **Test purpose**: Check conformance with this requirement

b. **Test method**: Verify that queries involving these properties return the correct result for a test dataset.

c. **Reference**: Section 7.2

d. **Test Type**: Capabilities

A.2.2 relation_family = Egenhofer

A.2.2.1 `/conf/topology-vocab-extension/eh-spatial-relations`

Requirement: `/req/topology-vocab-extension/eh-spatial-relations`

Implementations shall allow the properties `geo:ehEquals`, `geo:ehDisjoint`, `geo:ehMeet`, `geo:ehOverlap`, `geo:ehCovers`, `geo:ehCoveredBy`, `geo:ehInside` and `geo:ehContains` to be used in SPARQL graph patterns.

a. **Test purpose**: Check conformance with this requirement

b. **Test method**: Verify that queries involving these properties return the correct result for a test dataset.

c. **Reference**: Section 7.3

d. **Test Type**: Capabilities

A.2.3 relation_family = RCC8

A.2.3.1 `/conf/topology-vocab-extension/rcc8-spatial-relations`

Requirement: `/req/topology-vocab-extension/rcc8-spatial-relations`

Implementations shall allow the properties `geo:rcc8eq`, `geo:rcc8dc`, `geo:rcc8ec`, `geo:rcc8po`, `geo:rcc8tppi`, `geo:rcc8tpp`, `geo:rcc8ntpp`, `geo:rcc8ntppi` to be used in SPARQL graph patterns.

a. **Test purpose**: Check conformance with this requirement
b. **Test method**: Verify that queries involving these properties return the correct result for a test dataset.
c. **Reference**: Section 7.4
d. **Test Type**: Capabilities

A.3 Conformance Class: Geometry Extension (serialization, version)

Conformance Class IRI: `/conf/geometry-extension`

A.3.1 Tests for all Serializations

A.3.1.1 `/conf/geometry-extension/geometry-class`

Requirement: `/req/geometry-extension/geometry-class`

Implementations shall allow the RDFS class `geo:Geometry` to be used in SPARQL graph patterns.

a. **Test purpose**: Check conformance with this requirement
b. **Test method**: Verify that queries involving `geo:Geometry` return the correct result on a test dataset
c. **Reference**: Section 8.6.1
d. **Test Type**: Capabilities

A.3.1.6 `/conf/geometry-extension/geometry-collection-class`

Requirement: `/req/geometry-extension/geometry-collection-class`

Implementations shall allow the RDFS class Geometry Collection to be used in SPARQL graph patterns.

a. **Test purpose**: check conformance with this requirement
b. **Test method**: verify that queries involving Geometry Collection return the correct result on a test dataset
c. **Reference**: Section 8.6.2
d. **Test Type**: Capabilities

A.3.1.2 `/conf/geometry-extension/feature-properties`

Requirement: `/req/geometry-extension/feature-properties`

Implementations shall allow the properties `geo:hasGeometry`, `geo:hasDefaultGeometry`, `geo:hasLength`, `geo:hasArea`, `geo:hasVolume` `geo:hasCentroid`, `geo:hasBoundingBox` and `geo:hasSpatialResolution` to be used in SPARQL graph patterns.

a. **Test purpose**: Check conformance with this requirement
b. **Test method**: Verify that queries involving these properties return the correct result for a test dataset.
c. **Reference**: Section 6.4

d. **Test Type**: Capabilities

A.3.1.3 `/conf/geometry-extension/geometry-properties`

Requirement: `/req/geometry-extension/geometry-properties`

Implementations shall allow the properties `geo:dimension`, `geo:coordinateDimension`, `geo:spatialDimension`, `geo:isEmpty`, `geo:isSimple` and `geo:hasSerialization` to be used in SPARQL graph patterns.

a. **Test purpose**: Check conformance with this requirement

b. **Test method**: Verify that queries involving these properties return the correct result for a test dataset.

c. **Reference**: Section 8.7

d. **Test Type**: Capabilities

A.3.1.4 `/conf/geometry-extension/query-functions`

Requirement: `/req/geometry-extension/query-functions`

Implementations shall support the functions `geof:distance`, `geof:buffer`, `geof:intersection`, `geof:union`, `geof:isEmpty`, `geof:isSimple`, `geof:area`, `geof:length`, `geof:numGeometries`, `geof:geometryN`, `geof:transform`, `geof:dimension`, `geof:difference`, `geof:symDifference`, `geof:envelope` and `geof:boundary` as SPARQL extension functions, consistent with the definitions of their corresponding functions in Simple Features [ISO19125-1] (`distance`, `buffer`, `intersection`, `union`, `isEmpty`, `isSimple`, `area`, `length`, `numGeometries`, `geometryN`, `transform`, `dimension`, `difference`, `symDifference`, `envelope` and `boundary` respectively) and other attached definitions and also `geof:maxX`, `geof:maxY`, `geof:maxZ`, `geof:minX`, `geof:minY` and `geof:minZ` SPARQL extension functions.

a. **Test purpose**: Check conformance with this requirement

b. **Test method**: Verify that a set of SPARQL queries involving each of the following functions returns the correct result for a test dataset when using the specified serialization and version: `geof:distance`, `geof:buffer`, `geof:intersection`, `geof:union`, `geof:isEmpty`, `geof:isSimple`, `geof:area`, `geof:length`, `geof:numGeometries`, `geof:geometryN`, `geof:transform`, `geof:dimension`, `geof:difference`, `geof:symDifference`, `geof:envelope` and `geof:boundary`.

c. **Reference**: Section 8.9

d. **Test Type**: Capabilities

A.3.1.5 `/conf/geometry-extension/srid-function`

Requirement: `/req/geometry-extension/srid-function`

Implementations shall support get SRID as a SPARQL extension function.

a. **Test purpose**: Check conformance with this requirement

b. **Test method**: Verify that a SPARQL query involving the get SRID function returns the correct result for a test dataset when using the specified serialization and version.

c. **Reference**: Section 8.9.15

d. **Test Type**: Capabilities

A.3.1.5 /conf/geometry-extension/sa-functions

Requirement: /req/geometry-extension/sa-functions

Implementations shall support geof:boundingBox, geof:boundingCircle, geof:centroid, geof:concatLines, geof:concaveHull, geof:convexHull and geof:union2 as a SPARQL extension functions.

a. **Test purpose**: Check conformance with this requirement

b. **Test method**: Verify that queries involving these functions return the correct result for a test dataset.

c. **Reference**: [_function_safuncs]

d. **Test Type**: Capabilities

A.3.2 serialization = WKT

A.3.2.1 /conf/geometry-extension/wkt-literal

Requirement: /req/geometry-extension/wkt-literal

All RDFS Literals of type geo:wktLiteral shall consist of an optional IRI identifying the coordinate reference system and a required Well Known Text (WKT) description of a geometric value. Valid geo:wktLiteral instances are formed by either a WKT string as defined in [ISO13249] or by concatenating a valid absolute IRI, as defined in [IETF3987], enclose in angled brackets (< & >) followed by a single space (Unicode U+0020 character) as a separator, and a WKT string as defined in [ISO13249].

a. **Test purpose**: Check conformance with this requirement

b. **Test method**: Verify that queries involving WKT Literal values return the correct result for a test dataset.

c. **Reference**: Section 8.8.1.1

d. **Test Type**: Capabilities

A.3.2.2 /conf/geometry-extension/wkt-literal-default-srs

Requirement: /req/geometry-extension/wkt-literal-default-srs

The IRI <http://www.opengis.net/def/crs/OGC/1.3/CRS84> shall be assumed as the spatial reference system for geo:wktLiteral instances that do not specify an explicit spatial reference system IRI.

a. **Test purpose**: Check conformance with this requirement

b. **Test method**: Verify that queries involving WKT Literal values without an explicit encoded SRS IRI return the correct result for a test dataset.

c. **Reference**: Section 8.8.1.1

d. **Test Type**: Capabilities

A.3.2.3 `/conf/geometry-extension/wkt-axis-order`

Requirement: `/req/geometry-extension/wkt-axis-order`

Coordinate tuples within WKT Literal instances shall be interpreted using the axis order defined in the SRS used.

a. **Test purpose**: Check conformance with this requirement

b. **Test method**: Verify that queries involving WKT Literal values return the correct result for a test dataset.

c. **Reference**: Section 8.8.1.1

d. **Test Type**: Capabilities

A.3.2.4 `/conf/geometry-extension/wkt-literal-empty`

Requirement: `/req/geometry-extension/wkt-literal-empty`

An empty RDFS Literal of type WKT Literal shall be interpreted as an empty geometry.

a. **Test purpose**: Check conformance with this requirement

b. **Test method**: Verify that queries involving empty WKT Literal values return the correct result for a test dataset.

c. **Reference**: Section 8.8.1.1

d. **Test Type**: Capabilities

A.3.2.5 `/conf/geometry-extension/geometry-as-wkt-literal`

Requirement: `/req/geometry-extension/geometry-as-wkt-literal`

Implementations shall allow the RDF property `geo:asWKT` to be used in SPARQL graph patterns.

a. **Test purpose**: Check conformance with this requirement

b. **Test method**: Verify that queries involving the `geo:asWKT` property return the correct result for a test dataset.

c. **Reference**: Section 8.8.1.2

d. **Test Type**: Capabilities

A.3.2.6 `/req/geometry-extension/asWKT-function`

Requirement: `/req/geometry-extension/asWKT-function`

Implementations shall support `geof:asWKT`, as a SPARQL extension function

a. **Test purpose**: Check conformance with this requirement

b. **Test method**: Verify that a set of SPARQL queries involving the `geof:asWKT` function returns the correct result for a test dataset when using the specified serialization and version.

c. **Reference**: [_function_aswkt]

d. **Test Type**: Capabilities

A.3.3 serialization = GML

A.3.3.1 `/conf/geometry-extension/gml-literal`

Requirement: `/req/geometry-extension/gml-literal`

All `geo:gmlLiteral` instances shall consist of a valid element from the GML schema that implements a subtype of GM_Object as defined in [OGC 07-036].

a. **Test purpose**: Check conformance with this requirement

b. **Test method**: Verify that queries involving `geo:gmlLiteral` values return the correct result for a test dataset.

c. **Reference**: Section 8.8.2.1

d. **Test Type**: Capabilities

A.3.3.2 `/conf/geometry-extension/gml-literal-empty`

Requirement: `/req/geometry-extension/gml-literal-empty`

An empty `geo:gmlLiteral` shall be interpreted as an empty geometry.

a. **Test purpose**: Check conformance with this requirement

b. **Test method**: Verify that queries involving empty `geo:gmlLiteral` values return the correct result for a test dataset.

c. **Reference**: Section 8.8.2.1

d. **Test Type**: Capabilities

A.3.3.3 `/conf/geometry-extension/gml-profile`

Requirement: `/req/geometry-extension/gml-profile`

Implementations shall document supported GML profiles.

a. **Test purpose**: Check conformance with this requirement

b. **Test method**: Examine the implementation’s documentation to verify that the supported GML profiles are documented.

c. **Reference**: Section 8.8.2.1

d. **Test Type**: Documentation

A.3.3.4 `/conf/geometry-extension/geometry-as-gml-literal`

Requirement: `/req/geometry-extension/geometry-as-gml-literal`

Implementations shall allow the RDF property `geo:asGML` to be used in SPARQL graph patterns.

a. **Test purpose**: Check conformance with this requirement
b. **Test method**: Verify that queries involving the `geo:asGML` property return the correct result for a test dataset.
c. **Reference**: Section 8.8.2.2
d. **Test Type**: Capabilities

A.3.3.5 `/req/geometry-extension/asGML-function`

Requirement: `/req/geometry-extension/asGML-function`

Implementations shall support `geof:asGML`, as a SPARQL extension function

a. **Test purpose**: Check conformance with this requirement
b. **Test method**: Verify that a set of SPARQL queries involving the `geof:asGML` function returns the correct result for a test dataset when using the specified serialization and version.
c. **Reference**: [_function_asgml]
d. **Test Type**: Capabilities

A.3.4 serialization = GEOJSON

A.3.4.1 `/req/geometry-extension/geojson-literal`

Requirement: `/req/geometry-extension/geojson-literal`

All `geo:geoJSONLiteral` instances shall consist of valid JSON that conforms to the GeoJSON specification [GEOJSON]

a. **Test purpose**: Check conformance with this requirement
b. **Test method**: Verify that queries involving `geo:geoJSONLiteral` values return the correct result for a test dataset.
c. **Reference**: Section 8.8.2.2
d. **Test Type**: Capabilities

A.3.4.2 `/req/geometry-extension/geojson-literal-srs`

Requirement: `/req/geometry-extension/geojson-literal-default-srs`

The IRI <http://www.opengis.net/def/crs/OGC/1.3/CRS84> shall be assumed as the SRS for `geo:geoJSONLiteral` instances that do not specify an explicit SRS IRI.

a. **Test purpose**: Check conformance with this requirement

b. **Test method**: Verify that queries involving `geo:geoJSONLiteral` values without an explicit encoded SRS IRI return the correct result for a test dataset.

c. **Reference**: Section 8.8.3.1

d. **Test Type**: Capabilities

A.3.4.3 `/req/geometry-extension/geojson-literal-empty`

Requirement: `/req/geometry-extension/geojson-literal-empty`

An empty `geo:geoJSONLiteral` shall be interpreted as an empty geometry.

a. **Test purpose**: Check conformance with this requirement

b. **Test method**: Verify that queries involving empty `geo:geoJSONLiteral` values return the correct result for a test dataset.

c. **Reference**: Section 8.8.3.1

d. **Test Type**: Capabilities

A.3.4.4 `/req/geometry-extension/geometry-as-geojson-literal`

Requirement: `/req/geometry-extension/geometry-as-geojson-literal`

Implementations shall allow the RDF property `geo:asGeoJSON` to be used in SPARQL graph patterns.

a. **Test purpose**: Check conformance with this requirement

b. **Test method**: Verify that queries involving the `geo:asGeoJSON` property return the correct result for a test dataset.

c. **Reference**: Section 8.8.3.2

d. **Test Type**: Capabilities

A.3.4.5 `/req/geometry-extension/asGeoJSON-function`

Requirement: `/req/geometry-extension/asGeoJSON-function`

Implementations shall support `geof:asGeoJSON`, as a SPARQL extension function

a. **Test purpose**: Check conformance with this requirement

b. **Test method**: Verify that a set of SPARQL queries involving the `geof:asGeoJSON` function returns the correct result for a test dataset when using the specified serialization and version.

c. **Reference**: [_function_asgeojson]

d. **Test Type**: Capabilities

A.3.5 serialization = KML

A.3.5.1 /req/geometry-extension/kml-literal

Requirement: /req/geometry-extension/kml-literal

All geo:kmlLiteral instances shall consist of a valid element from the KML schema that implements a kml:AbstractObjectGroup as defined in [OGCKML].

a. **Test purpose**: Check conformance with this requirement

b. **Test method**: Verify that queries involving geo:kmlLiteral values return the correct result for a test dataset.

c. **Reference**: [_rdfs_datatype_geomklliteral]

d. **Test Type**: Capabilities

A.3.5.2 /req/geometry-extension/kml-literal-srs

Requirement: /req/geometry-extension/kml-literal-default-srs

The IRI <http://www.opengis.net/def/crs/OGC/1.3/CRS84> shall be assumed as the SRS for geo:kmlLiteral instances that do not specify an explicit SRS IRI.

a. **Test purpose**: Check conformance with this requirement

b. **Test method**: Verify that queries involving geo:kmlLiteral values without an explicit encoded SRS IRI return the correct result for a test dataset.

c. **Reference**: [_rdfs_datatype_geomklliteral]

d. **Test Type**: Capabilities

A.3.5.3 /req/geometry-extension/kml-literal-empty

Requirement: /req/geometry-extension/kml-literal-empty

An empty geo:kmlLiteral shall be interpreted as an empty geometry.

a. **Test purpose**: Check conformance with this requirement

b. **Test method**: Verify that queries involving empty geo:kmlLiteral values return the correct result for a test dataset.

c. **Reference**: [_rdfs_datatype_geomklliteral]

d. **Test Type**: Capabilities

A.3.5.4 /req/geometry-extension/geometry-as-kml-literal

Requirement: /req/geometry-extension/geometry-as-kml-literal

Implementations shall allow the RDF property geo:asKML to be used in SPARQL graph patterns.

a. **Test purpose**: Check conformance with this requirement

b. **Test method**: Verify that queries involving the `geo:asKML` property return the correct result for a test dataset.

c. **Reference**: Section 8.8.4.2

d. **Test Type**: Capabilities

A.3.5.5 `/req/geometry-extension/asKML-function`

Requirement: `/req/geometry-extension/asKML-function`

Implementations shall support as KML, as a SPARQL extension function

a. **Test purpose**: Check conformance with this requirement

b. **Test method**: Verify that a set of SPARQL queries involving the `geof:asKML` function returns the correct result for a test dataset when using the specified serialization and version.

c. **Reference**: [_function_askml]

d. **Test Type**: Capabilities

A.3.6 serialization = DGGS

A.3.6.1 `/req/geometry-extension/dggs-literal`

Requirement: `/req/geometry-extension/dggs-literal`

All RDFS Literals of type `geo:dggsLiteral` shall consist of a DGGS geometry serialization formulated according to a specific DGGS literal type identified by a datatype specializing this generic datatype.

a. **Test purpose**: Check conformance with this requirement

b. **Test method**: Verify that queries do not use use this datatype but instead use specializations of it.

c. **Reference**: [_dggs_serialization_serializationdggs]

d. **Test Type**: Capabilities

A.3.6.2 `/req/geometry-extension/dggs-literal-empty`

Requirement: `/req/geometry-extension/dggs-literal-empty`

An empty `geo:dggsLiteral` shall be interpreted as an empty geometry.

a. **Test purpose**: Check conformance with this requirement

b. **Test method**: Verify that queries involving empty `geo:dggsLiteral` values return the correct result for a test dataset.

c. **Reference**: [_dggs_serialization_serializationdggs]

d. **Test Type**: Capabilities

A.3.6.3 `/req/geometry-extension/geometry-as-dggs-literal`

Requirement: `/req/geometry-extension/geometry-as-dggs-literal`

Implementations shall allow the RDF property `geo:asDGGS` to be used in SPARQL graph patterns.

a. **Test purpose**: Check conformance with this requirement

b. **Test method**: Verify that queries involving the `geo:asDGGS` property return the correct result for a test dataset.

c. **Reference**: Section 8.8.5.2

d. **Test Type**: Capabilities

A.3.6.4 `/req/geometry-extension/asDGGS-function`

Requirement: `/req/geometry-extension/asDGGS-function`

Implementations shall support `geof:asDGGS`, as a SPARQL extension function

a. **Test purpose**: Check conformance with this requirement

b. **Test method**: Verify that a set of SPARQL queries involving the `geof:asDGGS` function returns the correct result for a test dataset when using the specified serialization and version.

c. **Reference**: [_function_asdggs]

d. **Test Type**: Capabilities

A.4 Conformance Class: Geometry Topology Extension (relation_family, serialization, version)

Conformance Class IRI: `/conf/geometry-topology-extension`

A.4.1 Tests for all relation families

A.4.1.1 `/conf/geometry-topology-extension/relate-query-function`

Requirement: `/req/geometry-topology-extension/relate-query-function`

Implementations shall support `geof:relate` as a SPARQL extension function, consistent with the relate operator defined in Simple Features [ISO19125-1].

a. **Test purpose**: Check conformance with this requirement
b. **Test method**: Verify that a set of SPARQL queries involving the `geof:relate` function returns the correct result for a test dataset when using the specified serialization and version.
c. **Reference**: Section 9.2
d. **Test Type**: Capabilities

A.4.2 relation_family = Simple Features

A.4.2.1 `/conf/geometry-topology-extension/sf-query-functions`

Requirement: `/req/geometry-topology-extension/sf-query-functions`

Implementations shall support `geof:sfEquals`, `geof:sfDisjoint`, `geof:sfIntersects`, `geof:sfTouches`, `geof:sfCrosses`, `geof:sfWithin`, `geof:sfContains` and `geof:sfOverlaps` as SPARQL extension functions, consistent with their corresponding DE-9IM intersection patterns, as defined by Simple Features [ISO19125-1].

a. **Test purpose**: Check conformance with this requirement
b. **Test method**: Verify that a set of SPARQL queries involving each of the following functions returns the correct result for a test dataset when using the specified serialization and version: `geof:sfEquals`, `geof:sfDisjoint`, `geof:sfIntersects`, `geof:sfTouches`, `geof:sfCrosses`, `geof:sfWithin`, `geof:sfContains`, `geof:sfOverlaps` .
c. **Reference**: Section 7.2
d. **Test Type**: Capabilities

A.4.3 relation_family = Egenhofer

A.4.3.1 `/conf/geometry-topology-extension/eh-query-functions`

Requirement: `/req/geometry-topology-extension/eh-query-functions`

Implementations shall support `geof:ehEquals`, `geof:ehDisjoint`, `geof:ehMeet`, `geof:ehOverlap`, `geof:ehCovers`, `geof:ehCoveredBy`, `geof:ehInside` and `geof:ehContains` as SPARQL extension functions, consistent with their corresponding DE-9IM intersection patterns, as defined by Simple Features [ISO19125-1].

a. **Test purpose**: Check conformance with this requirement

b. **Test method**: Verify that a set of SPARQL queries involving each of the following functions returns the correct result for a test dataset when using the specified serialization and version: `geof:ehEquals`, `geof:ehDisjoint`, `geof:ehMeet`, `geof:ehOverlap`, `geof:ehCovers`, `geof:ehCoveredBy`, `geof:ehInside`, `geof:ehContains`.

c. **Reference**: Section 7.3

d. **Test Type**: Capabilities

A.4.4 relation_family = RCC8

A.4.4.1 `/conf/geometry-topology-extension/rcc8-query-functions`

Requirement: `/req/geometry-topology-extension/rcc8-query-functions`

Implementations shall support `geof:rcc8eq`, `geof:rcc8dc`, `geof:rcc8ec`, `geof:rcc8po`, `geof:rcc8tppi`, `geof:rcc8tpp`, `geof:rcc8ntpp` and `geof:rcc8ntppi` as SPARQL extension functions, consistent with their corresponding DE-9IM intersection patterns, as defined by Simple Features [ISO19125-1].

a. **Test purpose**: Check conformance with this requirement

b. **Test method**: Verify that a set of SPARQL queries involving each of the following functions returns the correct result for a test dataset when using the specified serialization and version: `geof:rcc8eq`, `geof:rcc8dc`, `geof:rcc8ec`, `geof:rcc8po`, `geof:rcc8tppi`, `geof:rcc8tpp`, `geof:rcc8ntpp`, `geof:rcc8ntppi` .

c. **Reference**: Section 7.4

d. **Test Type**: Capabilities

A.5 Conformance Class: RDFS Entailment Extension (relation_family, serialization, version)

Conformance Class IRI: `/conf/rdfs-entailment-extension`

A.5.1 Tests for all implementations

A.5.1.1 `/conf/rdfsentailmentextension/bgp-rdfs-ent`

Requirement: `/req/rdfs-entailment-extension/bgp-rdfs-ent`

Basic graph pattern matching shall use the semantics defined by the RDFS Entailment Regime [SPARQLENT].

a. **Test purpose**: Check conformance with this requirement
b. **Test method**: Verify that a set of SPARQL queries involving entailed RDF triples returns the correct result for a test dataset using the specified serialization, version and relation_family.
c. **Reference**: Section 10.2
d. **Test Type**: Capabilities

A.5.2 serialization = WKT

A.5.2.1 `/conf/rdfs-entailment-extension/wkt-geometry-types`

Requirement: `/req/rdfs-entailment-extension/wkt-geometry-types`

Implementations shall support graph patterns involving terms from an RDFS/OWL class hierarchy of geometry types consistent with the one in the specified version of Simple Features [ISO19125-1].

a. **Test purpose**: Check conformance with this requirement
b. **Test method**: Verify that a set of SPARQL queries involving WKT Geometry types returns the correct result for a test dataset using the specified version of Simple Features.
c. **Reference**: Section 10.3.1
d. **Test Type**: Capabilities

A.5.3 serialization = GML

A.5.3.1 `/conf/rdfs-entailment-extension/gml-geometry-types`

Requirement: `/req/rdfs-entailment-extension/gml-geometry-types`

Implementations shall support graph patterns involving terms from an RDFS/OWL class hierarchy of geometry types consistent with the GML schema that implements GM_Object using the specified version of GML [OGC07-036].

a. **Test purpose**: Check conformance with this requirement
b. **Test method**: Verify that a set of SPARQL queries involving GML Geometry types returns the correct result for a test dataset using the specified version of GML.
c. **Reference**: Section 10.4.1
d. **Test Type**: Capabilities

A.6 Conformance Class: Query Rewrite Extension (relation_family, serialization, version)

Conformance Class IRI: `/conf/query-rewrite-extension`

A.6.1 relation_family = Simple Features

A.6.1.1 `/conf/query-rewrite-extension/sf-query-rewrite`

Requirement: `/req/query-rewrite-extension/sf-query-rewrite`

Basic graph pattern matching shall use the semantics defined by the RIF Core Entailment Regime [SPARQLENT] for the RIF rules [RIFCORE] `geor:sfEquals`, `geor:sfDisjoint`, `geor:sfIntersects`, `geor:sfTouches`, `geor:sfCrosses`, `geor:sfWithin`, `geor:sfContains` and `geor:sfOverlaps`.

a. **Test purpose**: Check conformance with this requirement

b. **Test method**: Verify that queries involving the following query transformation rules return the correct result for a test dataset when using the specified serialization and version: `geor:sfEquals`, `geor:sfDisjoint`, `geor:sfIntersects`, `geor:sfTouches`, `geor:sfCrosses`, `geor:sfWithin`, `geor:sfContains` and `geor:sfOverlaps`.

c. **Reference**: Section 9.3

d. **Test Type**: Capabilities

A.6.2 relation_family = Egenhofer

A.6.2.1 `/conf/query-rewrite-extension/eh-query-rewrite`

Requirement: `/req/query-rewrite-extension/eh-query-rewrite`

Basic graph pattern matching shall use the semantics defined by the RIF Core Entailment Regime [SPARQLENT] for the RIF rules [RIFCORE] `geor:ehEquals`, `geor:ehDisjoint`, `geor:ehMeet`, `geor:ehOverlap`, `geor:ehCovers`, `geor:ehCoveredBy`, `geor:ehInside` and `geor:ehContains`.

a. **Test purpose**: Check conformance with this requirement

b. **Test method**: Verify that queries involving the following query transformation rules return the correct result for a test dataset when using the specified serialization and version: `geor:ehEquals`, `geor:ehDisjoint`, `geor:ehMeet`, `geor:ehOverlap`, `geor:ehCovers`, `geor:ehCoveredBy`, `geor:ehInside`, `geor:ehContains`.

c. **Reference**: Section 9.4

d. **Test Type**: Capabilities

A.6.3 relation_family = RCC8

A.6.3.1 `/conf/query-rewrite-extension/rcc8-query-rewrite`

Requirement: `/req/query-rewrite-extension/rcc8-query-rewrite`

Basic graph pattern matching shall use the semantics defined by the RIF Core Entailment Regime [SPARQLENT] for the RIF rules [RIFCORE] `geor:rcc8eq`, `geor:rcc8dc`, `geor:rcc8ec`, `geor:rcc8po`, `geor:rcc8tppi`, `geor:rcc8tpp`, `geor:rcc8ntpp` and `geor:rcc8ntppi`.

a. **Test purpose**: Check conformance with this requirement

b. **Test method**: Verify that queries involving the following query transformation rules return the correct result for a test dataset when using the specified serialization and version: `geor:rcc8eq`, `geor:rcc8dc`, `geor:rcc8ec`, `geor:rcc8po`, `geor:rcc8tppi`, `geor:rcc8tpp`, `geor:rcc8ntpp`, `geor:rcc8ntppi`.

c. **Reference**: Section 11.4

d. **Test Type**: Capabilities

Annex B - Functions Summary (normative)

B.0 Overview

This annex summarises all the functions defined in GeoSPARQL, providing descriptions of their parameter requirements and return types.

B.1 Functions Summary Table

Function Name	Input Parameters	Valid Input Types	Output Datatype	Output Datatype Subtypes	2D	3D
Simple Features Functions						
sfContains	2x ogc:geomLiteral	1x Polygon, 1x Geometry	xsd:boolean		Yes	Yes
sfCrosses	2x ogc:geomLiteral	2x LineString	xsd:boolean		Yes	Yes
sfDisjoint	2x ogc:geomLiteral	2x Geometry	xsd:boolean		Yes	Yes
sfEquals	2x ogc:geomLiteral	2x Geometry	xsd:boolean		Yes	Yes
sfIntersects	2x ogc:geomLiteral	2x Polygon	xsd:boolean		Yes	Yes
sfOverlaps	2x ogc:geomLiteral	2x Polygon	xsd:boolean		Yes	Yes
sfTouches	2x ogc:geomLiteral	2x Geometry	xsd:boolean		Yes	Yes
sfWithin	2x ogc:geomLiteral	1x Geometry, 1x Polygon	xsd:boolean		Yes	Yes
Egenhofer Functions						
ehContains	2x ogc:geomLiteral	1x Polygon, 1x Geometry	xsd:boolean		Yes	Yes
ehCoveredBy	2x ogc:geomLiteral	1x Polygon, 1x Geometry	xsd:boolean		Yes	Yes
ehCovers	2x ogc:geomLiteral	1x Polygon, 1x Geometry	xsd:boolean		Yes	Yes

Function Name	Input Parameters	Valid Input Types	Output Datatype	Output Datatype Subtypes	2D	3D
ehDisjoint	2x ogc:geomLiteral	2x Geometry	xsd:boolean		Yes	Yes
ehEquals	2x ogc:geomLiteral	2x Geometry	xsd:boolean		Yes	Yes
ehMeet	2x ogc:geomLiteral		xsd:boolean		Yes	Yes
ehOverlap	2x ogc:geomLiteral		xsd:boolean		Yes	Yes
ehInside	2x ogc:geomLiteral		xsd:boolean		Yes	Yes
Region Connection Calculus Functions						
rcc8dcc	2x ogc:geomLiteral		xsd:boolean		Yes	Yes
rcc8ecc	2x ogc:geomLiteral		xsd:boolean		Yes	Yes
rcc8eq	2x ogc:geomLiteral		xsd:boolean		Yes	Yes
rcc8ntpp	2x ogc:geomLiteral		xsd:boolean		Yes	Yes
rcc8ntppi	2x ogc:geomLiteral		xsd:boolean		Yes	Yes
rcc8po	2x ogc:geomLiteral		xsd:boolean		Yes	Yes
rcc8tpp	2x ogc:geomLiteral		xsd:boolean		Yes	Yes

Function Name	Input Parameters	Valid Input Types	Output Datatype	Output Datatype Subtypes	2D	3D
rcc8tppi	2x ogc:geomLiteral		xsd:boolean		Yes	Yes
Spatial Aggregate Functions						
aggBoundingBox	1 or more ogc:geomLiteral		ogc:geomLiteral	square Polygon(not DGGS), CellList (DGGS)	Yes	Yes
aggBoundingCircle	1 or more ogc:geomLiteral		ogc:geomLiteral	Polygon(not DGGS) CellList(DGGS)	Yes	Yes
aggCentroid	1 or more ogc:geomLiteral		ogc:geomLiteral	Point(not DGGS), Cell(DGGS)	Yes	Yes
aggConcatLines	1 or more ogc:geomLiteral		ogc:geomLiteral	square LineString(not DGGS) OrderedCellList(DGGS)	Yes	Yes
aggConcaveHull	1 or more ogc:geomLiteral		ogc:geomLiteral	Polygon(not DGGS), CellList(DGGS)	Yes	Yes
aggConvexHull	1 or more ogc:geomLiteral		ogc:geomLiteral	Polygon(not DGGS), CellList(DGGS)	Yes	Yes
aggUnion	1 or more ogc:geomLiteral		ogc:geomLiteral	Polygon(not DGGS), CellList(DGGS)	Yes	Yes

Function Name	Input Parameters	Valid Input Types	Output Datatype	Output Datatype Subtypes	2D	3D
Non-topological Query Functions						
area	1x ogc:geomLiteral	Polygon	xsd:double	xsd:double	Yes	No
boundary	1x ogc:geomLiteral	Geometry	ogc:geomLiteral	LineString(not DGGS), OrderedCellList(DGGS)	Yes	Yes
buffer	1x ogc:geomLiteral, 1x xsd:double, 1x xsd:anyURI	any	ogc:geomLiteral	(Multi)Polygon(not DGGS), CellList(DGGS)	Yes	Yes
convexHull	1x ogc:geomLiteral	Geometry	ogc:geomLiteral	LineString(not DGGS)	Yes	Yes
coordinateDimension	1x ogc:geomLiteral	Geometry	xsd:integer		Yes	Yes
difference	2x ogc:geomLiteral	2x Geometry	ogc:geomLiteral	(Multi)Polygon(not DGGS), CellList(DGGS)	Yes	Yes
dimension	1x ogc:geomLiteral	Geometry	xsd:double	xsd:double	Yes	Yes
distance	2x ogc:geomLiteral, 1x xsd:anyURI	2x Geometry	xsd:double	xsd:double	Yes	Yes

Function Name	Input Parameters	Valid Input Types	Output Datatype	Output Datatype Subtypes	2D	3D
envelope	1x ogc:geomLiteral, 1x xsd:anyURI	Geometry	ogc:geomLiteral	(Multi)Polygon(not DGGS), CellList(DGGS)	Yes	Yes
geometryN	1x ogc:geomLiteral	GeometryCollection(not DGGS)	xsd:double	xsd:double	Yes	No
geometryType	1x ogc:geomLiteral	Geometry	xsd:string		Yes	Yes
getSRID	1x ogc:geomLiteral	Geometry	xsd:anyURI		Yes	Yes
intersection	2x ogc:geomLiteral	2x Geometry	ogc:geomLiteral	Polygon(not DGGS), CellList(DGGS)	Yes	Yes
is3D	1x ogc:geomLiteral	Geometry	xsd:boolean		Yes	Yes
isEmpty	1x ogc:geomLiteral	Geometry	xsd:boolean		Yes	Yes
isMeasured	1x ogc:geomLiteral	Geometry	xsd:boolean		Yes	Yes
isSimple	1x ogc:geomLiteral	Geometry	xsd:boolean		Yes	Yes
length	1x ogc:geomLiteral	Geometry	xsd:double	xsd:double	Yes	No
numGeometries	1x ogc:geomLiteral	Geometry(not DGGS)	xsd:double	xsd:double	Yes	No

Function Name	Input Parameters	Valid Input Types	Output Datatype	Output Datatype Subtypes	2D	3D
projectTo	1x ogc:geomLiteral	Geometry	geo:gmlLiteral		Yes	Yes
spatialDimension	1x ogc:geomLiteral	Geometry	xsd:integer		Yes	Yes
symDifference	2x ogc:geomLiteral	2x Geometry	ogc:geomLiteral	(Multi)Polygon(not DGGS), CellList(DGGS)	Yes	Yes
transform	1x ogc:geomLiteral, 1x xsd:anyURI	Geometry	ogc:geomLiteral	Geometry	Yes	No
union	2x ogc:geomLiteral	2x Geometry	ogc:geomLiteral	Polygon(not DGGS), CellList(DGGS)	Yes	Yes
Serialization Functions						
asDGGS	1x ogc:geomLiteral	Geometry	geo:dggsLiteral		Yes	Yes
asGeoJSON	1x ogc:geomLiteral	Geometry	geo:geoJSONLiteral		Yes	Yes
asGML	1x ogc:geomLiteral, 1x xsd:string	Geometry	geo:gmlLiteral		Yes	Yes
asKML	1x ogc:geomLiteral	Geometry	geo:kmlLiteral		Yes	Yes
asWKT	1x ogc:geomLiteral	Geometry	geo:wktLiteral		Yes	Yes
Extent Functions						

Function Name	Input Parameters	Valid Input Types	Output Datatype	Output Datatype Subtypes	2D	3D
getSRID	1x ogc:geomLiteral	Geometry	xsd:anyURI		Yes	Yes
maxX	1x ogc:geomLiteral	Geometry	xsd:double		Yes	Yes
maxY	1x ogc:geomLiteral	Geometry	xsd:double		Yes	Yes
maxZ	1x ogc:geomLiteral	Geometry	xsd:double		Yes	Yes
minX	1x ogc:geomLiteral	Geometry	xsd:double		Yes	Yes
minY	1x ogc:geomLiteral	Geometry	xsd:double		Yes	Yes
minZ	1x ogc:geomLiteral	Geometry	xsd:double		Yes	Yes
Other Functions						
relate	2x ogc:geomLiteral		xsd:string	xsd:string	Yes	Yes

B.2 GeoSPARQL to SFA Functions Mapping

The following table indicates which GeoSPARQL non-topological query functions map to Simple Features Access ([ISO19125-1]) functions and in which GeoSPARQL version the functions are defined.

Where the Simple Features Access function has the same name as the GeoSPARQL function, 'x' is recorded.

GeoSPARQL Function	in 1.0	in 1.1	SFA
area		x	
			asBinary
asWKT*	x	x	asText
boundary	x	x	x
buffer	x	x	x
convexHull	x	x	x
coordinateDimension		x	x
difference	x	x	x
dimension		x	x
distance	x	x	x
envelope	x	x	x
geometryN		x	
geometryType		x	x
getSRID	x	x	SRID
intersection	x	x	x
is3D			x
isEmpty		x	x
isMeasured		x	x
isSimple		x	x
length		x	
maxX		x	
maxY		x	
maxZ		x	
minX		x	
minY		x	
minZ		x	
numGeometries		x	

GeoSPARQL Function	in 1.0	in 1.1	SFA
projectTo		x	
spatialDimension		x	x
symDifference	x	x	x
transform		x	x
union	x	x	x

* GeoSPARQL's `asWKT` is only a partial implementation of `asText` since `asWKT` only returns WKT, not textual geometry literal data in general.

Annex C - GeoSPARQL Examples (informative)

C.0 Overview

This Annex provides examples of the GeoSPARQL ontology and functions. In addition to these, extended examples are provided seperately by the GeoSPARQL 1.1 profile, see the GeoSPARQL Standard structure for the link to those examples.

C.1 RDF Examples

This Section illustrates GeoSPARQL ontology modelling with extended examples.

C.1.1 Classes

C.1.1.1 `SpatialObject`

The `SpatialObject` class is defined in Section 6.2.1.

C.1.1.1.1 Basic use

Basic use (as per the example in the class definition)

```
eg:x
    a geo:SpatialObject ;
    skos:prefLabel "Object X";
.
```

NOTE It is unlikely that users of GeoSPARQL will create many instances of `geo:SpatialObject` as its two more concrete subclasses, `geo:Feature` & `geo:Geometry`, are more directly relatable to real-world phenomena and use.

C.1.1.1.2 Size Properties

The "size" properties - `geo:hasSize`, `geo:hasMetricSize`, `geo:hasLength`, `geo:hasMetricLength`, `geo:hasPerimeterLength`, `geo:hasMetricPerimeterLength`, `geo:hasArea`, `geo:hasMetricArea`, `geo:hasVolume` and `geo:hasMetricVolume` - are all applicable to instances of `geo:SpatialObject` although, as per the note in the section above, they are likely to be used with `geo:Feature` & `geo:Geometry` instances.

```
@prefix qudt: <http://qudt.org/schema/qudt/> .
@prefix unit: <http://qudt.org/vocab/unit/> .

eg:moreton-island
    a geo:SpatialObject ;

    skos:prefLabel "Moreton Island" ;
    rdfs:seeAlso "https://en.wikipedia.org/wiki/Moreton_Island"^^xsd:anyURI ;

    geo:hasPerimeterLength [
        qudt:numericValue "92.367"^^xsd:float ;
        qudt:unit unit:KiloM ;
    ];
    geo:hasMetricPerimeterLength "92367"^^xsd:double ;
.
```

Here a spatial object, Moreton Island, has the distance of its coastline given with two properties: `geo:hasPerimeterLength` & `geo:hasMetricPerimeterLength`. The object for the first is a Blank Node with a QUDT value property of 92.367 and a QUDT unit property of `unit:KiloM` (kilometre). The object for the second is the literal `92367` (a double) which is, by the property's definition, a number of metres.

The use of the *Quantities, Units, Dimensions and Types (QUDT)* ontology[13] and its `qudt:numericValue` & `qudt:unit` is just one of many possible ways to convey the value of `geo:hasPerimeterLength` and any subproperty of `geo:hasSize`.

C.1.1.2 `Feature`

The `Feature` class is defined in Section 6.2.2.

C.1.1.2.1 Basic use

```
eg:x
    a geo:Feature ;
    skos:prefLabel "Feature X" ;
.
```

Here a `Feature` is declared and given a preferred label.

C.1.1.2.2 A `Feature` related to a `Geometry`

```
eg:x
    a geo:Feature ;
    skos:prefLabel "Feature X" ;
    geo:hasGeometry [
        geo:asWKT "MULTIPOLYGON (((149.06016 -35.23610, 149.06062 -35.23604, ... ,
149.06016 -35.23610)))"^^geo:wktLiteral ;
    ] ;
.
```

Here a `geo:Feature` is declared, given a preferred label and a Geometry for that `geo:Feature` is indicated with the use of `geo:hasGeometry`. The `Geometry` indicated is described using a *Well-Known Text* literal value, indicated by the property `geo:asWKT` and the literal type `geo:wktLiteral`.

C.1.1.2.3 `Feature` with `Geometry` and size (area)

```
eg:x
    a geo:Feature ;
    skos:prefLabel "Feature X" ;
        geo:hasGeometry [
            geo:asWKT "MULTIPOLYGON (((149.06016 -35.23610, 149.06062 -35.23604, ... ,
149.06016 -35.23610)))"^^geo:wktLiteral ;
    ] ;
    geo:hasMetricArea "8.9E4"^^xsd:double ;
.
```

This example and the example below (B 1.1.2.4) show the same Section 6.2.2, but with a different specification of its area. This example shows the recommended way to express size: by using a subproperty of Section 6.3.2 (in this case, [Property: geo:MetricArea]). These subproperties have fixed units based on meter (the unit of distance in the International System of Units).

C.1.1.2.4 `Feature` with `Geometry` and non-metric size

```
@prefix qudt: <http://qudt.org/schema/qudt/> .

eg:x
    a geo:Feature ;
    skos:prefLabel "Feature X";
        geo:hasGeometry [
            geo:asWKT "MULTIPOLYGON (((149.06016 -35.23610, 149.06062 -35.23604, ... ,
149.06016 -35.23610)))"^^geo:wktLiteral ;
    ] ;
    geo:hasArea [
        qudt:numericValue "2.2E5"^^xsd:double ;
        qudt:unit <http://qudt.org/vocab/unit/AC> ;  # international acre
    ] ;
.
```

Here a Section 6.2.2 is described as per the previous example but its area is expressed in non-metric units: the acre.

C.1.1.2.5 `Feature` with two different `Geometry` instances indicated

```
eg:x
    a geo:Feature ;
    skos:prefLabel "Feature X";
    geo:hasGeometry [
        rdfs:label "Official boundary" ;
        rdfs:comment "Official boundary from the Department of Xxx" ;
        geo:asWKT "MULTIPOLYGON (((149.06016 -35.23610, 149.06062 -35.23604, ... ,
149.06016 -35.23610)))"^^geo:wktLiteral ;
    ] ,
    [
        rdfs:label "Unofficial boundary" ;
        rdfs:comment "Unofficial boundary as actually used by everyone" ;
        geo:asWKT "MULTIPOLYGON (((149.06016 -35.23610, 149.06062 -35.23604, ... ,
149.06016 -35.23610)))"^^geo:wktLiteral ;
    ] ;
.
```

In this example, `Feature X` has two different `Geometry` instances indicated with their different explained in annotation properties. No GeoSPARQL ontology properties are used to indicate a difference in these `Geometry` instances thus machine use of this `Feature` woud not be easily able to differentiate them.

C.1.1.2.6 `Feature` with two different `Geometry` instances with different property values

```
eg:x
    a geo:Feature ;
    skos:prefLabel "Feature X";
    geo:hasGeometry [
        geo:hasMetricSpatialResolution "100"^^xsd:double ;
        geo:asWKT "MULTIPOLYGON (((149.0601 -35.2361, 149.0606 -35.2360, ... ,
149.0601 -35.2361)))"^^geo:wktLiteral ;
    ] ,
    [
        geo:hasMetricSpatialResolution "5"^^xsd:double ;
        geo:asWKT "MULTIPOLYGON (((149.06016 -35.23610, 149.06062 -35.23604, ... ,
149.06016 -35.23610)))"^^geo:wktLiteral ;
    ] ;
.
```

In this example, `Feature X` has two different `Geometry` instances indicated with different spatial resolutions. Machine use of this `Feature` would be able to differentiate the two `Geometry` instances based on this use of <<.

C.1.1.2.7 `Feature` with non-metric size

```
@prefix dbp: <http://dbpedia.org/resource/> .
@prefix qudt: <http://qudt.org/schema/qudt/> .

ex:Seleucia_Artemita
    a geo:Feature ;
    skos:prefLabel "The route from Seleucia to Artemita"@en ;
    geo:hasLength [
      qudt:unit ex:Schoenus ;
      qudt:value "15"^^xsd:integer ;
    ]
.
ex:Schoenus
  a qudt:Unit;
  skos:exactMatch dbp:Schoenus;
.
```

In this example it is not possible to convert the length of the feature to meters, because the historical length unit does not have a known precise conversion factor.

C.1.1.2.8 `Feature` with two different types of `Geometry` instances

```
eg:x
    a geo:Feature ;
    skos:prefLabel "Feature X";
    geo:hasGeometry [
        geo:asWKT "POLYGON ((149.06016 -35.23610, 149.060620 -35.236043, ... ,
149.06016 -35.23610))"^^geo:wktLiteral ;
    ] ;
    geo:hasCentroid [
        geo:asWKT "POINT (149.06017784 -35.23612321)"^^geo:WktLiteral ;
    ] ;
.
```

Here a `Feature` instance has two geometries, one indicated with the general property `hasGeometry` and a second indicated with the specialised property `hasCentroid` which suggests the role that the indicated geometry plays. Note that while `hasGeometry` may indicate any type of `Geometry`, `hasCentroid` should only be used to indicate a point geometry. It may be informally inferred that the polygonal geometry is the `Feature` instance's boundary.

C.1.1.2.9 `Feature` with multiple sizes

```
ex:lake-x
    a geo:Feature ;
    skos:prefLabel "Lake X" ;
    eg:hasFeatureCategory <http://example.com/cat/lake> ;
    geo:hasMetricArea "9.26E4"^^xsd:double ;
    geo:hasMetricVolume "6E5"^^xsd:double ;
.
```

This example shows a `Feature` instance with area and volume declared. A categorization of the Feature is given through the use of the `eg:hasFeatureCategory` dummy property which, along with the Feature's preferred label, indicate that this Feature is a lake. Having both an area and a volume makes sense for a lake.

C.1.1.3 `Geometry`

The `Geometry` class is defined in Section 8.6.1.

C.1.1.3.1 Basic Use

```
eg:y a geo:Geometry ;
    skos:prefLabel "Geometry Y";
.
```

Here a `Geometry` is declared and given a preferred label.

From GeoSPARQL 1.0 use, the most commonly observed use of a `Geometry` is in relation to a `Feature` as per the example in [B 1.1.2.2 A `Feature` related to a `Geometry`] and often the `Geometry` is indirectly declared by the use of `hasGeometry` on the `Feature` instance indicating a Blank Node, however it is entirely possible to declare `Geometry` instances without any `Feature` instances. The next basic example declares a `Geometry` instance with an demonstration absolute URI and data.

```
<https://example.com/geometry/y>
    a geo:Geometry ;
    skos:prefLabel "Geometry Y";
    geo:asWKT "MULTIPOLYGON (((149.06016 -35.23610, 149.060620 -35.236043, ... ,
149.06016 -35.23610)))"^^geo:wktLiteral ;
.
```

Here the `Geometry` instance has data in WKT form and, since no CRS is declared, WGS84 is the assumed, default, CRS.

C.1.1.3.2 A `Geometry` with multiple serializations

```
eg:x
    a geo:Feature ;
    skos:prefLabel "Feature X";
    geo:hasGeometry [
        geo:asWKT "<http://www.opengis.net/def/crs/EPSG/0/4326> MULTIPOLYGON
(((149.06016 -35.23610, 149.060620 -35.236043, ... , 149.06016
-35.23610)))"^^geo:wktLiteral ;
        geo:asDGGS "CELLLIST ((R1234 R1235 R1236 ... R1256))"^^eg:auspixDggsLiteral ;
    ] ;
.
```

Here a single `Geometry`, linked to a `Feature` instance, is expressed using two different serializations: Well-known Text and the example AusPIX DGGS. Note that the latter is not formally defined in GeoSPARQL.

C.1.1.3.3 `Geometry` with scalar spatial property

```
eg:x
    a geo:Feature ;
    skos:prefLabel "Feature X";
    geo:hasGeometry eg:x-geo ;
.

eg:x-geo
    a geo:Geometry ;
    geo:asWKT "MULTIPOLYGON (((149.06016 -35.23610, 149.060620 -35.236043, ... ,
149.06016 -35.23610)))"^^geo:wktLiteral ;
    geo:hasMetricArea "8.7E4"^^xsd:double;
.
```

This example shows a Feature, `eg:x`, with a Geometry, `eg:x-geo`, which has both a serilization (WKT) indicated with the predicate `go:asWKT` and a scalar area indicated with the predicate `geo:hasMetricArea`. While it is entirely possible that scalar areas can be calculated from polygons, it may be efficient to store a pre-calculated scalar area in addition to the polygon. Perhaps the polygon is large and detailed and a one-time calculation with results stored is efficient for repeated use.

This use of a scalar spatial measurement property with a Geometry, here `geo:hasMetricArea`, is possible since the domain of such properties is `geo:SpatialObject`, the superclass of both `geo:Feature` and `geo:Geometry`.

C.1.1.4 `SpatialObjectCollection`

`geo:SpatialObjectCollection` isn't really intended to be implemented - it's essentially an abstract class - therefore no examples of its use are given. See the following two sections for examples of the concrete `geo:FeatureCollection` & `geo:GeometryCollection` classes.

C.1.1.5 `FeatureCollection`

This example shows a `FeatureCollection` instance containing 3 `Feature` instances.

```
ex:fc-x
    a geo:FeatureCollection ;
    dcterms:title "Feature Collection X" ;
    rdfs:member
        ex:feature-something ,
        ex:feature-other ,
        ex:feature-another ;
.
```

All of the GeoSPARQL collection classes are unordered since they are succlasses of the generic `rdfs:Container`, however implementers should consider that there are many ways to order the members of a `FeatureCollection` such as the `Feature` instances labels, their areas, geometries or any other property.

C.1.1.6 `GeometryCollection`

This example shows a `GeometryCollection` instance containing 3 `Geometry` instances.

```
ex:gc-x
    a geo:GeometryCollection ;
    dcterms:title "Geometry Collection X" ;
    rdfs:member
        ex:geometry-shape ,
        ex:geometry-othershape ,
        ex:geometry-anothershape ;
.
```

As per `FeatureCollection`, the `GeometryCollection` itse;f doesn't impose any ordering on its member `Geometry` instances, however there are many ways to order them, based on their own properties.

C.1.1.7 Simple Features classes

Most of the geometry seralizations used in GeoSPARQL define the geometry type - point, polygon etc. *within* the literal, e.g. WKT can encode `POLYGN(())` or 'POINT()', however the *Simple Features Vocabulary* resource within GeoSPARQL 1.1 contains specialised Geometry RDF classes such as `sf:Polygon`, `sf:PolyhedralSurface` and others.

It may be appropriate to use these specialised forms of Geometry in circumstances when geometry type differentiation is required within RDF and not withing specialised literal handling. This is the case when type differentiation must occur within plain SPARQL, not GeoSAPRQL.

The following example shows a `Feature` instance with two `Geometry` instances where the *Simple Features Vocabulary* classes are used to indicate the Geometry type:

```
ex:x
    a geo:Feature ;
    rdfs:label "Feature X" ;
    geo:hasGeometry [
        a sf:Point ;
        geo:asWKT "POINT(...)" ;
        rdfs:comment "A point geometry for Feature X, possibly a centroid though not
declared one" ;
    ] ;
    geo:hasGeometry [
        a sf:Polygon ;
        geo:asWKT "POLYGON((...))" ;
        rdfs:comment "A polygon geometry for Feature X" ;
    ] ;
```

There are several GeoSPARQL properties that suggest they could be used with particular *Simple Features Vocabulary* geometry types, for instance, `geo:hasCentroid` indicates is could be used with a `sf:Point` and `geo:hasBoundingBox` indicates use with an `sf:Envelope`.

C.1.2 Properties

C.1.2.1 Spatial Object Properties

See the section C.1.1.1.2 Size Properties above.

C.1.2.2 Feature Properties

This example shows a `geo:Feature` instance with each of the properties defined in Section 6.4 used, except for the properties `geo:hasMetricSize` and `geo:hasSize`, that are intended to be used through their subproperties and `geo:hasMetricPerimeterLength` and `geo:hasPerimeterLength` which are examplified in C.1.1.1.2 Size Properties.

```
@prefix qudt: <http://qudt.org/schema/qudt/> .

eg:x
    a geo:Feature ;
    skos:preferredLabel "Feature X" ;
    geo:hasGeometry [
        geo:asWKT "<http://www.opengis.net/def/crs/EPSG/0/4326> POLYGON ((149.06016
-35.23610, ... , 149.06016 -35.23610))"^^geo:wktLiteral ;
    ] ;
    geo:hasDefaultGeometry [
        geo:asWKT "<http://www.opengis.net/def/crs/EPSG/0/4326> POLYGON ((149.0601
-35.2361, ... , 149.0601 -35.2361))"^^geo:wktLiteral ;
    ] ;
    geo:hasMetricLength "355"^^xsd:double ;
    geo:hasLength [
        qudt:numericValue 355 ;
        qudt:unit <http://qudt.org/vocab/unit/M> ;  # meter
    ] ;
    geo:hasMetricArea "8.7E4"^^xsd:double ;
    geo:hasArea [
        qudt:numericValue 8.7 ;
        qudt:unit <http://qudt.org/vocab/unit/HA> ;  # hectare
    ] ;
    geo:hasMetricVolume "624432"^^xsd:double ;
    geo:hasVolume [
        qudt:numericValue 624432 ;
        qudt:unit <http://qudt.org/vocab/unit/M3> ;  # cubic meter
    ] ;
    geo:hasCentroid [
        geo:asWKT "POINT (149.06017 -35.23612)"^^geo:wktLiteral ;
    ] ;
    geo:hasBoundingBox [
        geo:asWKT "<http://www.opengis.net/def/crs/EPSG/0/4326> POLYGON ((149.060
-35.236, ... , 149.060 -35.236))"^^geo:wktLiteral ;
    ] ;
    geo:hasMetricSpatialResolution "5"^^xsd:double ;
    geo:hasSpatialResolution [
        qudt:numericValue 5 ;
        qudt:unit <http://qudt.org/vocab/unit/M> ;  # meter
    ] ;
.
```

The properties defined for this example’s `Feature` instance are vaguely aligned in that the values are not real but are not unrealistic either. It is outside the scope of GeoSPARQL to validate `Feature` instances' property values.

Note that this `Feature` has a 2D `Geometry` and yet a property indicating a scalar volume: `geo:hasVolume`. Used in this way, the scalar property is indicating information that cannot be calculated from other information about the `Feature` such as its geometry. Perhaps a volume for the feature has been esimated or measured in such a way that a 3D geometry was not created.

C.1.2.3 Geometry Properties

This example shows a `Geometry` instance, a Blank Node, declared in relation to a `Feature` instance, with each of the properties defined in Section 8.7 used.

```
@prefix qudt: <http://qudt.org/schema/qudt/> .
@prefix unit: <http://qudt.org/vocab/unit/> .

eg:x
    a geo:Feature ;
    geo:hasGeometry [
        skos:prefLabel "Geometry Y" ;
        geo:dimension 2 ;
        geo:coordinateDimension 2 ;
        geo:spatialDimension 2 ;
        geo:isEmpty false ;
        geo:isSimple true ;
        geo:hasSerialization "<http://www.opengis.net/def/crs/EPSG/0/4326> POLYGON
((149.060 -35.236, ... , 149.060 -35.236)))"^^geo:wktLiteral ;
        geo:hasSpatialAccuracy [
            qudt:numericValue "30"^^xsd:float ;
            qudt:unit unit:CentiM ;  # centimetres
        ] ;
        geo:hasMetricSpatialAccuracy "0.3"^^xsd:double ;
    ] ;
.
```

In this example, each of the standards properties defined for a `Geometry` instance has realistic values, for example, the is empty is set to `false` since the `Geometry` contains information.

C.1.2.4 Geometry Serializations

This section shows a `Geometry` instance for a `Feature` instance which is represented in all supported GeoSPARQL serlializations. The geometry values given are real geometry values and approximate Moreton Island in Queensland, Australia.

Note that the concrete DGGS serialization used is for example purposes only as it is not formally defined in GeoSPARQL.

```
eg:x
    a geo:Feature ;
    geo:hasGeometry [
        geo:asWKT """<http://www.opengis.net/def/crs/EPSG/0/4326>
            POLYGON ((
                153.3610112 -27.0621757,
                153.3658177 -27.1990606,
                153.421436 -27.3406573,
```

```
            153.4269292 -27.3607835,
            153.4434087 -27.3315078,
            153.4183848 -27.2913403,
            153.4189391 -27.2039578,
            153.4673476 -27.0267166,
            153.3610112 -27.0621757
        ))"""^^geo:wktLiteral ;

    geo:asGML """<gml:Polygon
            srsName="http://www.opengis.net/def/crs/EPSG/0/4326">
            <gml:exterior>
                <gml:LinearRing>
                    <gml:posList>
                        -27.0621757 153.3610112
                        -27.1990606 153.3658177
                        -27.3406573 153.421436
                        -27.3607835 153.4269292
                        -27.3315078 153.4434087
                        -27.2913403 153.4183848
                        -27.2039578 153.4189391
                        -27.0267166 153.4673476
                        -27.0621757 153.3610112
                    </gml:posList>
                </gml:LinearRing>
            </gml:exterior>
        </gml:Polygon>"""^^go:gmlLiteral ;

    geo:asKML """<Polygon>
            <outerBoundaryIs>
                <LinearRing>
                    <coordinates>
                    153.3610112,-27.0621757
                    153.3658177,-27.1990606
                    153.421436,-27.3406573
                    153.4269292,-27.3607835
                    153.4434087,-27.3315078
                    153.4183848,-27.2913403
                    153.4189391,-27.2039578
                    153.4673476,-27.0267166
                    153.3610112,-27.0621757
                    </coordinates>
                </LinearRing>
            </outerBoundaryIs>
        </Polygon>"""^^go:kmlLiteral ;

    geo:asGeoJSON """{
            "type": "Polygon",
            "coordinates": [[
                [153.3610112, -27.0621757],
                [153.3658177, -27.1990606],
                [153.421436, -27.3406573],
```

```
                [153.4269292, -27.3607835],
                [153.4434087, -27.3315078],
                [153.4183848, -27.2913403],
                [153.4189391, -27.2039578],
                [153.4673476, -27.0267166],
                [153.3610112, -27.0621757]
            ]]
        }"""^^geo:geoJSONLiteral ;

    geo:asDGGS """CELLLIST ((R8346031 R8346034 R8346037
        R83460058 R83460065 R83460068 R83460072 R83460073 R83460074 R83460075
R83460076
        R83460077 R83460078 R83460080 R83460081 R83460082 R83460083 R83460084
R83460085
        R83460086 R83460087 R83460088 R83460302 R83460305 R83460308 R83460320
R83460321
        R83460323 R83460324 R83460326 R83460327 R83460332 R83460335 R83460338
R83460350
        R83460353 R83460356 R83460362 R83460365 R83460380 R83460610 R83460611
R83460612
        R83460613 R83460614 R83460615 R83460617 R83460618 R83460641 R83460642
R83460644
        R83460645 R83460648 R83460672 R83460686 R83463020 R83463021 R834600487
R834600488
        R834600557 R834600558 R834600564 R834600565 R834600566 R834600567
R834600568
        R834600571 R834600572 R834600573 R834600574 R834600575 R834600576
R834600577
        R834600578 R834600628 R834600705 R834600706 R834600707 R834600708
R834600712
        R834600713 R834600714 R834600715 R834600716 R834600717 R834600718
R834601334
        R834601335 R834601336 R834601337 R834601338 R834601360 R834601361
R834601363
        R834601364 R834601366 R834601367 R834601600 R834601601 R834601603
R834601606
        R834601630 R834601633 R834603220 R834603221 R834603223 R834603224
R834603226
        R834603227 R834603250 R834603251 R834603253 R834603256 R834603280
R834603283
        R834603510 R834603511 R834603512 R834603513 R834603514 R834603515
R834603516
        R834603517 R834603540 R834603541 R834603543 R834603544 R834603546
R834603547
        R834603570 R834603573 R834603576 R834603681 R834603682 R834603684
R834603685
        R834603687 R834603688 R834603810 R834603830 R834603831 R834603832
R834603833
        R834603834 R834603835 R834603836 R834603837 R834603860 R834603861
R834603863
        R834603864 R834603866 R834603867 R834606021 R834606022 R834606024
```

```
R834606025
             R834606028 R834606052 R834606055 R834606160 R834606161 R834606162
R834606164
             R834606165 R834606167 R834606168 R834606200 R834606203 R834606206
R834606230
             R834606233 R834606236 R834606260 R834606263 R834606266 R834606401
R834606402
             R834606405 R834606408 R834606432 R834606471 R834606472 R834606474
R834606475
             R834606477 R834606478 R834606500 R834606503 R834606506 R834606530
R834606533
             R834606536 R834606560 R834606563 R834606566 R834606712 R834606715
R834606718
             R834606750 R834606751 R834606752 R834606753 R834606754 R834606755
R834606757
             R834606758 R834606781 R834606782 R834606784 R834606785 R834606788
R834606800
             R834606803 R834606806 R834606807 R834606830 R834606831 R834606833
R834606834
             R834606835 R834606836 R834606837 R834606838 R834606870 R834606873
R834606874
             R834606876 R834606877 R834630122 R834630125 R834630226 R834630230
R834630231
             R834630232 R834630234 R834630235 R834630237 R834630238 R834630240
R834630241
             R834630242 R834630243 R834630244 R834630245 R834630246 R834630247
R834630261
             R834630262 R834630264 R834630265 R834630268 R834630270 R834630271
R834630273
             R834630276 R834630502))"""^^eg:auspixDggsLiteral ;
    ] ;
.
```

[13] http://www.qudt.org

C.2 Example SPARQL Queries & Rules

This Section provides example data and then illustrates the use of GeoSPARQL functions and the application of rules with that data.

C.2.1 Example Data

The following RDF data (Turtle format) encodes application-specific spatial data. The resulting spatial data is illustrated in the figure below. The RDF statements define the feature class `my:PlaceOfInterest`, and two properties are created for associating geometries with features: `my:hasExactGeometry` and `my:hasPointGeometry`. `my:hasExactGeometry` is designated as the default geometry for the `my:PlaceOfInterest` feature class.

All the following examples use the parameter values relation_family = Simple Features, serialization = WKT, and version = 1.0.

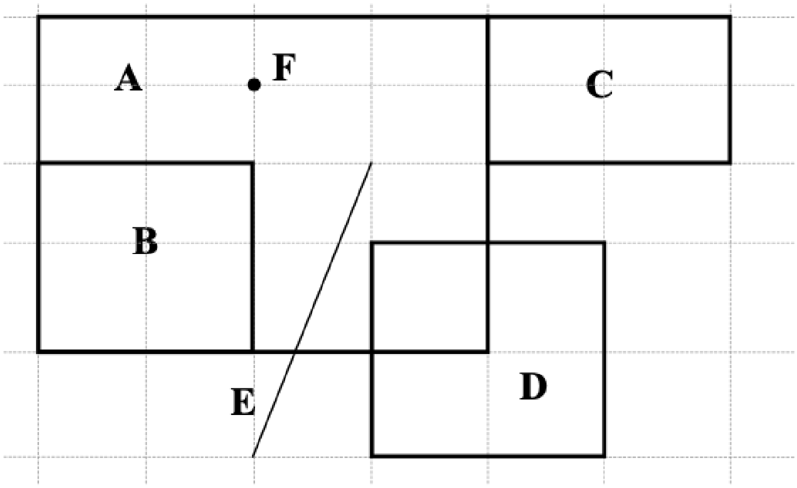

Figure 4. Illustration of spatial data

```
@prefix geo: <http://www.opengis.net/ont/geosparql#> .
@prefix my: <http://example.org/ApplicationSchema#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix sf: <http://www.opengis.net/ont/sf#> .

my:PlaceOfInterest a rdfs:Class ;
    rdfs:subClassOf geo:Feature .

my:A a my:PlaceOfInterest ;
    my:hasExactGeometry my:AExactGeom ;
    my:hasPointGeometry my:APointGeom .

my:B a my:PlaceOfInterest ;
    my:hasExactGeometry my:BExactGeom ;
    my:hasPointGeometry my:BPointGeom .

my:C a my:PlaceOfInterest ;
```

```
    my:hasExactGeometry my:CExactGeom ;
    my:hasPointGeometry my:CPointGeom .

my:D a my:PlaceOfInterest ;
    my:hasExactGeometry my:DExactGeom ;
    my:hasPointGeometry my:DPointGeom .

my:E a my:PlaceOfInterest ;
    my:hasExactGeometry my:EExactGeom .

my:F a my:PlaceOfInterest ;
    my:hasExactGeometry my:FExactGeom .

my:hasExactGeometry a rdf:Property ;
    rdfs:subPropertyOf geo:hasDefaultGeometry,
        geo:hasGeometry .

my:hasPointGeometry a rdf:Property ;
    rdfs:subPropertyOf geo:hasGeometry .

my:AExactGeom a sf:Polygon ;
    geo:asWKT """<http://www.opengis.net/def/crs/OGC/1.3/CRS84>
             Polygon((-83.6 34.1, -83.2 34.1, -83.2 34.5,
             -83.6 34.5, -83.6 34.1))"""^^geo:wktLiteral.

my:APointGeom a sf:Point ;
    geo:asWKT """<http://www.opengis.net/def/crs/OGC/1.3/CRS84>
             Point(-83.4 34.3)"""^^geo:wktLiteral.

my:BExactGeom a sf:Polygon ;
    geo:asWKT """<http://www.opengis.net/def/crs/OGC/1.3/CRS84>
             Polygon((-83.6 34.1, -83.4 34.1, -83.4 34.3,
             -83.6 34.3, -83.6 34.1))"""^^geo:wktLiteral.

my:BPointGeom a sf:Point ;
    geo:asWKT """<http://www.opengis.net/def/crs/OGC/1.3/CRS84>
             Point(-83.5 34.2)"""^^geo:wktLiteral.

my:CExactGeom a sf:Polygon ;
    geo:asWKT """<http://www.opengis.net/def/crs/OGC/1.3/CRS84>
             Polygon((-83.2 34.3, -83.0 34.3, -83.0 34.5,
             -83.2 34.5, -83.2 34.3))"""^^geo:wktLiteral.

my:CPointGeom a sf:Point ;
    geo:asWKT """<http://www.opengis.net/def/crs/OGC/1.3/CRS84>
             Point(-83.1 34.4)"""^^geo:wktLiteral.

my:DExactGeom a sf:Polygon ;
    geo:asWKT """<http://www.opengis.net/def/crs/OGC/1.3/CRS84>
             Polygon((-83.3 34.0, -83.1 34.0, -83.1 34.2,
             -83.3 34.2, -83.3 34.0))"""^^geo:wktLiteral.
```

```
my:DPointGeom a sf:Point ;
    geo:asWKT """<http://www.opengis.net/def/crs/OGC/1.3/CRS84>
                 Point(-83.2 34.1)"""^^geo:wktLiteral.

my:EExactGeom a sf:LineString ;
    geo:asWKT """<http://www.opengis.net/def/crs/OGC/1.3/CRS84>
                 LineString(-83.4 34.0, -83.3 34.3)"""^^geo:wktLiteral.

my:FExactGeom a sf:Point ;
    geo:asWKT """<http://www.opengis.net/def/crs/OGC/1.3/CRS84>
                 Point(-83.4 34.4)"""^^geo:wktLiteral.
```

C.2.2 Example Queries

This Section illustrates the use of GeoSPARQL functions through a series of example queries.

C.2.2.1

Find all features that feature `my:A` *contains, where spatial calculations are based on* `my:hasExactGeometry`.

```
PREFIX my: <http://example.org/ApplicationSchema#>
PREFIX geo: <http://www.opengis.net/ont/geosparql#>
PREFIX geof: <http://www.opengis.net/def/function/geosparql/>

SELECT ?f
WHERE {
    my:A my:hasExactGeometry ?aGeom .
    ?aGeom geo:asWKT ?aWKT .
    ?f my:hasExactGeometry ?fGeom .
    ?fGeom geo:asWKT ?fWKT .

    FILTER (
        geof:sfContains(?aWKT, ?fWKT) &&
            !sameTerm(?aGeom, ?fGeom)
        )
)
```

Result:

?f
`my:B`
`my:F`

C.2.2.2

Find all features that are within a transient bounding box geometry, where spatial calculations are based on `my:hasPointGeometry`.

```
PREFIX my: <http://example.org/ApplicationSchema#>
PREFIX geo: <http://www.opengis.net/ont/geosparql#>
PREFIX geof: <http://www.opengis.net/def/function/geosparql/>

SELECT ?f
WHERE {
    ?f my:hasPointGeometry ?fGeom .
    ?fGeom geo:asWKT ?fWKT .
    FILTER (
        geof:sfWithin(
            ?fWKT,
            "<http://www.opengis.net/def/crs/OGC/1.3/CRS84>
            Polygon ((-83.4 34.0, -83.1 34.0,
                        -83.1 34.2, -83.4 34.2,
                        -83.4 34.0))"^^geo:wktLiteral
        )
    )
}
```

Result:

?f
`my:D`

C.2.2.3

Find all features that touch the union of feature `my:A` *and feature* `my:D`*, where computations are based on* `my:hasExactGeometry`.

```
PREFIX my: <http://example.org/ApplicationSchema#>
PREFIX geo: <http://www.opengis.net/ont/geosparql#>
PREFIX geof: <http://www.opengis.net/def/function/geosparql/>

SELECT ?f
WHERE {
    ?f my:hasExactGeometry ?fGeom .
    ?fGeom geo:asWKT ?fWKT .
    my:A my:hasExactGeometry ?aGeom .
    ?aGeom geo:asWKT ?aWKT .
    my:D my:hasExactGeometry ?dGeom .
    ?dGeom geo:asWKT ?dWKT .
    FILTER (
        geof:sfTouches(
            ?fWKT,
            geof:union(?aWKT, ?dWKT)
        )
    )
}
```

Result:

?f
`my:C`

C.2.2.4

Find the 3 closest features to feature my:C, where computations are based on `my:hasExactGeometry`.

```
PREFIX uom: <http://www.opengis.net/def/uom/OGC/1.0/>
PREFIX my: <http://example.org/ApplicationSchema#>
PREFIX geo: <http://www.opengis.net/ont/geosparql#>
PREFIX geof: <http://www.opengis.net/def/geosparql/function>

SELECT ?f
WHERE {
    my:C my:hasExactGeometry ?cGeom .
    ?cGeom geo:asWKT ?cWKT .
    ?f my:hasExactGeometry ?fGeom .
    ?fGeom geo:asWKT ?fWKT .
    FILTER (?fGeom != ?cGeom)
}
ORDER BY ASC (geof:distance(?cWKT, ?fWKT, uom:metre))
LIMIT 3
```

Result:

?f
my:A
my:D
my:E

C.2.2.5

Find the maximum and minimum coordinates of a given set of geometries.

```
PREFIX geo: <http://www.opengis.net/ont/geosparql#>
PREFIX geof: <http://www.opengis.net/def/function/geosparql/>

SELECT ?minX ?minY ?minZ ?maxX ?maxY ?maxZ
WHERE {
    BIND ("<http://www.opengis.net/def/crs/OGC/1.3/CRS84>
            Polygon Z((-83.4 34.0 0, -83.1 34.0 1,
                        -83.1 34.2 1, -83.4 34.2 1,
                        -83.4 34.0 0))"^^geo:wktLiteral) AS ?testgeom)
    BIND(geof:minX(?testgeom) AS ?minX)
    BIND(geof:maxX(?testgeom) AS ?maxX)
    BIND(geof:minY(?testgeom) AS ?minY)
    BIND(geof:maxY(?testgeom) AS ?maxY)
    BIND(geof:maxZ(?testgeom) AS ?maxZ)
    BIND(geof:minZ(?testgeom) AS ?minZ)
}
```

Result:

?minX	?minY	?minZ	?maxX	?maxY	?maxZ
-83.4	34.0	0	-83.1	34.2	1

C.2.3 Example Rule Application

This section illustrates the query transformation strategy for implementing GeoSPARQL rules.

C.2.3.1

Find all features or geometries that overlap feature `my:A`.

Original Query:

```
PREFIX geo: <http://www.opengis.net/ont/geosparql#>

SELECT ?f
WHERE { ?f geo:sfOverlaps my:A }
```

Transformed Query (application of transformation rule geor:sfOverlaps):

```
PREFIX my: <http://example.org/ApplicationSchema#>
PREFIX geo: <http://www.opengis.net/ont/geosparql#>
PREFIX geof: <http://www.opengis.net/def/function/geosparql/>

SELECT ?f
WHERE {
    { # check for asserted statement
        ?f geo:sfOverlaps my:A }
    UNION
    { # feature □ feature
        ?f geo:hasDefaultGeometry ?fGeom .
        ?fGeom geo:asWKT ?fSerial .
        my:A geo:hasDefaultGeometry ?aGeom .
        ?aGeom geo:asWKT ?aSerial .
        FILTER (geof:sfOverlaps(?fSerial, ?aSerial))
    }
    UNION
    { # feature □ geometry
        ?f geo:hasDefaultGeometry ?fGeom .
        ?fGeom geo:asWKT ?fSerial .
        my:A geo:asWKT ?aSerial .
        FILTER (geof:sfOverlaps(?fSerial, ?aSerial))
    }
    UNION
    { # geometry □ feature
        ?f geo:asWKT ?fSerial .
        my:A geo:hasDefaultGeometry ?aGeom .
        ?aGeom geo:asWKT ?aSerial .
        FILTER (geof:sfOverlaps(?fSerial, ?aSerial))
    }
    UNION
    { # geometry □ geometry
        ?f geo:asWKT ?fSerial .
        my:A geo:asWKT ?aSerial .
        FILTER (geof:sfOverlaps(?fSerial, ?aSerial))
    }
```

Result:

?f
my:D
my:DExactGeom
my:E
my:EExactGeom

C.2.4 Example Geometry Serialization Conversion Functions

For the geometry literal values in C.1.2.4 Geometry Serializations:

Application of the function `geof:asWKT` to the GML, KML, GeoJSON and DGGS literals should return WKT literal and similarly for each of the other conversion methods, `geof:asGML`, `geof:asKML`, `geof:asGeoJSON` & `geof:asDGGS`.

Note that the application of `geof:asDGGS` requires a `specificDggsDatatype` parameter which indicates the particular DGGS literal form being converted to. In the case of C.1.2.4 Geometry Serializations, this value would be `eg:auspixDggsLiteral`, the example datatype of the AusPIX DGGS.

Annex D - Usage of SHACL shapes (informative)

D.0 Overview

This Annex provides guidance on the usage of the SHACL shapes included with GeoSPARQL 1.1.

The Shapes Constraint Language SHACL allows the specification of constraints on RDF data, phrased as a set of conditions modeled in "Shape" graphs.

In GeoSPARQL 1.1, SHACL Shapes area defined in such a way that they validate anticipated graph structures expected by requirements defined in the standard. Users may validate a given RDF document claiming conformance to GeoSPARQL 1.1 by using these Shapes and use the validation results to correct any mistakes.

D.1 Tools

SHACL Shapes provided with GeoSPARQL are used to verify the graph structure of GeoSPARQL graphs. There are several SHACL tools that one can using to validate data using this Shapes information:

- PySHACL: A Python implementation based on the RDF library RDFlib
- Apache Jena SHACL: a Java implementation, based on Apache Jena
- SHACL Playground: An online, JavaScript-based implementation that allows validation without local tools
- Triple Stores: SHACL validation is part of many triple store implementations:
  - GraphDB
  - RDF4J
  - Apache Jena Fuseki

Validators produce error messages and warnings based on the SHACL standard's defined reporting structure.

D.2 Scope of SHACL Shapes provided with GeoSPARQL

The SHACL Shapes defined in the GeoSPARQL 1.1 standard all target the verification of specific graph structures, but except very few cases do not validate the content of literal types. In particular, the following attributes of the graph are validated:

- **Proper usage of GeoSPARQL classes**: These Shapes check for a proper usage of instances of GeoSPARQL classes. For example, we check that instances of collection classes should at least have one element and that instances of Geometry classes should at least have one serilization to avoid creating graphs which contain nodes without necessary information.
- **Geometry property consistency**: Certain checks are applied for properties describing geometries. For example we check dimensionality properties for corresponding values
- **Rudimentary checks of literal contents**: The SHACL Shapes defined in this standard do not substitute a verification of literal contents by validators of the respective data formats. However, they define checks using regular expressions to detect a falsely formatted geospatial literal. For example, if a GeoJSON literal is declared using its literal type, a SHACL shape will check for curly brackets to be present (as they are part of the JSON specification)

D.3 Table of SHACL Shapes

SHACL Shape ID	Severity	Test purpose	Requirements tested
Shape 1a	Violation	Each node with an incoming geo:hasGeometry, or a specialization of it, should have minimum one outgoing relation that is either geo:hasSerialization, or a specialization of it.	[req_geometry-extension_feature_properties], [req_geometry-extension_geometry_properties]
Shape 1b	Violation	Each node with an incoming geo:hasGeometry, or a specialization of it, can have a maximum of one outgoing geo:asWKT relation.	[req_geometry-extension_geometry_properties] [req_geometry-extension_geometry-as-wkt-literal]
Shape 1c	Violation	Each node with an incoming geo:hasGeometry, or a specialization of it, can have a maximum of one outgoing geo:asGML relation.	[req_geometry-extension_geometry_properties] [req_geometry-extension_geometry-as-gml-literal]
Shape 1d	Violation	Each node with an incoming geo:hasGeometry, or a specialization of it, can have a maximum of one outgoing geo:asGeoJSON relation.	[req_geometry-extension_geometry_properties] [req_geometry-extension_geometry-as-geojson-literal]
Shape 1e	Violation	Each node with an incoming geo:hasGeometry, or a specialization of it, can have a maximum of one outgoing geo:asKML relation.	[req_geometry-extension_geometry_properties] [req_geometry-extension_geometry-as-kml-literal]

SHACL Shape ID	Severity	Test purpose	Requirements tested
Shape 2	Violation	Each node with one or more outgoing relations that are either geo:hasSerialization, or a specialization of it, should have at least one incoming geo:hasGeometry relation or a specialization of it.	[req_geometry-extension_geometry-properties]
Shape 3a-c	Violation	A node that has an incoming geo:hasGeometry property, or specialization of it, cannot have an outgoing geo:hasGeometry property, or a specialization of, it at the same time (a geo:Feature cannot be a geo:Geometry at the same time)	[req_geometry-extension_feature-properties]
Shape 4	Violation	The target of a geo:hasSerialization property, or a specialization of, it should be an RDF literal	[req_geometry-extension_geometry-properties]
Shape 5	Violation	The target of a geo:asWKT property should be an RDF literal with datatype geo:wktLiteral	[req_geometry-extension_wkt-literal]
Shape 6	Violation	The target of a geo:asGML property should be an RDF literal with datatype geo:gmlLiteral	[req_geometry-extension_gml-literal]

SHACL Shape ID	Severity	Test purpose	Requirements tested
Shape 7	Violation	The target of a geo:asGeoJSON property should be an RDF literal with datatype geo:geoJSONLiteral	[req_geometry-extension_geojson-literal]
Shape 8	Violation	The target of a geo:asKML property should be an RDF literal with datatype geo:kmlLiteral	[req_geometry-extension_kml-literal]
Shape 9	Violation	A geo:Geometry node should have maximum of one outgoing geo:coordinateDimension property	[req_geometry-extension_geometry-properties]
Shape 10	Violation	A geo:Geometry node should have maximum of one outgoing geo:dimension property	[req_geometry-extension_geometry-properties]
Shape 11	Violation	A geo:Geometry node should have maximum of one outgoing geo:isEmpty property	[req_geometry-extension_geometry-properties]
Shape 12	Violation	A geo:Geometry node should have a maximum one outgoing geo:isSimple property	[req_geometry-extension_geometry-properties]
Shape 13	Violation	A geo:Geometry node should have maximum of one outgoing geo:spatialDimension property	[req_geometry-extension_geometry-properties]
Shape 14a	Violation	A geo:Geometry node should have maximum of one outgoing geo:hasSpatialResolution property	[req_geometry-extension_geometry-properties]

SHACL Shape ID	Severity	Test purpose	Requirements tested
Shape 14b	Violation	A geo:Geometry node should have maximum of one outgoing geo:hasSpatialAccuracy property	[req_geometry-extension_geometry-properties]
Shape 14c	Violation	A geo:Geometry node should have maximum of one outgoing geo:hasMetricSpatialAccuracy property	[req_geometry-extension_geometry-properties]
Shape 14d	Violation	A geo:Geometry node should have maximum of one outgoing geo:hasMetricSpatialResolution property	[req_geometry-extension_geometry-properties]
Shape 15	Violation	The content of an RDF literal with an incoming geo:asWKT relation must conform to a well-formed WKT string, as defined by its official specification (Simple Features Access)	[req_geometry-extension_wkt-literal]
Shape 16	Violation	The content of an RDF literal with an incoming geo:asWKT relation must conform to a well-formed WKT string, as defined by its official specification (Simple Features Access)	[req_geometry-extension_gml-literal]
Shape 17	Violation	The content of an RDF literal with an incoming geo:asGeoJSON relation must conform to a well-formed GeoJSON geometry string, as defined by its official specification	[req_geometry-extension_geojson-literal]

SHACL Shape ID	Severity	Test purpose	Requirements tested
Shape 18	Violation	The content of an RDF literal with an incoming geo:asKML relation must conform to a well-formed KML geometry XML string, as defined by its official specification	[req_geometry-extension_kml-literal]
Shape 20	Violation	If both geo:dimension and geo:coordinateDimension properties are asserted, the value of geo:dimension should be less than or equal to the value of geo:coordinateDimension	[req_geometry-extension_geometry-properties]
Shape 21a	Violation	An instance of geo:FeatureCollection should have at least one outgoing rdfs:member relation	[req_core_spatial-feature-collection-class]
Shape 21b	Violation	An instance of geo:FeatureCollection should only have outgoing rdfs:member going to geo:Feature instances	[req_core_spatial-feature-collection-class]
Shape 22a	Violation	An instance of geo:GeometryCollection should have at least one outgoing rdfs:member relation	[req_core_spatial-geometry-collection-class]
Shape 22b	Violation	An instance of geo:GeometryCollection should only have outgoing rdfs:member relations to geo:Geometry instances	[req_core_spatial-geometry-collection-class]

SHACL Shape ID	Severity	Test purpose	Requirements tested
Shape 23a	Violation	An instance of geo:SpatialObjectCollection should have at least one outgoing rdfs:member relation	[req_core_spatial-object-collection-class]
Shape 23b	Violation	An instance of geo:SpatialObjectCollection should only have outgoing rdfs:member relations going to geo:SpatialObject instances, or subclasses of them	[req_core_spatial-object-collection-class]

Annex E - Alignments (informative)

E.0 Overview

This Annex provides alignments of GeoSPARQL to other well known ontologies that are either commonly used with GeoSPARQL or could be.

The prefixes used for the ontologies mapped to in all following sections are given in the following table.

dcterms:	http://purl.org/dc/terms/
geo:	http://www.opengis.net/ont/geosparql#
geom:	http://geovocab.org/geometry#
gn:	https://www.geonames.org/ontology#
juso:	http://rdfs.co/juso/
lgd:	http://linkedgeodata.org/ontology/
locn:	https://www.w3.org/ns/locn
obo:	http://purl.obolibrary.org/obo/
osm:	https://w3id.org/openstreetmap/terms#
osmm:	https://www.openstreetmap.org/meta/
osmt:	https://wiki.openstreetmap.org/wiki/Key:
pos:	http://www.w3.org/2003/01/geo/wgs84_pos#
prov:	http://www.w3.org/ns/prov#
rdf:	http://www.w3.org/1999/02/22-rdf-syntax-ns#
rdfs:	http://www.w3.org/2000/01/rdf-schema#
sdo:	https://schema.org
sosa:	http://www.w3.org/ns/sosa/
spatialuk:	http://data.ordnancesurvey.co.uk/ontology/spatialrelations/
spatialukgeom:	http://data.ordnancesurvey.co.uk/ontology/geometry/
spatial:	http://geovocab.org/spatial#
ssn:	http://www.w3.org/ns/ssn/
time:	http://www.w3.org/2006/time#
wdt:	http://www.wikidata.org/entity/

E.1 ISA Programme Location Core Vocabulary (LOCN)

LOCN Source: https://www.w3.org/ns/locn

The LOCN specification provides notes on the use of GeoSPARQL literals (see https://www.w3.org/ns/locn#changes).

From Element	Mapping relation	To Element	Notes
`geo:Feature`	`rdfs:subClassOf`	`dcterms:Location`	LOCN states that `dcterms:Location` "represents any location, irrespective of size or other restriction". As such, it can be considered as a superclass of `geo:Feature`.
`locn:Address`	`rdfs:subClassOf`	`geo:Feature`	Although LOCN does not explicitly indicate spatial or geometry properties for `locn:Address`, this class can be considered as a specialized form of a `geo:Feature`.
`geo:Geometry`	`rdfs:subClassOf`	`locn:Geometry`	In LOCN, class `locn:Geometry` "[…] defines the notion of geometry at the conceptual level, and it shall be encoded by using different formats". More precisely, its instances can be either literals or individuals. The GeoSPARQL's class `geo:Geometry` is more narrowly defined, as its instances can only be individuals, and not literals.

From Element	Mapping relation	To Element	Notes
`geo:hasGeometry`	`rdfs:subPropertyOf`	`locn:geometry`	In LOCN, the usage note to property `locn:geometry` states that "Depending on how a geometry is encoded, the range of this property may be one of the following: a literal […], an instance of a geometry class […], geocoded URIs […]". The GeoSPARQL's property `geo:hasGeometry` is more narrowly defined, as it can only be used with instances of `geo:Geometry`, and not with literals.

E.2 WGS84 Geo Positioning: an RDF vocabulary (POS)

POS Source: http://www.w3.org/2003/01/geo/

From Element	Mapping relation	To Element	Notes
`geo:SpatialObject`	`owl:equivalentClass`	`pos:SpatialThing`	Both classes are unrestricted, essentially abstract classes
`pos:Point`	`rdfs:subClassOf`	`geo:Geometry`	Via `pos:Point` `rdfs:subClassOf` `pos:SpatialThing` but since `pos:Point` usage notes indicates direct postitioning, it is a form of geometry
`pos:Point`	`owl:equivalentClass`	`sf:Point`	
`pos:lat_long`	`rdfs:subPropertyOf`	`geo:hasSerialization`	A special datatype is not indicated for use with this property by POS, unlike GeoSPARQL's `geo:hasSerialization` object literals
`pos:location`	`rdfs:subPropertyOf`	`rdfs:hasGeometry`	

E.3 Geonames Ontology (GN)

Geonames source: http://www.geonames.org/ontology/documentation.html

From Element	Mapping relation	To Element	Notes
`gn:Feature`	`owl:equivalentClass`	`geo:Feature`	
`gn:GeonamesFeature`	`rdfs:subClassOf`	`geo:Feature`	The GN class is defined as "A feature described in geonames database... "
`geo:Feature`	`rdfs:subClassOf`	`gn:Class`	The GN class' definition reads "A class of features"
`gn:locatedIn`	`owl:equivalentProperty`	`geo:sfWithin`	
`gn:nearby`	`rdfs:subPropertyOf`	`geo:sfDisjoint`	A `gn:nearby` B means A is not within or touching B. The only close SF property is disjoint
`gn:neighbour`	`owl:equivalentProperty`	`geo:sfTouches`	

E.4 NeoGeo Vocabulary

NeoGeo Source: http://geovocab.org/ / http://geovocab.org/doc/neogeo/

From Element	Mapping relation	To Element	Notes
spatial:Feature	owl:equivalentClass	geo:Feature	
spatial:C	rdfs:subPropertyOf	geo:rcc8ec	Sub proerty not equivalent property since the NeoGeo property has more restrictive domain & range
spatial:DR	rdfs:subPropertyOf	geo:rcc8dc	
spatial:EC	rdfs:subPropertyOf	geo:rcc8ec	
spatial:EQ	rdfs:subPropertyOf	geo:rcc8eq	
spatial:NTPP	rdfs:subPropertyOf	geo:rcc8ntpp	
spatial:NTPPi	rdfs:subPropertyOf	geo:rcc8ntppi	
spatial:O	rdfs:subPropertyOf	geo:sfOverlaps	
spatial:P	rdfs:subPropertyOf	geo:sfWithin	
spatial:PO	rdfs:subPropertyOf	geo:rcc8po	
spatial:PP	rdfs:subPropertyOf	geo:sfWithin	
spatial:PPi	rdfs:subPropertyOf	geo:sfContains	
spatial:Pi	rdfs:subPropertyOf	geo:sfContains	
spatial:TPP	rdfs:subPropertyOf	geo:rcc8tpp	
spatial:TPPi	rdfs:subPropertyOf	geo::rcc8tppi	
geom:Geometry	owl:equivalentClass	geo:Geometry	
geom:BoundingBox	rdfs:subClassOf	geo:Geometry	GeoSPARQL doesn't have a BoundingBox class but has a generic Geometry class that is the range of the geo:hasBoundigBox property
geom:GeometryCollection	owl:equivalentClass	geo:GeometryCollection	
geom:LineString	owl:equivalentClass	sf:LineString	
geom:LinearRing	owl:equivalentClass	sf:LinearRing	
geom:MultiLineString	owl:equivalentClass	sf:MultiLineString	
geom:MultiPoint	owl:equivalentClass	sf:MultiPoint	
geom:MultiPolygon	owl:equivalentClass	sf:MultiPolygon	
geom:Polygon	owl:equivalentClass	sf:Polygon	
geom:Point	owl:equivalentClass	sf:Point	

From Element	Mapping relation	To Element	Notes
`geo:hasGeometry`	`rdfs:subPropertyOf`	`geom:geometry`	`geo:hasGeometry` has more restrictve domain

- The `geom:bbox` property relates a Geometry to another Geometry and is thus not equivalent to GeoSPARQL's Feature-to-Geometry `geo:hasBoundingBox`.
  - An equivalent to `geo:bbox` could be made using a `geo:Feature` with a `geo:Geometry`, indicated by `geo:hasGeometry` and a second, specialised Bounding Box `geo:Geometry` indicated with `geo:hasBoundingBox`

E.5 Juso Ontology

Juso Source: http://rdfs.co/juso/

Juso contains mappings to GeoSPARQL but uses `owl:sameAs` which it should instead use `owl:equivalentClass`.

From Element	Mapping relation	To Element
`juso:SpatialThing`	`owl:equivalentClass`	`geo:SpatialObject`
`juso:Feature`	`owl:equivalentClass`	`geo:Feature`
`juso:Geometry`	`owl:equivalentClass`	`geo:Geometry`
`juso:Point`	`owl:equivalentClass`	`sf:Point`
`juso:geometry`	`owl:equivalentProperty`	`geo:hasGeometry`
`juso:parent`	`rdfs:subPropertyOf`	`geo:sfWithin`
`juso:political_division`	`rdfs:subPropertyOf`	`geo:sfContains`
`juso:within`	`owl:equivalentProperty`	`geo:sfWithin`

E.6 Time Ontology in OWL (TIME)

TIME Source: https://www.w3.org/TR/owl-time/

There are no direct class or property correspondences between GeoSPARQL and TIME however class patterning is similar:

- TIME uses `time:hasTime` to indicate that something has a temporal projection
- GeoPSARQL uses `geo:hasGeometry` to indicate that a `geo:Feature` has a spatial projection

and

- TIME uses properties such as `time:inXSDDate` to indicate the position of temporal entities on a temporal reference system
- GeoSPARQL uses properties such as `geo:asWKT` to indicate the position of spatial entities (Geometries) on spatial reference systems

OWL TIME sets no domain for `time:hasTime` thus this property may be used with anything, including a GeoSPARQL `geo:Feature` so that a spati-temporal Feature may be indicated like this:

```
:flooded-area-x
    a geo:Feature ;
    geo:hasGeometry [
        a geo:Geometry ;
        geo:asWKT "POLYGON (((...)))"^^geo:wktLiteral ;
    ] ;
    time:hasTime [
        a time:ProperInterval ;
        time:hasBeginning [
            time:inXSDDate "..."^^xsd:date ;
        ] ;
        time:hasEnd [
            time:inXSDDate "..."^^xsd:date ;
        ] ;
    ] ;
.
```

In the above example, `:flooded-area-x` is a spatio-temporal Feature that has both a GeoSPARQL spatial projection - a `geo:Geometry` - and a temporal projection - a `time:ProperInterval` which is a specailised form of `time:TemporalEntity`.

Another possible use of TIME with GeoSPARQL is to assign temporality to individual `geo:Geometry` instances. This is allowed given `time:hasTime's open domain:

```
:flooded-area-x
    a geo:Feature ;
    geo:hasGeometry [
        a geo:Geometry ;
        geo:asWKT "POLYGON (((...)))"^^geo:wktLiteral ;
        time:hasTime [ ... ] ;
    ] ;
.
```

In contrast to the first example, `:flooded-area-x` is inferred to be a spatio-temporal Feature but since it is the Geometry of `:flooded-area-x` that has a temporality, it is possible to describe other Geometries of `:flooded-area-x` with other temporalities.

E.7 schema.org

schema.org Source: https://schema.org

From Element	Mapping relation	To Element	Notes
geo:Geometry	rdfs:subClassOf	sdo:GeoShape	A GeoShape can various literal geometry representation
sdo:GeospatialGeometry	owl:equivalentClass	geo:SpatialObject	Since GeospatialGeometry is the domain of SimpleFeature-like properties and a superclass of GeoShape
sdo:GeoCoordinates	rdfs:subClassOf	geo:Geometry	GoCoordinates uses direct lat, long, elevation etc properties to indicate position, not a while geometry serialization but it is nevertheless a form of a Geometry
sdo:geo	rdfs:subPropertyOf	geo:hasGeometry	
sdo:geoCoveredBy	owl:equivalentProperty	geo:ehCoveredBy	
sdo:geoCovers	owl:equivalentProperty	geo:ehCovers	
sdo:geoCrosses	owl:equivalentProperty	geo:sfCrosses	
sdo:geoDisjoint	owl:equivalentProperty	geo:sfDisjoint	
sdo:geoEquals	owl:equivalentProperty	geo:sfEquals	
sdo:geoIntersects	owl:equivalentProperty	geo:sfIntersects	
sdo:geoOverlaps	owl:equivalentProperty	geo:sfOverlaps	
sdo:geoTouches	owl:equivalentProperty	geo:sfTouches	
sdo:geoWithin	owl:equivalentProperty	geo:sfWithin	
sdo:Landform	rdfs:subClassOf	geo:Feature	

E.8 Semantic Sensor Network Ontology (SSN)

SSN Source: https://www.w3.org/TR/vocab-ssn/

SSN and GeoSPARQL do not cover overlapping concerns directly and therefore there are no direct class or property correspondences between them, however SSN provides advice on the use of GeoSPARQL for location, see Section 7.1 (https://www.w3.org/TR/vocab-ssn/#x7-1-location):

> GeoSPARQL … provides a flexible and relatively complete platform for geospatial objects, that fosters interoperability between geo-datasets. To do so, these entities can be declared as instances of geo:Feature and geometries can be assigned to them via the geo:hasGeometry property. In case of classes, e.g., specific features of interests such as rivers, these can be defined as subclasses of geo:Feature.

E.9 DCMI Metadata Terms (DCTERMS)

DCTERMS Source: https://www.dublincore.org/specifications/dublin-core/dcmi-terms/

From Element	Mapping relation	To Element	Notes
`geo:Feature`	`rdfs:SubClassOf`	`dcterms:Location`	A Location is a "A spatial region or named place."
`geo:hasGeometry`	`rdfs:subPropertyOf`	`dcterms:spatial`	`dcterms:spatial` indicates the "Spatial characteristics of the resource", thus it is a more general form of GeoSPARQL's `geo:hasGeometry` which indicates geometry spatial information

- `dcterms:coverage` is extremely generic - "The spatial or temporal topic of the resource, spatial applicability of the resource, or jurisdiction under which the resource is relevant." - but DCTERMS indicates its range includes a `dcterms:Location`, so it is a property for indicating a `geo:Feature`, not a `geo:Geometry` and for which GeoSPARQL has no equivalent. Often, `dcterms:coverage` is used to indicate a spatial extent such as a bounding box but `dcterms:spatial` could be used for this with more precsision. GeoSPARQL now provides a `geo:hasBoundingBox` property, so such a property could be used if a Bounding Box is wanted to be indicated.

DCTERMS-related geometry literals, such as the *DCMI Box Encoding Scheme_*[14] *and the _DCMI Point Encoding Scheme_*[15] *could be indicated as GeoSPARQL geometry literals if a literal datatype were created for each. For example, the _DCMI Point Encoding Scheme* example of "The highest point in Australia" with the literal value `east=148.26218; north=-36.45746; elevation=2228; name=Mt. Kosciusko` might be encoded in GeoSPARQL like this:

```
:mt-kosciusko
    a geo:Feature ;
    geo:hasGeometry [
        a geo:Geometry ;
        geo:hasSerialization "east=148.26218; north=-36.45746; elevation=2228;
name=Mt. Kosciusko"^^ex:dcmiPoint ;
    ] ;
.
```

[14] https://www.dublincore.org/specifications/dublin-core/dcmi-box/

[15] https://www.dublincore.org/specifications/dublin-core/dcmi-point/

E.10 The Provenance Ontology (PROV)

PROV Source: https://www.w3.org/TR/prov-o/

From GeoSPARQL's point of view, PROV is an "upper" ontology - one dealing with more abstract concepts - and only one of PROV's three main classes of object - `Entity`, `Activity` & `Agent` - has direct relations to GeoSPARQL classes and that is `Entity`. This is because GeoSPARQL characterises things - spatial objects - which are a kind of `Entity` but does not deal with events (`Activity`) or things with agency (`Agent`).

From Element	Mapping relation	To Element	Notes
`geo:SpatialObjectCollection`	`rdfs:subClassOf`	`prov:Collection`	PROV's class is a generic collection class and GeoSPARQL's property is clearly a specialised form of it that may only consist of certain class instances (`geo:SpatialObject`)
`geo:SpatialObject`	`rdfs:subClassOf`	`prov:Entity`	All SpatialObjects fit within PROV's Entity's definition: "An entity is a physical, digital, conceptual, or other kind of thing with some fixed aspects; entities may be real or imaginary."
`geo:Feature`	`rdfs:subClassOf`	`prov:Location`	A Location "...can be an identifiable geographic place (ISO 19112), but it can also be a non-geographic place such as a directory, row, or column" so seem to be wider in scope than GeoSPARQL's Feature although a Feature could indeed be something such as a "directory, row, or column"

- The PROV property `prov:atLocation` indicates `prov:Location` instances, which may be `geo:Feature` instnaces, but GeoSPARQL has no property to indicate a `geo:Feature`, so no mapping is possible. Indicating features is commonly done in ontologies used GeoSPARQL but not within

GeoSPARQL.

- Derivative relations between GeoSPARQL objects could be modelled using PROV, for instance a BoundingBox may be indicated as haveing been derived from a Polygon like this:

```
:bounding-box-y prov:wasDerivedFrom :polygon-x .
```

E.11 WikiData

From Element	Mapping relation	To Element	Notes
`wdt:P625`	`owl:equivalentProperty`	`geo:asWKT`	The Wikidata description of this property labeled "coordinate location" note that "For Earth, please note that only WGS84 coordinating system is supported at the moment" but that is a system limit, not an ontological one
`wdt:P3896`	`owl:propertyChainAxiom`	`(geo:hasGeometry geo:asGeoJSON)`	This Wikidata property labeled "geoshape" indicated GeoJSON geomettry literal content for a Feature, but it allows information other than just Geometry in the GeoJSON whereas GeoSPARQL does not.
`wdt:P3096`	`owl:propertyChainAxiom`	`(geo:hasGeometry geo:asKML)`	This Wikidata property labeled "KML File" links to a KML file which is related to the respective instance. This may not be the same representation as in GeoSPARQL, as GeoSPARQL KML literals only encode the geometry part of a KML.

From Element	Mapping relation	To Element	Notes
`wd:Q82794`	`rdfs:subClassOf`	`geo:Feature`	The Wikidata class is labeled "geographic region" and thus is a subclass of the more general `geo:Feature`. There are likely many other classes in Wikidata that could be interpreted as subclasses of `geo:Feature`
`wd:Q618123`	`owl:equivalentClass`	`geo:Feature`	The Wikidata class is labeled "geographical feature" and thus corresponds to `geo:Feature`.
`wd:Q25404640`	`owl:equivalentClass`	`geo:SpatialObject`	The Wikidata class is labeled "spatial object" and thus corresponds to `geo:SpatialObject`.
`wdt:P150`	`rdfs:subPropertyOf`	`geo:sfContains`	The Wikidata property is labeled "contains administrative territorial entity" but also alternatively labeled "contains", "has districts" and others. There are likely many other specialised forms of `geo:sfContains` and `geo:sfWithin` in Wikidata

From Element	Mapping relation	To Element	Notes
geo:sfWithin	rdfs:subPropertyOf	wdt:P361	The Wikidata property is labeled "part of" and is sometimes used to indicate Feature parthood. There are likley other parthood properties like this in Wikipedia that may also be used as superproperties of GeoSPARQL feature relations properties. The Wikidata inverse is wdt:Q65964571 "has part"
geo:sfContains	rdfs:subPropertyOf	wd:Q65964571	The property labeled "has part" is the inverse of wdt:P361 (see above)
wdt:P131	rdfs:subPropertyOf	geo:sfContains	The Wikidata property is labeled "located in the administrative territorial entity" and is essentially the inverse of wdt:150 (described above)
wdt:P706	rdfs:subPropertyOf	geo:sfWithin	The Wikidata property is labeled "located in/on physical feature" and is indicated for use with a "(geo)physical feature" and not to be used for administrative features where wdt:131 (see above) should be

From Element	Mapping relation	To Element	Notes
`wdt:P4688`	`rdfs:subClassOf`	`geo:Feature`	The Wikidata class is labeled "geomorphological unit" and is one of many Wikidata feature classes that could be expressed as a subclass of `geo:Feature`. More specailised geological unit examples are `Q5107` "continent" and `wdt:P4552` "mountain range".
`wdt:P2046`	`owl:equivalentProperty`	`geo:hasArea`	The Wikidata property is labeled "area". It indicates a microformat - NUMBER + SPACE + ALLOWED_UNIT_LABEL - with a fixed set of ALLOWED_UNIT_LABELs to present values and units of measure.

E.12 OpenStreetMap Ontologies

There are several approaches to make OpenStreetMap data accessible in the Linked Open Data cloud.

E.12.1 LinkedGeoData

LinkedGeoData emerged from a resarch project linking

From Element	Mapping relation	To Element	Notes
Any LGD Class	`rdfs:subClassOf`	`geo:Feature`	Any class defined in the LinkedGeoData ontology is a subclass of geo:Feature

E.12.2 OpenStreetMap RDF (Sophox)

https://wiki.openstreetmap.org/wiki/Sophox#How_OSM_data_is_stored

From Element	**Mapping relation**	**To Element**	**Notes**
`osmm:loc`	`owl:equivalentProperty`	`geo:asWKT`	The OpenStreetMap RDF property osmm:loc includes WKTliterals which depending on the type of the subject instance describe an OSM node or the centroid of a way or OSM relation
`osmm:type` 'n'	`owl:equivalentClass`	`sf:Point`	The OpenStreetMap RDF property osmm:type with value 'n' describes an OSM Node which is equivalent to a sf:Point
`osmm:type` 'w'	`owl:equivalentClass`	`sf:LineString`	The OpenStreetMap RDF property osmm:type with value 'w' describes an OSM Way which is equivalent to a sf:LineString

From Element	Mapping relation	To Element	Notes
`osmm:type` 'r'	`owl:equivalentClass`	`sf:GeometryCollection`	The OpenStreetMap RDF property osmm:type with value 'r' describes an OSM relation Way which is equivalent to a sf:GeometryCollection
`osmm:has`	`owl:equivalentProperty`	`geo:sfContains`, `geo:ehContains`, `geo:rcc8ntpp`	The OpenStreetMap RDF property osmm:has describes that a relation contains a way or that a way contains a node
`osmm:isClosed` true	`owl:equivalentClass`	`sf:Polygon`	The OpenStreetMap RDF property osmm:isClosed indicates whether a Way is closed, i.e. if it constitutes a Polygon
`osmm:isClosed` false	`owl:equivalentClass`	`sf:LineString`	The OpenStreetMap RDF property osmm:isClosed indicates whether a Way is closed, i.e. if it constitutes a Polygon

E.12.3 Routable Tiles Ontology

https://github.com/openplannerteam/routable-tiles-ontology

From Element	Mapping relation	To Element	Notes
`osm:Element`	`owl:equivalentClass`	`geo:Geometry`	The class osm:Element is equivalent to a geo:Geometry
`osm:Node`	`owl:equivalentClass`	`sf:Point`	The class osm:Node is equivalent to a sf:Point
`osm:Way`	`owl:equivalentClass`	`sf:LineString`	The class osm:Way is equivalent to a sf:LineString
`osm:Relation`	`owl:equivalentClass`	`sf:GeometryCollection`	The class osm:Relation is equivalent to a sf:GeometryCollection

E.13 Ordnance Survey UK Spatial Ontology

http://www.ordnancesurvey.co.uk/legacy/ontologies/spatialrelations.owl & http://www.ordnancesurvey.co.uk/legacy/ontologies/geometry.owl

: These two ontologies will be withdrawn during 2022.

The ontology authors note: "We are pleased to have contributed to the discussion some ten years ago but recognise that the subject area has moved on. We would not recommend people starting to relate to our ontology now, and we look forward to migrating to some more authoritative one in due course."

From Element	Mapping relation	To Element	Notes
`spatialuk:contains`	`owl:equivalentProperty`	`geo:sfContains`	
`spatialuk:disjoint`	`owl:equivalentProperty`	`geo:sfDisjoint`	
`spatialuk:easting`	`owl:equivalentProperty`	-	Distance in metres east of National Grid origin
`spatialuk:equals`	`owl:equivalentProperty`	`geo:sfEquals`	
`spatialuk:northing`	`owl:equivalentProperty`	-	Distance in metres north of National Grid origin
`spatialuk:touches`	`owl:equivalentProperty`	`geo:sfTouches`	
`spatialuk:within`	`owl:equivalentProperty`	`geo:sfWithin`	
`spatialukgeom:AbstractGeometry`	`owl:equivalentProperty`	`geo:Geometry`	
`spatialukgeom:extent`	`owl:equivalentProperty`	`geo:hasGeometry`	The range of spatialukgeom:extent is constrained to 2D geometries
`spatialukgeom:asGML`	`owl:equivalentProperty`	`geo:asGML`	The properties are equivalent, but the range of spatialukgeom:asGML is more general: An rdf:XMLLiteral

- `spatialuk:easting` describes a latitude coordinate east of the national UK grid and GeoSPARQL does not contain modelling of individual coorinate reference system elements
- `spatialuk:northing` describes a longitude coordinate north of the national UK grid so, as above, has not GeoSPARQL equivalent

E.14 CIDOC CRM Geo

CRMGeo Source: https://www.cidoc-crm.org/crmgeo/sites/default/files/CRMgeo1_2.pdf

From Element	Mapping relation	To Element	Notes
cidoc:SP1_PhenomenalSpaceTimeVolume	rdfs:subClassOf	geo:Feature	The CIDOC CRMgeo class SP1_PhenomenalSpaceTimeVolume is a subclass of geo:Feature as described in the CRMgeo 1.2 specification document.
cidoc:SP2_PhenomenalPlace	rdfs:subClassOf	geo:Feature	The CIDOC CRMgeo class SP2_PhenomenalPlace is a subclass of geo:Feature as described in the CRMgeo 1.2 specification document.
cidoc:SP5_GeometricPlaceExpression	rdfs:subClassOf	geo:Geometry	The CIDOC CRMgeo class SP5_GeometricPlaceExpression is a subclass of geo:Geometry as described in the CRMgeo 1.2 specification document.
cidoc:SP6_DeclarativePlace	rdfs:subClassOf	geo:Geometry	The CIDOC CRMgeo class SP6_DeclarativePlace is a subclass of geo:Geometry as described in the CRMgeo 1.2 specification document.
cidoc:SP7_DelcarativePlace	rdfs:subClassOf	geo:Geometry	The CIDOC CRMgeo class SP7_DelcarativePlace is a subclass of geo:Geometry as described in the CRMgeo 1.2 specification document.

From Element	Mapping relation	To Element	Notes
`cidoc:SP10_DeclarativeTimeSpan`	`rdfs:subClassOf`	`geo:Geometry`	The CIDOC CRMgeo class SP10_DeclarativeTimeSpan is a subclass of geo:Geometry as described in the CRMgeo 1.2 specification document.
`cidoc:SP14_TimeExpression`	`rdfs:subClassOf`	`geo:Geometry`	The CIDOC CRMgeo class SP14_TimeExpression is a subclass of geo:Geometry as described in the CRMgeo 1.2 specification document.
`cidoc:SP15_Geometry`	`rdfs:subClassOf`	`geo:Geometry`	The CIDOC CRMgeo class SP15_Geometry is a subclass of geo:Geometry as described in the CRMgeo 1.2 specification document.

E.15 Basic Formal Ontology (BFO)

BFO Source: https://basic-formal-ontology.org/bfo-2020.html, and from there, an OWL ontology of BFO2020 at https://github.com/BFO-ontology/BFO-2020

From Element	Mapping relation	To Element	Notes
`geo:SpatialObject`	`rdfs:subClassOf`	`obo:BFO_0000004` "independent continuant"	BFO's "independent continuant" is the superclass of "material entity" & "immaterial entity" which are mapped to Feature & Geometry respectively, so at least some independent continuants must be Spatial Objects
`obo:BFO_0000040` "material entity"	`rdf:subClassOf`	`geo:Feature`	A BFO "material entity" is something that "has some portion of matter as continuant part" so some Features are such, however Features may be imaginary too
`obo:BFO_0000029` "site"	`rdf:subClassOf`	`geo:Feature`	BFO's sites either covert the same areas as or have locations determined in relation to material entities so sites are Features but not necissarily the other way around
`geo:Geometry`	`owl:equivalentClass`	`obo:BFO_0000006` "spatial region"	BFO's "spatial region" class is described as a "spatial projection of a portion of spacetime" so Geometry appears to be equivalent. The BFO example indicates it's for 1-, 2- & 3-D spatial regions, as Geometry is.
`geo:Geometry`	`rdfs:subClassOf`	`obo:BFO_0000011` "spatiotemporal region"	GEO Geometry doesn't contain temporality but all GEO Geometry instances can sensibly be assumed to be within time, even if imaginary, so the BFO class is the superclass. Note that this mapping is also inherited from `geo:Geometry` being an equivalent class of `obo:BFO_0000006`, so consistency of alignment is retained. This mapping is made to highlight GeoSPARQL's lack of temporality representation
`geo:hasGeometry`	`rdfs:subPropertyOf`	`obo:BFO_0000211` "occupies spatial region at all times"	The BFO property links a thing that is not a spatial region to a spatial region, so it can be used as `geo:hasGeometry` is used when the thing is taken to be a `geo:Feature` and the spatial region a `geo:Geometry`. No GeoSPARQL temporality indicators mean mappings are eternal.

From Element	Mapping relation	To Element	Notes
`geo:hasGeometry`	`rdfs:subPropertyOf`	`obo:BFO_0000210` "occupies spatial region at some time"	A transitive mapping from the mapping above. Temporal qualification can be used with GeoSPARQL, see the OWL TIME alignment.
`geo:sfWithin`	`rdfs:subPropertyOf`	`obo:BFO_0000082` "located in at all times"	The BFO property "located in at all times" is a super property of `geo:sfWithin` when the thing located in the spatial region are defined to both be instances of `geo:Feature`. Since GeoSPARQL natively supplies no temporal qualifiers, pure GeoSPARQL assertions are assumed to be eternal: "…at all times"
`geo:sfWithin`	`rdfs:subPropertyOf`	`obo:BFO_0000171` "located in at some time"	A transitive mapping from the mapping above. Temporal qualification can be used with GeoSPARQL, see the OWL TIME alignment.
`obo:BFO_0000066` "occurs in"	`rdfs:range`	`geo:SpatialObject`	The BFO property relates a temporal activity to a spatial region but since GeoSPARQL has no notion of events, no mapping to this property can be made. However, BFO indicates this property should be used with a BFO "spatial region" (`geo:Geometry`) range value but from GeoSPARQL's point of view, it could also be used with a `geo:Feature` where the "in" would be taken to be within the feature's geometry, so the superclass of feature and geometry is given as the range
`obo:BFO_0000216` "spatially projects onto at some time"	`rdfs:range`	`geo:SpatialObject`	The reasoning is the same as for "occurs in"

Annex F - CQL / GeoSPARQL Mapping (informative)

F.0 Overview

This annex presents a mapping between the Common Query Language(CQL) [CQLDEF] and GeoSPARQL as well as generic SPARQL [SPARQL]. This is likely of relevance to the the delivery of GeoSPARQL data via systems such as the OGC's Web Feature Service [WFS] and OGC API Features [OGCAPIF] which implement CQL.

F.1 Accessing spatial Features in a SPARQL endpoint

Spatial *Features* accessed via SPARQL endpoints [SPARQLPROT] are, as defined in the GeoSPARQL standard, instances of the OWL class `geo:Feature` or of subclasses of it. They may have one or more `geo:hasGeometry` properties indicating `geo:Geometry` instances and other properties related to the Feature. They may also be grouped into `geo:FeatureCollection` instances where `geo:FeatureCollection` is a new class in GeoSPARQL 1.1, specifically for the description of collections of `geo:Feature` instances.

The following example SPARQL query retrieves all Features within the Feature Collection with the IRI `ex:x` within a given SPARQL endpoint.

```
SELECT ?fcollection ?item ?rel ?val ?geom
WHERE {
  ex:x rdfs:member ?item .
  ?item rdfs:subClassOf* geo:Feature .
}
```

GeoSPARQL's `geo:FeatureCollection` definition requires that `geo:Feature` instances are to be linked to the Collection by use of the `rdf:member` property. No inverse property is defined.

> **NOTE** Some CQL-implementing systems, such as OGC API, have fixed notions of Feature Collections and require that Features be members of exactly one Feature Collection. There is no such restriction in GeoSPARQL: Features may be members of one or more Feature Collections.

An extension to the above can retrieve any Geometry serializations for the Features within Feature Collection `ex:x`:

```
SELECT ?fcollection ?item ?rel ?val ?geom
WHERE {
  ex:x rdfs:member ?item .
  ?item rdfs:subClassOf* geo:Feature .

  OPTIONAL {
    ?item geo:hasGeometry/geo:hasSerialization ?geom
  }
}
```

Some additional concerns for GeoSPARQL / CQL or OGC API Features Feature Collections mappings are:

- APIs may need more information about the `geo:FeatureCollection` instance for correct handling, in particular, an identifier and perhaps a label. If the back-end data store also contains information for the `geo:FeatureCollection` instance then this may be queried for. If not, the API might need to create such data

  - One particular scenario observed is that OGC APIs require token-like identifiers for Feature Collections and GeoSPARQL IRIs, or their parts, may not be able to be used for such. In these cases, the RDF property `dcterms:identifier` may be used to store appropriate token-like identifiers

- Perhaps only data in a certain namespace is of interest. The solution is to apply FILTER expressions to the SPARQL query

F.2 Mappings from CQL2 statements to GeoSPARQL queries

This section presents lists of equivalences between Common Query Language (CQL2) [CQLDEF] statements and GeoSPARQL statements.

F.2.1 Query Parameters

Several query parameters may be given as parameters to the HTTP request of CD-implementing systems, such as the OGC API Features service. These parameters have an influence on the SPARQL query to be executed for the retrieval of a FeatureCollection to be exposed using an OGC API Features service.

Query Parameter	Example	SPARQL Expression	Example	Comment
limit	limit=5	LIMIT	LIMIT 5	
offset	offset=10	OFFSET	OFFSET 10	
bbox	`bbox=160.6,-55.95,-170,-25.89`	`FILTER( geo:sfIntersects())`	`FILTER(geo:sfIntersects(?geom, "POLYGON((160.6 -55.95,160.6 -25.89, -170 -25.89, -170 -55.95, 160.6 -55.95))"^^geo:wktLiteral))`	WKT does not define a type boundingbox, therefore a bbox is converted to a Polygon
datetime	`datetime=2018-02-12T23%3A20%3A52Z`	-	-	GeoSPARQL doesn't detail temporal aspects of data. Filtering data using RDF temporal properties may be achieved using basic SPARQL queries and also OWL TIME [TIME]

F.2.2 Literal Values

CQL2 defines literal values for a variety of datatypes. The following table shows the equivalences of these values in RDF which may be used in any GeoSPARQL query.

CQL2 literal	Examples	(Geo)SPARQL literal	Examples
String	"This is a string"	`xsd:string`	"This is a string"^^xsd:string

CQL2 literal	Examples	(Geo)SPARQL literal	Examples
Number	-100 3.14159	`xsd:int`, `xsd:integer`, `xsd:double`	"-100"^^xsd:integer "3.14159"^^xsd:double
Boolean	true false	`xsd:boolean`	"true"^^xsd:boolean "false"^^xsd:boolean
Spatial Geometry (WKT)	POINT(1 1)	WKT Literal	"POINT(1 1)"^^geo:wktLiteral
Spatial Geometry (JSON)	{"type": "Point", "coordinates":[1,1]}	GeoJSON Literal	"{"type": "Point", "coordinates":[1,1]}"^^geo:geoJSONLiteral
Temporal Literal	1969-07-20 1969-07-20T20:17:40Z	`xsd:date`, `xsd:dateTime`, `xsd:dateTimeStamp`	"1969-07-20"^^xsd:date "1969-07-20T20:17:40Z"^^xsd:dateTime

F.2.3 Property references

CQL2 allows the referencing of properties in a Feature Collection it is targeting for filtering. A property reference is converted to a triple pattern as shown in the following example. A SPARQL variable `?item` is assumed to represent the Feature Collection.

Property Reference	Triple pattern
name="OGC"	`?item my:name "OGC"^^xsd:string`
number=5	`?item my:number "5"^^xsd:integer`
number>5	`?item my:number ?number . FILTER(?number>5)`

F.2.4 Comparison Predicates

CQL2 defines comparison predicates to compare two scalar expressions. A comparison predicate is converted to a triple pattern as shown in the following example. A SPARQL variable `?item` is assumed to represent the Feature Collection.

Comparison predicate	Triple pattern	Comment
name="OGC"	`?item my:name "OGC"^^xsd:string`	Equality statements can be converted to a triple pattern
number=5	`?item my:number "5"^^xsd:integer`	
number>5	?item my:number ?number . FILTER(?number>5)	Arithmetic comparisons (<,>,>=,⇐) are converted to filter expressions

Comparison predicate	Triple pattern	Comment
number BETWEEN 5 AND 10	`?item my:number ?number . FILTER(?number>=5 && ?number<=10)`	BETWEEN statements are converted to arithmetic expressions
name IN ("OGC","W3C")	`?item my:name IN ("OGC", "W3C")`	IN statements may also be expressed using SPARQL VALUES statements
name IS NOT NULL	`EXISTS {?item my:name ?name }`	NOT NULL statements are converted to EXIST statements
name LIKE "OGC."	`?item my:name ?name . FILTER(regex(?name, "OGC.", "i" ))`	LIKE statements are converted to SPARQL regex filters
INTERSECTS(geometry1, geometry2)	`FILTER(geof:sfIntersects(?geometry1,?geometry2))`	The INTERSECTS filter statement is converted to a GeoSPARQL FILTER statement

There is no direct GeoSPARQL equivalent to a CRS-based CQL filter, however certain GeoSPARQL geometry literals have explicity CRS/SRS information that may be filtered using SPARQL `REGEX` operators.

F.2.5 Spatial Operators

GeoSPARQL includes equivalents of many CQL2 filter functions as can be seen in the table below.

CQL2 Filter Expression	GeoSPARQL Filter Function
CONTAINS(geometry1,geometry2)	FILTER(geof:sfContains(?geometry1,?geometry2))
CROSSES(geometry1,geometry2)	FILTER(geof:sfCrosses(?geometry1,?geometry2))
DISJOINT(geometry1,geometry2)	FILTER(geof:sfDisjoint(?geometry1,?geometry2))
EQUALS(geometry1,geometry2)	FILTER(geof:sfEquals(?geometry1,?geometry2))
INTERSECTS(geometry1,geometry2)	FILTER(geof:sfIntersects(?geometry1,?geometry2))
OVERLAPS(geometry1,geometry2)	FILTER(geof:sfOverlaps(?geometry1,?geometry2))
TOUCHES(geometry1,geometry2)	FILTER(geof:sfTouches(?geometry1,?geometry2))
WITHIN(geometry1,geometry2)	FILTER(geof:sfWithin(?geometry1,?geometry2))

F.2.6 Temporal Operators

Temporal operators are not part of the GeoSPARQL standard.

CQL2 Filter Expression	GeoSPARQL Filter Function
beginTime AFTER 1969-07-16T13:32:00Z	N/A
beginTime BEFORE 1969-07-16T13:32:00Z	N/A
beginTime BEGINS 1969-07-16T13:32:00Z	N/A
beginTime BEGUNBY 1969-07-16T13:32:00Z	N/A
beginTime DURING 1969-07-16T13:32:00Z	N/A
beginTime ENDEDBY 1969-07-16T13:32:00Z	N/A
beginTime ENDS 1969-07-16T13:32:00Z	N/A
beginTime MEETS 1969-07-16T13:32:00Z	N/A
beginTime METBY 1969-07-16T13:32:00Z	N/A
beginTime OVERLAPPEDBY 1969-07-16T13:32:00Z	N/A
beginTime TCONTAINS 1969-07-16T13:32:00Z	N/A
beginTime TEQUALS 1969-07-16T13:32:00Z	N/A
beginTime TOVERLAPS 1969-07-16T13:32:00Z	N/A

As noted above in Section F.2.1 Query Parameters, temporal filtering of RDF data via SPARQL queries is possible with standard SPARQL functions to compare date values (`xsd:date`, `xsd:dateTime` and `xsd:dateTimeStamp` literals) and OWL TIME [TIME] may be used to assert temporal relations between objects.

F.3 Mappings from Simple Features for SQL

The following table maps the functions and properties from Simple Features for SQL [ISO19125-1] to GeoSPARQL.

Simple Features for SQL	GeoSPARQL Equivalent	Since GeoSPARQL	Related Property Available	Since GeoSPARQL
2.1.1.1 Basic Methods on Geometry				
Dimension(): Double	`geof:dimension`	-	`geo:dimension`	1.0
GeometryType(): Integer	Class of geometry instance	1.0	N/A	-
SRID(): Integer	`geof:getSRID`	1.0	N/A	-
Envelope(): Geometry	`geof:envelope`	1.0	`geo:hasBoundingBox`	1.1
AsText(): String	`geof:asWKT`	1.1	`geo:asWKT`	1.0
AsBinary(): Binary	N/A	-	N/A	-
IsEmpty(): Integer	`geof:isEmpty`	-	`geo:isEmpty`	1.0
IsSimple(): Integer	`geof:isEmpty`	-	`geo:isSimple`	1.0
Boundary(): Geometry	`geof:boundary`	1.0	N/A	-
2.1.1.2 Spatial Relations				
Equals(anotherGeometry: Geometry): Integer	`geof:sfEquals`	1.0	`geo:sfEquals`	1.0
Disjoint(anotherGeometry: Geometry): Integer	`geof:sfDisjoint`	1.0	`geo:sfDisjoint`	1.0
Intersects(anotherGeometry: Geometry): Integer	`geof:sfIntersects`	1.0	`geo:sfIntersects`	1.0
Touches(anotherGeometry: Geometry): Integer	`geof:sfTouches`	1.0	`geo:sfTouches`	1.0

Simple Features for SQL	GeoSPARQL Equivalent	Since GeoSPARQL	Related Property Available	Since GeoSPARQL
Crosses(anotherGeometry: Geometry): Integer	`geof:sfCrosses`	1.0	`geo:sfCrosses`	1.0
Within(anotherGeometry: Geometry): Integer	`geof:sfWithin`	1.0	`geo:sfWithin`	1.0
Contains(anotherGeometry: Geometry): Integer	`geof:sfContains`	1.0	`geo:sfContains`	1.0
Overlaps(anotherGeometry: Geometry): Integer	`geof:sfOverlaps`	1.0	`geo:sfOverlaps`	1.0
Relate(anotherGeometry: Geometry, IntersectionPatternMatrix: String): Integer	`geof:relate`	1.0	N/A	-
2.1.1.3 Spatial Analysis				
Buffer(distance: Double): Geometry	`geof:buffer`	1.0	N/A	-
ConvexHull(): Geometry	`geof:convexHull`	1.0	N/A	-
Intersection(anotherGeometry: Geometry): Geometry	`geof:intersection`	1.0	N/A	-
Union(anotherGeometry: Geometry): Geometry	`geof:union`	1.0	N/A	-
Difference(anotherGeometry: Geometry): Geometry	`geof:difference`	1.0	N/A	-
SymDifference(anotherGeometry: Geometry): Geometry	`geof:symDifference`	1.0	N/A	-

Simple Features for SQL	GeoSPARQL Equivalent	Since GeoSPARQL	Related Property Available	Since GeoSPARQL
2.1.2.1 GeometryCollection				
NumGeometries(): Integer	`geof:numGeometries`	-	N/A	-
GeometryN(N: Integer): Geometry	`geof:geometryN`	-	N/A	-
2.1.3.1 Point				
X(): Double	N/A	-	N/A	-
Y(): Double	N/A	-	N/A	-
Z(): Double (not in the SQL spec, but a logical extension)	N/A	-	N/A	-
M(): Double (not in the SQL spec, but a logical extension)	N/A	-	N/A	-
2.1.5.1 Curve				
Length(): Double	`geof:length`	-	`geo:hasLength`	1.1
StartPoint(): Point	N/A	-	N/A	-
EndPoint(): Point	N/A	-	N/A	-
IsClosed(): Integer	N/A	-	N/A	-
IsRing(): Integer	N/A	-	N/A	-
2.1.6.1 LineString				
NumPoints(): Integer	N/A	-	N/A	-
PointN(N: Integer): Point	N/A	-	N/A	-
2.1.7.1 MultiCurve				
IsClosed(): Integer	N/A	-	N/A	-
Length(): Double	`geof:length`	-	`geo:hasLength`	1.1
2.1.9.1 Surface				
Area(): Double	`geof:area`	-	`geo:hasArea`	1.1
Centroid(): Point	`geof:centroid`	1.1	`geo:hasCentroid`	1.1
PointOnSurface(): Point	N/A	-	N/A	-
2.1.10.1 Polygon				

Simple Features for SQL	GeoSPARQL Equivalent	Since GeoSPARQL	Related Property Available	Since GeoSPARQL
ExteriorRing(): LineString	N/A	-	N/A	-
NumInteriorRing() : Integer	N/A	-	N/A	-
InteriorRingN(N: Integer): LineString	N/A	-	N/A	-
2.1.11.1 MultiSurface				
Area(): Double	`geof:area`	-	`geo:hasArea`	1.1
Centroid(): Point	`geof:centroid`	1.1	`geo:hasCentroid`	1.1
PointOnSurface(): Point	N/A	-	N/A	-

Bibliography

- Clementini, Eliseo; Di Felice, Paolino and van Oosterom, Peter, *A small set of formal topological relationships suitable for end-user interaction* (1993). In Abel, David; Ooi, Beng Chin (eds.). Advances in Spatial Databases: Third International Symposium, SSD '93 Singapore, June 23–25, 1993 Proceedings. Lecture Notes in Computer Science. 692/1993. Springer. pp. 277–295. doi:10.1007/3-540-56869-7_16
- [QUAL] Cohn, Anthony G.; Brandon Bennett; John Gooday; Nicholas Mark Gotts, *Qualitative Spatial Representation and Reasoning with the Region Connection Calculus*, GeoInformatica, 1, 275–316 (1997) https://doi.org/10.1023/A:1009712514511
- [FORMAL] Egenhofer, M. *A Formal Definition of Binary Topological Relationships* (1989). In: Litwin, W. and Schek, H.J., Eds., Proceedings of the 3rd International Conference on Foundations of Data Organization and Algorithms (FODO), Paris, France, Lecture Notes in Computer Science, 367, (Springer-Verlag, New York, 1989) 457-472. (1989)
- [CATEG] Egenhofer, Max and J. Herring *Categorizing Binary Topological Relations Between Regions, Lines, and Points* (1990). Geographic Databases, Technical Report, Department of Surveying Engineering, University of Maine
- [AUSPIX] Geoscience Australia, *AusPIX: An Australian Government implimentation of the rHEALPix DGGS in Python* (2020). https://github.com/GeoscienceAustralia/AusPIX_DGGS
- [ISO13249] International Organization for Standardization/International Electrotechnical Commission 13249-3, *ISO/IEC 13249-3: Information technology — Database languages — SQL multimedia and application packages — Part 3: Spatial* (2000)
- [ISO19105] International Organization for Standardization, *ISO 19105: Geographic information – Conformance and testing* (2000)
- [ISO19107] International Organization for Standardization, *ISO 19107: Geographic information — Spatial schema* (2003)
- [ISO19109] International Organization for Standardization, *ISO 19109: Geographic information — Rules for application schemas* (2005)
- [ISO19156] International Organization for Standardization, *ISO 19156: Geographic information — Observations and measurements* (2011) http://www.opengis.net/doc/is/om/2.0
- [IETF3987] Internet Engineering Task Force, *RFC 3987: Internationalized Resource Identifiers (IRIs)*. IETF Request for Comment (January 2005). https://tools.ietf.org/html/rfc3987
- [IETF5234] Internet Engineering Task Force, *RFC 5234: Internet Standard 68: Augmented BNF for Syntax Specifications: ABNF*. IETF Request for Comment (2008) https://tools.ietf.org/html/std68
- [GEOJSON] Internet Engineering Task Force, *RFC 7946: The GeoJSON Format*. IETF Request for Comment (August 2016). https://tools.ietf.org/html/rfc7946
- [OGC07-036] Open Geospatial Consortium, *OGC 07-036: Geography Markup Language (GML) Encoding Standard*, Version 3.2.1 (27 August 2007). https://portal.ogc.org/files/?artifact_id=20509
- [OGCKML] Open Geospatial Consortium, *OGC KML 2.3*. OGC Implementation Standard (04 August 2015). http://www.opengis.net/doc/IS/kml/2.3
- [OGCSFACA] Open Geospatial Consortium, *OGC 06-103r4* OpenGIS® Implementation Standard

for Geographic information - Simple feature access - Part 1: Common architecture. http://www.opengis.net/doc/is/sfa/1.2.1

- [CHARTER] Open Geospatial Consortium, *OGC GeoSPARQL SWG Charter*, OGC Working Group Charter (25 August 2020). https://github.com/opengeospatial/ogc-geosparql/blob/master/charter/swg_charter.pdf
- Open Geospatial Consortium, *OGC API - Features - Part 1: Core*. OGC Implementation Standard (14 October 2019). http://www.opengis.net/doc/IS/ogcapi-features-1/1.0
- [DGGSAS] Open Geospatial Consortium, *Abstract Standard Topic 21 - Discrete Global Grid Systems - Part 1 Core Reference system and Operations and Equal Area Earth Reference System*, Open Geospatial Consortium Standard (2021) http://www.opengis.net/doc/AS/dggs/2.0
- Open Geospatial Consortium, *OGC API - Features - Part 3: Filtering and the Common Query Language (CQL2)* (2021) http://www.opengis.net/doc/IS/ogcapi-features-3/1.0
- [WFS] Open Geospatial Consortium, *OpenGIS Web Feature Service 2.0 Interface Standard* (2010). OGC Implementation Standard (2 November 2010). http://www.opengeospatial.org/standards/wfs
- [LOGIC] Randell, D. A., Cui, Z. and Cohn, A. G.: A spatial logic based on regions and connection, Proc. 3rd Int. Conf. on Knowledge Representation and Reasoning, Morgan Kaufmann, San Mateo, pp. 165–176 (1992)
- [XSD1] World Wide Web Consortium, *XML Schema Part 1: Structures (Second Edition)*, W3C Recommendation (28 October 2004). https://www.w3.org/TR/xmlschema-1/
- [XSD2] World Wide Web Consortium, *XML Schema Part 2: Datatypes (Second Edition)*, W3C Recommendation (28 October 2004). https://www.w3.org/TR/xmlschema-2/
- [XML] World Wide Web Consortium, *Extensible Markup Language (XML) 1.0 (Fifth Edition)*, W3C Recommendation (26 November 2008). http://www.w3.org/TR/xml/
- [SKOS] World Wide Web Consortium, *SKOS Simple Knowledge Organization System Reference*, W3C Recommendation (18 August 2009). https://www.w3.org/TR/skos-reference/
- [XMLNS] World Wide Web Consortium, *Namespaces in XML 1.0 (Third Edition)*, W3C Recommendation (8 December 2009). https://www.w3.org/TR/xml-names/
- [OWL2] World Wide Web Consortium, *OWL 2 Web Ontology Language Document Overview (Second Edition)*, W3C Recommendation (11 December 2012). https://www.w3.org/TR/owl2-overview/
- [TIME] World Wide Web Consortium, *Time Ontology in OWL*, W3C Candidate Recommendation (26 March 2020). https://www.w3.org/TR/owl-time/
- [RIF] World Wide Web Consortium, *RIF Overview (Second Edition)*, W3C Working Group Note (5 February 2013). https://www.w3.org/TR/rif-overview/
- [RIFCORE] World Wide Web Consortium, *RIF Core Dialect (Second Edition)*, W3C Recommendation (5 February 2013) http://www.w3.org/TR/rif-core/
- [SPARQL] World Wide Web Consortium, *SPARQL 1.1 Query Language*, W3C Recommendation (21 March 2013). https://www.w3.org/TR/sparql11-query/
- [SPARQLENT] World Wide Web Consortium, *SPARQL 1.1 Entailment Regimes*, W3C Recommendation (21 March 2013). https://www.w3.org/TR/sparql11-entailment/

- [SPARQLPROT] World Wide Web Consortium, *SPARQL 1.1 Protocol*, W3C Recommendation (21 March 2013) http://www.w3.org/TR/sparql11-protocol/
- [SPARQLRESX] World Wide Web Consortium, *SPARQL Query Results XML Format (Second Edition)*, W3C Recommendation (21 March 2013). https://www.w3.org/TR/rdf-sparql-XMLres/
- [SPARQLRESJ] World Wide Web Consortium, *SPARQL 1.1 Query Results JSON Format*, W3C Recommendation (21 March 2013). http://www.w3.org/TR/sparql11-results-json/
- [TURTLE] World Wide Web Consortium, *RDF 1.1 Turtle - Terse RDF Triple Language*, W3C Recommendation (25 February 2014). https://www.w3.org/TR/turtle/
- [RDFXML] World Wide Web Consortium, *RDF 1.1 XML Syntax*, W3C Recommendation (25 February 2014). https://www.w3.org/TR/rdf-syntax-grammar/
- [RDF] World Wide Web Consortium, *RDF 1.1 Concepts and Abstract Syntax*, W3C Recommendation (25 February 2014). https://www.w3.org/TR/rdf11-concepts/
- [RDFSEM] World Wide Web Consortium, *RDF 1.1 Semantics*, W3C Recommendation (25 February 2014). https://www.w3.org/TR/rdf11-mt/
- [RDFS] World Wide Web Consortium, *RDF Schema 1.1*, W3C Recommendation (25 February 2014). https://www.w3.org/TR/rdf-schema/
- [SHACL] World Wide Web Consortium, *Shapes Constraint Language (SHACL)*, W3C Recommendation (20 July 2017). https://www.w3.org/TR/shacl/
- [PROF] World Wide Web Consortium, *The Profiles Vocabulary*, W3C Working Group Note (18 December 2019). https://www.w3.org/TR/dx-prof/
- [JSON-LD] World Wide Web Consortium, *JSON-LD 1.1: A JSON-based Serialization for Linked Data*, W3C Recommendation (16 July 2020). https://www.w3.org/TR/json-ld11/