OGC GeoSPARQL - A Geographic Query Language for RDF Data
# Table of Contents

1. Preface ............................................................................................................. 3
2. Submitting organizations ................................................................................. 4
   iii. Submission contact points ......................................................................... 4
3. Clarifications ...................................................................................................... 5
4. Changes to the OGC® Abstract Specification .................................................. 6
5. Foreword ........................................................................................................... 7
6. Introduction ....................................................................................................... 8
   RDF ..................................................................................................................... 8
   SPARQL ............................................................................................................... 8
   GeoSPARQL Standard structure ....................................................................... 9
7. OGC GeoSPARQL - A Geographic Query Language for RDF Data ................. 12
   1. Scope ............................................................................................................. 13
   2. Conformance ................................................................................................. 14
   3. Normative references .................................................................................. 16
   4. Terms and definitions .................................................................................. 17
      4.1. Semantic Web ......................................................................................... 17
      4.2. Spatial ..................................................................................................... 17
   5. Conventions ................................................................................................. 19
      5.1. Symbols and abbreviated terms ............................................................. 19
      5.2. Namespaces .......................................................................................... 19
      5.3. Placeholder IRIs .................................................................................... 20
      5.4. RDF Serializations ............................................................................... 20
   6. Core .............................................................................................................. 21
      6.1. SPARQL .................................................................................................. 21
      6.2. Classes .................................................................................................... 22
   7. Topology Vocabulary Extension .................................................................... 32
      7.1. Parameters .............................................................................................. 32
      7.2. Simple Features Relation Family ........................................................... 32
      7.3. Egenhofer Relation Family ................................................................... 33
      7.4. RCC8 Relation Family .......................................................................... 34
      7.5. Equivalent RCC8, Egenhofer and Simple Features Topological Relations
      8. Geometry Extension ..................................................................................... 36
      8.1. Rationale ................................................................................................. 36
      8.2. GeoSPARQL and Simple Features (SFA-CA) ........................................ 36
      8.3. Recommendation for units of measure ................................................... 37
      8.4. Influence of Reference Systems on computations ................................. 37
OGC GeoSPARQL - A Geographic Query Language for RDF Data

Copyright notice
Copyright © 2023 Open Geospatial Consortium
To obtain additional rights of use, visit http://www.opengeospatial.org/legal/

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
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 OGC GeoSPARQL Standard defines:

- A formal profile;
- this 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;
- SHACL shapes for RDF data validation.

This document 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 Standard to the Open Geospatial Consortium Inc.:

a. CSIRO
b. Cubewerx Inc.
c. Defence Science and Technology Laboratory (DSTL)
d. Geonovum
e. Geoscape Australia
f. Geoscience Australia
g. Mainz University Of Applied Sciences
h. Oracle America
i. OSGeo
j. SURROUND Australia Pty Ltd.

iii. Submission contact points

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

<table>
<thead>
<tr>
<th>Contact</th>
<th>Company</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simon J.D. Cox</td>
<td>CSIRO</td>
</tr>
<tr>
<td>Panagiotis (Peter) A. Vretanos</td>
<td>Cubewerx Inc.</td>
</tr>
<tr>
<td>Paul Cripps</td>
<td>DSTL</td>
</tr>
<tr>
<td>Linda van den Brink</td>
<td>Geonovum</td>
</tr>
<tr>
<td>Joseph Abhayaratna</td>
<td>Geoscape Australia</td>
</tr>
<tr>
<td>Irina Bastrakova</td>
<td>Geoscience Australia</td>
</tr>
<tr>
<td>Timo Homburg</td>
<td>Mainz University Of Applied Sciences</td>
</tr>
<tr>
<td>Matthew Perry</td>
<td>Oracle America</td>
</tr>
<tr>
<td>Dimitris Kotzinos</td>
<td>OSGeo</td>
</tr>
<tr>
<td>Nicholas J. Car</td>
<td>SURROUND Australia Pty Ltd.</td>
</tr>
</tbody>
</table>
Clarifications

• The terms Spatial Reference System (SRS) and Coordinate Reference System (CRS) are no longer interchangeable. Spatial Reference System is now taken to be a broader category than Coordinate Reference System. These are defined in the Clause 4 section.

• Class definitions were updated to be more self-contained and easier to understand for people without a background in geoinformatics. The definitions are no longer dependent on other standards’ definitions, only informed by them.

• A section was added on the specification of units of measurement.

• A section was added on the influence of Reference Systems on computations.
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 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 defines 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 Resource Description Framework (RDF) 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 elements (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 an Internationalized Resource Identifier (IRI) [IETF3987], which globally and uniquely identifies the resource referenced. IRIs are an extension to Uniform Resource Identifiers (URIs) that allow for non-ASCII characters. In addition to functioning as identifiers, IRIs are usually, but not necessarily, resolvable. This means a person or machine can "dereference" them (click on them or otherwise execute 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 represented with a blank node (a local identifier without meaning outside the graph it is defined within). Objects can further be represented with a literal value. RDF uses the basic literal values from XML [XSD2], however the basic types can be extended for specialized purposes. Indeed, this document extends the basic types to include geometry data. The figure below shows a basic triple.

![Figure 1. An RDF Triple](image)

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, there are similarities to the (feature-instance-by-id, attribute, value) tuples of the General Feature Model [ISO19109], and to the relational model as well (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.

From Wikipedia[^1]:

SPARQL (pronounced "sparkle", 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. Section 17.6[^1] of the SPARQL specification [SPARQL] 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. It also defines extensions to the SPARQL query language for processing geospatial data.

GeoSPARQL does not directly provide support for temporality. Predicates for temporal relations may be used from the OWL Time Ontology [TIME], but query extension functions for spatiotemporal operations are not present in the GeoSPARQL standard.

**GeoSPARQL Standard structure**

The GeoSPARQL Standard consists of multiple parts, or *profile resources*. The comprehensive listing of these parts is given in the GeoSPARQL *profile definition*, (see [http://www.opengis.net/def/geosparql](http://www.opengis.net/def/geosparql)). Below is an overview of the major parts:

1. *profile definition*
   - [http://www.opengis.net/def/geosparql](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. **Standard document (this document)**
   - [http://www.opengis.net/doc/IS/geosparql/1.1](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 [OGCSFACA] [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](http://www.opengis.net/ont/geosparql);
   - For geographic information representation;
   - Based on the General Feature Model [ISO19109], Simple Features Access [OGCSFACA] [ISO19125-1], Geography Markup Language [GML] and SQL MM [ISO13249]
   - Defined within the specification document and also delivered in RDF.

4. **Functions & Rules vocabulary**
   - [http://www.opengis.net/def/geosparql/funcsrules](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](http://www.opengis.net/ont/sf);
   - Derived from the class model defined in Simple Features Access [OGCSFACA] [ISO19125-1];
   - Presented as an [OWL2] ontology.

6. **[SPARQL] extension functions defined within this document.**

7. **RDF data validator**
   - [http://www.opengis.net/def/geosparql/validator](http://www.opengis.net/def/geosparql/validator);
   - Defined using [SHACL];
   - Presented within a single RDF file.

8. **SPARQL 1.1 Service description for GeoSPARQL**
   - [http://www.opengis.net/def/geosparql/servicedescription](http://www.opengis.net/def/geosparql/servicedescription);
   - Defined using [SPARQLSERVDESC].

This 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 relationships between spatial objects.
• A geometry component defining 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 relationship between two features into an equivalent query involving concrete geometries and topological query functions.

Each of these components forms a set of Requirements known as a GeoSPARQL Conformance Class. Implementations can provide various levels of functionality by choosing which Conformance Classes to support. For example, a system based purely on qualitative spatial reasoning may support only the core and topological vocabulary Classes.

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 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 relationships 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 relationships 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.


[2] https://www.w3.org/TR/sparql11-query/#extensionFunctions
OGC GeoSPARQL - A Geographic Query Language for RDF Data
Chapter 1. Scope

The OGC GeoSPARQL Standard is comprised of 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. Instead GeoSPARQL 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 Geospatial communities to develop additional vocabularies for describing spatial information.
Chapter 2. Conformance

Conformance with this Standard shall be checked using all the relevant tests specified in Annex A - Abstract Test Suite (normative). 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 many individual Requirements and Conformance Classes which contain tests for one or more Requirements. GeoSPARQL implementations need not conform to all Conformance Classes but must state which individual ones they do conform to. GeoSPARQL implementations claiming conformance to a Conformance Class must pass all the tests defined for it in Annex A - Abstract Test Suite (normative).

Requirements and Conformance Class tests have IRIs that are relative to versioned namespace IRIs. Requirements and Conformance Class tests that are defined in GeoSPARQL 1.0 have IRIs relative to http://www.opengis.net/spec/geosparql/1.0/ and those added in GeoSPARQL 1.1 have IRIs relative to http://www.opengis.net/spec/geosparql/1.1/.

Many Conformance Classes are parameterized, and any parameters are explained in the detailed clauses for those Conformance Classes.

Table 1. Conformance Classes

<table>
<thead>
<tr>
<th>Conformance Class</th>
<th>Description</th>
<th>Subclause of the abstract test suite</th>
</tr>
</thead>
<tbody>
<tr>
<td>Core</td>
<td>Defines top-level spatial vocabulary components</td>
<td>A.1</td>
</tr>
<tr>
<td>Topology Vocabulary Extension</td>
<td>Defines topological relation vocabulary</td>
<td>A.2</td>
</tr>
<tr>
<td>Geometry Extension</td>
<td>Defines geometry vocabulary and non-topological query functions</td>
<td>A.3</td>
</tr>
<tr>
<td>Geometry Extension - DGGS</td>
<td>Defines the properties and functions of the Geometry Extension Conformance Classes for use with Discrete Global Grid System geometry representations</td>
<td>A.3.DGGS</td>
</tr>
<tr>
<td>Geometry Topology Extension</td>
<td>Defines topological query functions for geometry objects</td>
<td>A.4</td>
</tr>
<tr>
<td>RDFS Entailment Extension</td>
<td>Defines a mechanism for matching implicit RDF triples that are derived based on RDF and RDFS semantics</td>
<td>A.5</td>
</tr>
<tr>
<td>Conformance Class</td>
<td>Description</td>
<td>Subclause of the abstract test suite</td>
</tr>
<tr>
<td>---------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>-------------------------------------</td>
</tr>
<tr>
<td>Query Rewrite Extension</td>
<td>Defines query transformation rules for computing spatial relations between spatial objects based on their associated geometries</td>
<td>A.6</td>
</tr>
</tbody>
</table>

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

*Figure 2. Conformance Class Dependency Graph*
Chapter 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 linked to their full citation in the Bibliography.

- [OGCSFACA] [ISO19125-1], ISO 19125-1: Geographic information — Simple feature access — Part 1: Common architecture
- [OGCOM] [ISO19156], ISO 19156: Geographic information — Observations and measurements
- [GML], OGC 07-036: Geography Markup Language (GML) Encoding Standard
- [IETF3987], Internet Engineering Task Force, RFC 3987: Internationalized Resource Identifiers (IRIs)
- [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
Chapter 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 neighbor 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).
Chapter 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)
- **GIS**: Geographic Information System
- **GML**: Geography Markup Language
- **IRI**: Internationalized Resource Identifier
- **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
- **SQL**: Structured Query Language
- **SRS**: Spatial Reference System
- **URI**: Universal Resource Identifier
- **WKT**: Well Known Text (as defined by Simple Features or ISO 19125)
- **W3C**: World Wide Web Consortium (https://www.w3.org)
- **XML**: Extensible Markup Language

5.2. Namespaces

The following IRI namespace prefixes are used throughout this document:

- **ex**: http://example.com/ A non-resolving namespace for examples
- **geo**: http://www.opengis.net/ont/geosparql# GeoSPARQL Ontology
- **geof**: http://www.opengis.net/def/function/geosparql/ GeoSPARQL Functions
- **geor**: http://www.opengis.net/def/rule/geosparql/ GeoSPARQL Rules
- **gml**: http://www.opengis.net/ont/gml# [GML] vocabulary
5.3. Placeholder IRIs

All of these namespace prefixes in the previous section resolve to resources that contain their namespace content except for `ex:` (http://example.com/), which is used just for examples, and `ogc:` (http://www.opengis.net/), which is used as a placeholder, for example, `ogc:geomLiteral` is used to indicate any one of the specific geometry literal serializations defined here, such as `geo:wktLiteral`.

5.4. RDF Serializations

Three RDF serializations are used in this document. Terse RDF Triple Language `turtle` is used for RDF snippets placed within the main body of the document, and `turtle`, `JSON-LD` & `RDF/XML` are used for the examples in Annex B — GeoSPARQL Examples.
Chapter 6. Core

This clause establishes the Core Requirements class, with IRI /req/core, which has a corresponding Conformance Class, Core, with IRI /conf/core. These Requirements define 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. The RDFS and OWL vocabularies 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 gives an overview of the classes and properties defined by GeoSPARQL in the Core, Topology Vocabulary Extension and Geometry Extension, Geometry Topology Extension and RDFS Entailment Extension Conformance Classes.

![An overview of the Classes and Properties defined in GeoSPARQL. Where specific Classes and Properties are indicated, the 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.](image)

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.

A.1.1.1 /conf/core/sparql-protocol
6.2. Classes

Two main classes are defined: `geo:SpatialObject` and `geo:Feature`.

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:

``` reason
geo:SpatialObject
  a rdfs:Class, owl:Class ;
  rdfs:isDefinedBy geo ;
  skos:prefLabel "Spatial Object"@en ;
  skos:definition "Anything spatial (being or having a shape, position or an extent)."@en ;
  skos:note "Subclasses of this class are expected to be used for instance data."@en ;
.
```

Req 2 Implementations shall allow the RDFS class `geo:SpatialObject` to be used in SPARQL graph patterns.

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

Example:

``` reason
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` [OGCOM] [ISO19156] and is defined by the following:

``` reason
geo:Feature
  a rdfs:Class, owl:Class ;
  rdfs:isDefinedBy geo ;
  skos:prefLabel "Feature"@en ;
  rdfs:subClassOf geo:SpatialObject ;
  owl:disjointWith geo:Geometry ;
  skos:definition "A discrete spatial phenomenon in a universe of discourse."@en ;
  skos:note "A Feature represents a uniquely identifiable phenomenon, for example a river or an apple. While such phenomena (and therefore the Features used to represent them) are bounded, their boundaries may be crisp
```
(e.g., the declared boundaries of a state), vague (e.g., the
delineation of a valley versus its neighboring mountains), and change
with time (e.g., a storm front). While discrete in nature, Features
may be created from continuous observations, such as an isochrone
that determines the region that can be reached by ambulance within
5 minutes."@en 

Req 3 Implementations shall allow the RDFS class geo:Feature to be used in SPARQL graph
patterns.

A.1.2.1 /conf/core/spatial-object-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 "A collection of individual Spatial Objects."@en ;
  skos:note "This is the 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 ;
  ] ;
.

Membership of the generic rdfs:Container that defines this class is restricted to instances of Spatial
Object. Spatial Object Collection members are to be indicated with the rdfs:member property.

Req 4 Implementations shall allow the RDFS class geo:SpatialObjectCollection to be used in
SPARQL graph patterns.

A.1.2.3 /conf/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" ;
Membership of the more general Spatial Object Collection that defines this class is restricted to instances of Feature. geo:FeatureCollection members are to be indicated with the rdfs:member property.

**Req 5** Implementations shall allow the RDFS class geo:FeatureCollection to be used in SPARQL graph patterns.

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

### 6.3. Standard Properties for geo:SpatialObject

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

**Req 6** 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.1.2.5 /conf/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) cannot be used. If it is possible to express size in metric units, subproperties of geo:hasMetricSize should be used. This property has no range specification. This makes it possible to use other vocabularies for expressions of size, for example vocabularies for units of measurement or vocabularies for specifying measurement quality.

GeoSPARQL 1.1 defines the following subproperties of this property: geo:hasLength, geo:hasPerimeterLength, 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 cannot 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.

---

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.

---

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.
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.

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.

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:
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.

6.3.8. Property: geo:hasMetricArea

The property geo:hasMetricArea can be used to indicate the area of a Spatial Object in square meters (m²). It is a subproperty of geo:hasMetricSize. This property can be used for Spatial Objects having two or three dimensions.

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:

```sparql
PREFIX geo: <http://www.opengis.net/ont/geosparql#>
ASK
WHERE {
    ?g geo:hasMetricPerimeterLength ?p ;
    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 (m³). 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:

```sparql
PREFIX geo: <http://www.opengis.net/ont/geosparql#>
ASK
WHERE {
  ?g geo:hasMetricArea ?a ;
  geo:dimension ?d .
  FILTER (?d < 2)
}
```
6.4. Standard Properties for geo:Feature

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

**Req 7** Implementations shall allow the properties geo:hasGeometry, geo:hasDefaultGeometry, geo:hasCentroid and geo:hasBoundingBox to be used in SPARQL graph patterns.

A.3.1.3 /conf/core/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.

```xml
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).

```xml
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 the feature's geometry. Bounding-boxes are typically used 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
    exactly aligned with the axes of the coordinate reference system, which 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.

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.
Chapter 7. Topology Vocabulary Extension

This clause establishes the *Topology Vocabulary Extension* parameterized Requirements class. The IRI base is `/req/topology-vocab-extension`, which has a single corresponding Conformance Class *Topology Vocabulary Extension*, 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, such as RCC8 and 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 to 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 [OGCSFACA] [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.

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

7.2. Simple Features Relation Family

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

<table>
<thead>
<tr>
<th>Relation Name</th>
<th>Relation IRI</th>
<th>Domain/Range</th>
<th>Applies To Geometry Types</th>
<th>DE-9IM Intersection Pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td>equals</td>
<td>geo:sfEquals</td>
<td>geo:SpatialObject</td>
<td>All</td>
<td>(TFFFFFTFFT)</td>
</tr>
<tr>
<td>disjoint</td>
<td>geo:sfDisjoint</td>
<td>geo:SpatialObject</td>
<td>All</td>
<td>(FF**FF****)</td>
</tr>
</tbody>
</table>
### 7.3. Egenhofer Relation Family

This clause defines Requirements for the 9-intersection model for the binary topological relations (Egenhofer) relation family. The reader should consult references [FORMAL] and [CATEG] for a more detailed discussion of Egenhofer relations.

**Req 9** 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.2.2.1 /conf/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

<table>
<thead>
<tr>
<th>Relation Name</th>
<th>Relation IRI</th>
<th>Domain/Range</th>
<th>Applies To Geometry Types</th>
<th>DE-9IM Intersection Pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td>intersects</td>
<td>geo:sfIntersects</td>
<td>geo:SpatialObject</td>
<td>All</td>
<td>(T********)<em>T</em>******</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td><em><strong>T</strong></em>****</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td><strong><strong>T</strong></strong></td>
</tr>
<tr>
<td>touches</td>
<td>geo:sfTouches</td>
<td>geo:SpatialObject</td>
<td>All except P/P</td>
<td>(FT********)<em>F**T</em>******</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>F*<strong>T</strong>**</td>
</tr>
<tr>
<td>within</td>
<td>geo:sfWithin</td>
<td>geo:SpatialObject</td>
<td>All</td>
<td>(T<em>F</em><strong>F</strong>*</td>
</tr>
<tr>
<td>contains</td>
<td>geo:sfContains</td>
<td>geo:SpatialObject</td>
<td>All</td>
<td>(T***<em><strong>F</strong></em>)</td>
</tr>
<tr>
<td>overlaps</td>
<td>geo:sfOverlaps</td>
<td>geo:SpatialObject</td>
<td>A/A, P/P, L/L</td>
<td>(T<em>T</em>*<strong>T</strong>) for A/A, P/P;</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(1<em>T</em>*<strong>T</strong>) for L/L</td>
</tr>
<tr>
<td>crosses</td>
<td>geo:sfCrosses</td>
<td>geo:SpatialObject</td>
<td>P/L, P/A, L/A, L/L</td>
<td>(T****T**) for P/L, P/A, L/A;</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(0********) for L/L</td>
</tr>
</tbody>
</table>

---

[FORMAL] [CATEG]
<table>
<thead>
<tr>
<th>Relation Name</th>
<th>Relation IRI</th>
<th>Domain/Range</th>
<th>Applies To</th>
<th>DE-9IM Intersection Pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td>covers</td>
<td>geo:ehCovers</td>
<td>geo: SpatialObject</td>
<td>A/A, A/L, L/L</td>
<td>(T<em>TFT</em>FF*)</td>
</tr>
<tr>
<td>covered by</td>
<td>geo:ehCoveredBy</td>
<td>geo: SpatialObject</td>
<td>A/A, L/A, L/L</td>
<td>(TFF*TFT**)</td>
</tr>
<tr>
<td>inside</td>
<td>geo:ehInside</td>
<td>geo: SpatialObject</td>
<td>All</td>
<td>(TFF*FFT**)</td>
</tr>
<tr>
<td>contains</td>
<td>geo:ehContains</td>
<td>geo: SpatialObject</td>
<td>All</td>
<td>(T<em>TFF</em>FF*)</td>
</tr>
</tbody>
</table>

### 7.4. RCC8 Relation Family

This clause defines Requirements for the region connection calculus basic 8 (RCC8) relation family. The reader should consult references [QUAL] and [LOGIC] for a more detailed discussion of RCC8 relations.

**Req 10** 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.2.3.1 /conf/topology-vocab-extension/rcc8-spatial-relations

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

**Table 4. RCC8 Topological Relations**

<table>
<thead>
<tr>
<th>Relation Name</th>
<th>Relation IRI</th>
<th>Domain/Range</th>
<th>Applies To</th>
<th>DE-9IM Intersection Pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td>equals</td>
<td>geo:rcc8eq</td>
<td>geo: SpatialObject</td>
<td>A/A</td>
<td>(TFFFTFFFFT)</td>
</tr>
<tr>
<td>disconnected</td>
<td>geo:rcc8dc</td>
<td>geo: SpatialObject</td>
<td>A/A</td>
<td>(FFTFFTTTTT)</td>
</tr>
<tr>
<td>externally connected</td>
<td>geo:rcc8ec</td>
<td>geo: SpatialObject</td>
<td>A/A</td>
<td>(FFTFTTTTTT)</td>
</tr>
<tr>
<td>partially overlapping</td>
<td>geo:rcc8po</td>
<td>geo: SpatialObject</td>
<td>A/A</td>
<td>(TTTTTTTTTT)</td>
</tr>
<tr>
<td>tangential proper part</td>
<td>geo:rcc8tppi</td>
<td>geo: SpatialObject</td>
<td>A/A</td>
<td>(TTTFTTTFFT)</td>
</tr>
<tr>
<td>tangential proper part</td>
<td>geo:rcc8tpp</td>
<td>geo: SpatialObject</td>
<td>A/A</td>
<td>(TFFFTTTTTT)</td>
</tr>
<tr>
<td>non-tangential proper part</td>
<td>geo:rcc8ntpp</td>
<td>geo: SpatialObject</td>
<td>A/A</td>
<td>(TFFFTTTTTT)</td>
</tr>
<tr>
<td>non-tangential proper part</td>
<td>geo:rcc8ntppi</td>
<td>geo: SpatialObject</td>
<td>A/A</td>
<td>(TTTFTTTFFT)</td>
</tr>
</tbody>
</table>
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 $\neg$ denotes negation.

*Table 5. Equivalent Simple Features, RCC8 and Egenhofer relations*

<table>
<thead>
<tr>
<th>Simple Features</th>
<th>RCC8</th>
<th>Egenhofer</th>
</tr>
</thead>
<tbody>
<tr>
<td>equals</td>
<td>equals</td>
<td>equals</td>
</tr>
<tr>
<td>disjoint</td>
<td>disconnected</td>
<td>disjoint</td>
</tr>
<tr>
<td>intersects</td>
<td>$\neg$ disconnected</td>
<td>$\neg$ disjoint</td>
</tr>
<tr>
<td>touches</td>
<td>externally connected</td>
<td>meet</td>
</tr>
<tr>
<td>within</td>
<td>non-tangential proper part</td>
<td>inside $+$ coveredBy</td>
</tr>
<tr>
<td></td>
<td>tangential proper part</td>
<td></td>
</tr>
<tr>
<td>contains</td>
<td>non-tangential proper part</td>
<td>contains $+$ covers</td>
</tr>
<tr>
<td></td>
<td>inverse $+$ tangential proper part inverse</td>
<td></td>
</tr>
<tr>
<td>overlaps</td>
<td>partially overlapping</td>
<td>overlap</td>
</tr>
</tbody>
</table>
Chapter 8. Geometry Extension

This clause defines the Geometry Extension parameterized Requirements class with the base IRI /req/geometry-extension. There is a single corresponding conformance class Geometry Extension, with the IRI /conf/geometry-extension. These Requirements define a vocabulary for asserting and querying information about geometry data, and define 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\(^1\) was an early (2003) RDF vocabulary for "representing lat(itude), long(itude) and other information about spatially-located things. Geo specifies WGS84 as the reference datum". Further, many widely used Semantic Web vocabularies contain some spatial data support. For example, Dublin Core Terms provides a Location class\(^2\) for "A spatial region or named place." and schema.org provides a number of spatial object and geometry classes, such as GeoCoordinates\(^3\) and GeoShape\(^4\).

Many vocabularies such as the above provide little specific support for detailed geometries and only specify using the WGS84 Coordinate Reference System (CRS).

Since the first version of GeoSPARQL, many ontologies have imported GeoSPARQL. For example, the ISA Programme Location Core Vocabulary\(^5\) 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\(^6\) 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 dcat: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 ISO/OGC Simple Features Access - Common Architecture (SFA-CA) Standard [OGCSFACA]. Contrary to what the name may imply, SFA-CA is about Geometry and not about Features. SFA-CA describes simple geometry, meaning that geometric shapes are based on points and straight lines (linear 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 approach 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 SWG 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 either the OGC KML [OGCKML] or the GeoJSON format. For example, neither KML nor GeoJSON support the Triangulated Integrated Network (TIN) or Triangle geometry types.

### 8.3. Recommendation for units of measure

For geometric data to be interpreted and used correctly, the units of measure should be known. Typically, the particular Spatial Reference System (SRS) 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. Making the unit of measurement explicit will improve data interoperability. The recommended vocabulary for units of measurement for GeoSPARQL is the Quantities, Units, Dimensions and Types (QUDT) ontology, but others may be used, as long as they are well-described.

### 8.4. Influence of Reference Systems on computations

A Geometry object consists of a set of coordinates and a specification on how the coordinates should be interpreted. This specification is known as a Spatial reference System (SRS). Taken together, coordinates and SRS allow performing computations on Geometry objects. For example, sizes can be calculated or new Geometry objects can be created. Some Spatial 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 Spatial 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 decimal 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 or other SRS.

To avoid erroneous computations involving Geometry, data publishers are recommended to clearly indicate the type of space that is described by the SRS.
8.5. Parameters

The following parameters are defined for the Geometry Extension Requirements.

**serialization**

Specifies the serialization standard to use when generating geometry literals as well as the supported geometry types.

\[\text{NOTE}\]

A serialization strongly affects the geometry conceptualization. The WKT serialization aligns the geometry types with ISO 19125 Simple Features [OGCSFACA] [ISO19125-1]; 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 conceptually derived from UML class Geometry in [ISO19107] which is that standard's "root class of the geometric object taxonomy and supports interfaces common to all geographically referenced geometric objects". geo:Geometry is defined by the following:

```xml
geo:Geometry
   a rdfs:Class, owl:Class ;
   rdfs:isDefinedBy geo ;
   skos:prefLabel "Geometry"@en ;
   rdfs:subClassOf geo: SpatialObject ;
   owl:disjointWith geo:Feature ;
   skos:definition "A coherent set of direct positions in space. The positions are held within a Spatial Reference System (SRS)."@en ;
   skos:note "Geometry can be used as a representation of the shape, extent or location of a Feature and may exist as a self-contained entity."@en ;
```

\[\text{Req 11}\]

Implementations shall allow the RDFS class geo:Geometry to be used in SPARQL graph patterns. A.3.1.1 /conf/geometry-extension/geometry-class

8.6.2. Class: geo:GeometryCollection

The class Geometry Collection is defined by the following:
Membership of the general Spatial Object Collection that defines this class is restricted to instances of Geometry. geo:GeometryCollection members are to be indicated with the rdfs:member property.

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. Both the OGC Geography Markup Language (GML) and KML use a MultiGeometry type and Well Known Text (WKT) and GeoJSON use a GeometryCollection type. While the names of some of these objects are 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 12 Implementations shall allow the RDFS class geo:GeometryCollection to be used in SPARQL graph patterns.

A.3.1.2 /conf/geometry-extension/geometry-collection-class

8.7. Standard Properties for geo:Geometry

Properties are defined for describing geometry metadata.
Req 13 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.

A.3.1.4 /conf/geometry-extension/geometry-properties

8.7.1. Property: geo:dimension

The property geo:dimension is used to link 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
```

40
8.7.4. Property: geo:hasSpatialResolution

The property `geo:hasSpatialResolution` is defined to indicate the spatial resolution of the elements within a Geometry. Spatial resolution specifies the level of detail of a Geometry. It is the smallest distinguishable distance between adjacent coordinate sets. This property is not applicable to a point Geometry, because a point 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.

8.7.5. Property: geo:hasMetricSpatialResolution

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

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 truthfulness of the positions (coordinates) that define the Geometry. In this case accuracy defines a zone surrounding each coordinate within which 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).

---

**8.7.7. Property: geo:hasMetricSpatialAccuracy**

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

---

**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.
8.7.9. Property: geo:isSimple

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

```rdf
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).

```rdf
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 property 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 class 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. It 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) [OGCSFACA] [ISO19125-1] standard's conceptualization of geometry which defines classes such as Point, Curve and Surface and specialized 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 representation systems given here do not use the same terminology as SFA, in particular Discrete Global Grid Systems. 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

This section establishes the requirements for representing Geometry data in RDF based on Well-Known Text (WKT) as defined by Simple Features Access [OGCSFACA] [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 14 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], enclosed in angled brackets (< & >) followed by whitespace as a separator, and a WKT string as defined in [ISO13249].

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

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

```
wktLiteral ::= opt-iri-and-whitespace geometry-data
opt-iri-and-space = "<" IRI ">" LWSP / ""
```

The token opt-iri-and-whitespace may be either an IRI and whitespace (spaces, tabs, newlines) 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]. geometry-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 15** 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.3.2.2 /conf/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 16** Coordinate tuples within geo:wktLiteral shall be interpreted using the axis order defined in the spatial reference system used.

A.3.2.3 /conf/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/crs/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 17** An empty RDFS Literal of type geo:wktLiteral shall be interpreted as an empty Geometry.

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

### 8.8.2. Property: asWKT

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

**Req 18** Implementations shall allow the RDF property geo:asWKT to be used in SPARQL graph patterns.

A.3.2.5 /conf/geometry-extension/geometry-as-wkt-literal
8.8.2.1. Function: geof:asWKT

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

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

Req 19 Implementations shall support geof:asWKT as a SPARQL extension function.

A.3.2.6 /conf/geometry-extension/asWKT-function

8.8.3. Geography Markup Language

This section establishes a Requirements class for representing Geometry data in RDF based on GML as defined by the Geography Markup Language Encoding Standard [GML]. It defines one RDFS Datatype: GML Literal and one property, as GML.

8.8.3.1. 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 20 All `geo:gmlLiteral` instances shall consist of a valid element from the GML schema that implements a subtype of `GM_Object` as defined in [GML].

A.3.3.1 /conf/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:

```xml
<gml:Point
    srsName="http://www.opengis.net/def/crs/OGC/1.3/CRS84"
    xmlns:gml="http://www.opengis.net/gml/3.2">
    <gml:pos>-83.38 33.95</gml:pos>
</gml:Point>
```

Req 21 An empty `geo:gmlLiteral` shall be interpreted as an empty Geometry.

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

Req 22 Implementations shall document supported GML profiles.

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

8.8.4. Property: asGML

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

Req 23 Implementations shall allow the RDF property `geo:asGML` to be used in SPARQL graph patterns.

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

```xml
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 ;
.
```

8.8.4.1. Function: geof:asGML

```xml
```

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

**Req 24** Implementations shall support `geof:asGML` as a SPARQL extension function.

A.3.3.5 /conf/geometry-extension/asGML-function

### 8.8.5. GeoJSON

This section establishes a Requirements class for representing Geometry data in RDF based on Geographic JavaScript Object Notation (GeoJSON) as defined by Section 8.8.5. It defines one RDFS Datatype: GeoJSON Literal and one property, as GeoJSON.

#### 8.8.5.1. RDFS Datatype: `geo:geoJSONLiteral`

The datatype `geo:geoJSONLiteral` is used to contain the GeoJSON serialization of a Geometry.

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

**Req 25** All `geo:geoJSONLiteral` instances shall consist of the Geometry objects as defined in the GeoJSON specification [GEOJSON].

A.3.4.1 /conf/geometry-extension/geojson-literal

**Req 26** 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).

A.3.4.2 /conf/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:

```json
"{"type": "Point", "coordinates": [-83.38,33.95]}
```

```
"^^http://www.opengis.net/ont/geosparql#geoJSONLiteral>
```

**Req 27** An empty RDFS Literal of type `geo:geoJSONLiteral` shall be interpreted as an empty Geometry, i.e. `{"geometry": null}` in GeoJSON.

A.3.4.3 /conf/geometry-extension/geojson-literal-empty
8.8.6. Property: asGeoJSON

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

**Req 28** Implementations shall allow the RDF property `geo:asGeoJSON` to be used in SPARQL graph patterns.

A.3.4.4 /conf/geometry-extension/geometry-as-geojson-literal

```rml
geo:asGeoJSON
  a rdfs: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 ;
.
```

8.8.6.1. 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 29** Implementations shall support `geof:asGeoJSON` as a SPARQL extension function.

A.3.4.5 /conf/geometry-extension/asGeoJSON-function

8.8.7. Keyhole Markup Language

This section establishes the Requirements class 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`.

8.8.7.1. 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 30** All geo:kmlLiteral instances shall consist of the Geometry objects as defined in the KML specification [OGCKML].

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

**Req 31** 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).

A.3.5.2 /conf/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:

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

**Req 32** An empty RDFS Literal of type geo:kmlLiteral shall be interpreted as an empty Geometry.

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

### 8.8.8. Property: asKML

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

**Req 33** Implementations shall allow the RDF property geo:asKML to be used in SPARQL graph patterns.

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

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

```ttl
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 ;
 .
```

8.8.8.1. Function: geof:asKML

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 34** Implementations shall support `geof:asKML` as a SPARQL extension function.

A.3.5.5 /conf/geometry-extension/asKML-function

8.8.9. Discrete Global Grid System

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

The Requirements class defines one RDFS Datatype `http://www.opengis.net/ont/geosparql#dggsLiteral` and one property, `http://www.opengis.net/ont/geosparql#asDGGS`.

8.8.9.1. RDFS Datatype: geo:dggsLiteral

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

```xml
geo:dggsLiteral
  a rdfs:Datatype ;
  rdfs:isDefinedBy geo ;
  skos:prefLabel "DGGS Literal"@en ;
```

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

**Req 35** All RDFS Literals of type `geo:dggsLiteral` shall consist of an IRI identifying the specific DGGS and a representation of the DGGS geometry data. The IRI shall be enclosed in angled brackets (`< & >`) followed by whitespace as a separator, and then the DGGS geometry data, formulated according to the identified DGGS.

A.4.2.1 /conf/geometry-extension-dggs/dggs-literal

The following ABNF [IETF5234] syntax specification formally defines this literal:
The token `iri-and-whitespace` is an IRI and whitespace. The token `IRI` (Internationalized Resource Identifier) is essentially a web address and is defined in [IETF3987]. The token `LWSP` is one or more whitespace characters, as defined in [IETF5234]. `dggs-geometry-data` is geometry data formulated according to the DGGS identified by `IRI`.

An example of a DGGS literal for the AusPIX DGGS could be:

```
"<https://w3id.org/dggs/auspix> CELL (R3234)"^^geo:dggsLiteral
```

Where AusPIX is identified with the IRI `https://w3id.org/dggs/auspix` and `CELL (R3234)` is the representation of a geometry according to AusPIX.

**NOTE**

What `R3234` means, or the meaning of any other element within a DGGS' geometry data is not handled by GeoSPARQL, just as GeoPSARQL does not delve into the internals of other Geometry formats such as WKT or GeoJSON.

**Req 36** An empty RDFS Literal of type `geo:dggsLiteral`, shall be interpreted as an empty `geo:Geometry`.

**A.4.2.2 /conf/geometry-extension-dggs/dggs-literal-empty**

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

```
dggsLiteral ::= iri-and-space dggs-geometry-data
iri-and-space = "<" IRI ">" LWSP / ""
```

The tokens used above are as per the DGGS `ABNF` above.

**8.8.10. Property: geo:asDGGS**

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

**Req 37** Implementations shall allow the RDF property `geo:asDGGS` to be used in SPARQL graph patterns.

**A.4.2.3 /conf/geometry-extension-dggs/geometry-as-dggs-literal**

```
geo:asDGGS
  a rdf:Property, owl:DatatypeProperty;
  rdfs:subPropertyOf geo:hasSerialization;
  rdfs:isDefinedBy geo:;
```
8.8.10.1. Function: geof:asDGGS


The function geof:asDGGS converts geom to an equivalent DGGS representation, formulated according to the specific DGGS literal indicated by the IRI required to be present in the DGGS literal.

Req 38 Implementations shall support geof:asDGGS as a SPARQL extension function.

A.4.2.4 /conf/geometry-extension-dggs/asDGGS-function

8.9. Non-topological Query Functions

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


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

Req 40 Implementations shall support the functions geof:metricLength, geof:length, geof:metricPerimeter, geof:perimeter, geof:metricArea, geof:area, geof:geometryN, geof:maxX, geof:maxY, geof:maxZ, geof:minX, geof:minY, geof:minZ and geof:numGeometries as SPARQL extension functions which are defined in this standard, for non-DGGS geometry literals.

A.3.1.6 /conf/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 41 Implementations shall support the functions of Requirement 39 for DGGS geometry literals as SPARQL extension functions, in a manner which is consistent with definitions of these functions in Simple Features [OGCSFACA] [ISO19125-1], for non-DGGS geometry literals.

A.4.1.1 /conf/geometry-extension-dggs/query-functions

Req 42 Implementations shall support the functions of Requirement 40 for DGGS geometry literals as SPARQL extension functions which are defined in this standard, for non-DGGS geometry literals.

A.4.1.2 /conf/geometry-extension-dggs/query-functions-non-sf

Functions from this Requirements class are listed below, alphabetically.

8.9.1. Function notes

These notes apply to all 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
- A 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 unit 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 SRS.

For further discussion of the effects of errors during FILTER evaluation, consult Section 17[8] 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[9] of the SPARQL specification [SPARQL]:

54
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 FILTERs 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. See the Recommendation for specification of units of measurement. Also, the OGC has recommended units of measure vocabularies for use, see the OGC Definitions Server[10].

8.9.2. Function: geof:metricArea

geof:metricArea (geom: ogc:geomLiteral): xsd:double

The function geof:metricArea returns the area of geom in square meters. Must return zero for all geometry types other than Polygon. This function is similar to geof:area but does not need a specification of measurement unit.

8.9.3. Function: geof:area


The function geof:area returns the area of geom. Must return zero for all geometry types other than Polygon. This function is similar to geof:metricArea, which does not need a specification of measurement unit.

NOTE  See the Recommendation for specification of units of measurement.

8.9.4. Function: geof:boundary


The function geof:boundary returns the closure of the boundary of geom. Calculations are in the spatial reference system of geom.
8.9.5. Function: geof:boundingCircle

geof:boundingCircle (geom: ogc:geomLiteral): ogc:geomLiteral

The function geof:boundingCircle returns the minimum bounding circle around geom. Calculations are in the spatial reference system of geom.

8.9.6. Function: geof:metricBuffer


The function geof:metricBuffer 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 measurement unit.

8.9.7. Function: geof:buffer


The function geof:buffer 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 measurement unit.

NOTE See the Recommendation for specification of units of measurement.

8.9.8. Function: geof:centroid

geof:centroid (geom: ogc:geomLiteral): ogc:geomLiteral

The function geof:centroid returns the mathematical centroid of geom. The centroid point does not have to be part of the surface it is derived from.

8.9.9. Function: geof:convexHull


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

geof:concaveHull (geom: ogc:geomLiteral): ogc:geomLiteral

The function geof:concaveHull returns a geometric object that represents all Points in the concave hull of geom. Calculations are in the spatial reference system of geom. Various implementers use parameters to calculate a concave hull. As such, two implementations may return different results from their concave hull functions for the same geometry. Implementers should make clear any default values used to calculate a concave hull in their documentation.

8.9.11. Function: geof:coordinateDimension

geof:coordinateDimension (geom: ogc:geomLiteral): xsd:integer

The function geof:coordinateDimension returns the coordinate dimension of geom.

8.9.12. Function: geof:difference


The function geof:difference 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.

8.9.13. Function: geof:dimension

geof:dimension (geom: ogc:geomLiteral): xsd:integer

The function geof:dimension returns the dimension of geom. In non-homogeneous geometry collections, this will return the largest topological dimension of the contained objects.

8.9.14. Function: geof:metricDistance


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

8.9.15. Function: geof:distance

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

**NOTE** See the Recommendation for specification of units of measurement.

### 8.9.16. Function: geof:envelope

```
```

The function `geof:envelope` returns the minimum bounding box - a rectangle - of `geom`. Calculations are in the spatial reference system of `geom`.

### 8.9.17. Function: geof:geometryN

```
```

The function `geof:geometryN` returns the nth geometry of `geom` if it is a GeometryCollection that is defined in a literal type (such as in the case of a sf:GeometryCollection) or `geom` if it is a Geometry. This function is not applicable to the type geo:GeometryCollection, as elements in geo:GeometryCollection are not guaranteed to be ordered.

### 8.9.18. Function: geof:geometryType

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

The function `geof:geometryType` returns the URI of the subtype of Geometry of which this geometric object is an member. No attempt to reconcile different geometry subtypes across all support literals need be made.

### 8.9.19. Function: geof:getSRID

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

The function `geof:getSRID` returns the spatial reference system IRI for `geom`.

### 8.9.20. Function: geof:intersection

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

...
The function `geof:intersection` returns a geometric object that represents all Points in the intersection of `geom1` with `geom2`. Calculations are in the spatial reference system of `geom1`.

8.9.21. Function: `geof:is3D`  

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

The function `geof:is3D` returns true if `geom` has z coordinate values.

8.9.22. Function: `geof:isEmpty`  

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

The function `geof:isEmpty` returns true if `geom` is an empty geometry, i.e. contains no coordinates.

8.9.23. Function: `geof:isMeasured`  

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

The function `geof:isMeasured` returns true if `geom` has m coordinate values.

8.9.24. Function: `geof:isSimple`  

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

The function `geof:isSimple` returns true if `geom` is a simple geometry, i.e. has no anomalous geometric points, such as self intersection or self tangency.

8.9.25. Function: `geof:metricLength`  

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

The function `geof:metricLength` returns the length of `geom` in meters. The longest length from any one dimension is returned. This is for example the length of a line from its beginning point to its endpoint or the length of the boundary of a polygon. This function is similar to `geof:length` but does not need a specification of measurement unit.

The function `geof:length` returns the length of `geom`. The longest length from any one dimension is returned. This function is similar to `geof:metricLength`, which does not need a specification of measurement unit.

**NOTE** See the Recommendation for specification of units of measurement.

### 8.9.27. Function: geof:maxX

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

### 8.9.28. Function: geof:maxY

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

### 8.9.29. Function: geof:maxZ

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

### 8.9.30. Function: geof:minX

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

### 8.9.31. Function: geof:minY

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

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

### 8.9.33. Function: `geof:numGeometries`

The function `geof:numGeometries` returns the number of geometries of `geom`.

### 8.9.34. Function: `geof:perimeter`

The function `geof:perimeter` returns the perimeter of `geom` in the unit specified by the unit parameter for areal geometries. For non-areal geometries the result is equivalent to `geof:hasLength`.

### 8.9.35. Function: `geof:metricPerimeter`

The function `geof:metricPerimeter` returns the perimeter of `geom`. It is similar to the function `geof:perimeter`, but always returns the result in meters.

### 8.9.36. Function: `geof:spatialDimension`

The function `geof:spatialDimension` returns the spatial dimension of `geom`.

### 8.9.37. Function: `geof:symDifference`

The function `geof:symDifference` 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`.

### 8.9.38. Function: `geof:transform`
The function `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 as the type of the input literal.

### 8.9.39. Function: `geof:union`

The function `geof:union` 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 43** Implementations shall support `geof:getSRID` as a SPARQL extension function.

A.3.1.7 /conf/geometry-extension/srid-function

**Req 44** Implementations shall support `geof:getSRID` as a SPARQL extension function for DGGS literals.

A.4.1.3 /conf/geometry-extension-dggs/srid-function

### 8.10. Spatial Aggregate Functions

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


A.3.1.8 /conf/geometry-extension/sa-functions

**Req 46** Implementations shall support the functions of Requirement 45 as SPARQL extension functions which are defined in this standard, for DGGS geometry literals, in a manner which is consistent with definitions of these functions in Simple Features [OGCSFACA] [ISO19125-1].

[req_geometry_extension_dggs_sa_functions]

### 8.10.1. Function: `geof:aggBoundingBox`

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

8.10.2. Function: geof:aggBoundingCircle

```
```

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` calculates the centroid of the set of given geometries.

8.10.4. Function: geof:aggConcaveHull

```
ogc:geomLiteral
```

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

8.10.5. Function: geof:aggConvexHull

```
```

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

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

8.10.6. Function: geof:aggUnion

```
```

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

**NOTE**
This function is similar in name to `geof:union` used to calculate the union of just two geometries.
[5] https://www.w3.org/ns/locn
[6] https://www.w3.org/TR/vocab-dcat/#spatial-properties
[8] https://www.w3.org/TR/sparql11-query/#expressions
[9] https://www.w3.org/TR/sparql11-query/#operatorExtensibility
[10] https://www.ogc.org/def-server
Chapter 9. Geometry Topology Extension

This clause establishes the *Geometry Topology Extension* parameterized Requirements class with base IRI `/req/geometry-topology-extension`, which defines a collection of topological query functions that operate on geometry literals. These Requirements are parameterized to give implementations flexibility in the topological relation families and geometry serializations that they choose to support. These Requirements have a single corresponding conformance class *Geometry Topology Extension*, 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 to 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 [OGCSFACA] [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 47** Implementations shall support `geof:relate` as a SPARQL extension function, consistent with the relate operator defined in Simple Features [OGCSFACA] [ISO19125-1].

A.5.1.1 /conf/geometry-topology-extension/relate-query-function

9.2.1. Function: `geof:relate`

```
```

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

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

**Req 48** 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 [OGCSFACA] [ISO19125-1]. A.5.2.1 /conf/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>
<thead>
<tr>
<th>Query Function</th>
<th>Defining DE-9IM Intersection Pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>geof:sfIntersects(geom1: ogc:geomLiteral, geom2: ogc:geomLiteral): xsd:boolean</code></td>
<td><code>(Ffffffff F**T***** F***T****)</code></td>
</tr>
<tr>
<td><code>geof:sfTouches(geom1: ogc:geomLiteral, geom2: ogc:geomLiteral): xsd:boolean</code></td>
<td><code>(Ffffffff F**T***** F***T****)</code></td>
</tr>
<tr>
<td><code>geof:sfCrosses(geom1: ogc:geomLiteral, geom2: ogc:geomLiteral): xsd:boolean</code></td>
<td><code>(T*T***T**) for P/L, P/A, L/A; (0*T***T**) for L/L</code></td>
</tr>
<tr>
<td><code>geof:sfWithin(geom1: ogc:geomLiteral, geom2: ogc:geomLiteral): xsd:boolean</code></td>
<td><code>(T*T***T**) for A/A, P/P; (1*T***T**) for L/L</code></td>
</tr>
</tbody>
</table>

### 9.4. Egenhofer Relation Family

This clause establishes Requirements for the *Egenhofer* relation family. Consult references [FORMAL] and [CATEG] for a more detailed discussion of *Egenhofer* relations.
**Req 49** 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 [OGCSFACA] [ISO19125-1].

A.5.3.1 /conf/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**

<table>
<thead>
<tr>
<th>Query Function</th>
<th>Defining DE-9IM Intersection Pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>geof:ehCoveredBy(geom1: ogc:geomLiteral, geom2: ogc:geomLiteral): xsd:boolean</code></td>
<td><code>(TFF*TFT**)</code></td>
</tr>
</tbody>
</table>

**9.5. RCC8 Relation Family**

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

**Req 50** 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 [OGCSFACA] [ISO19125-1].

A.5.4.1 /conf/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**

<table>
<thead>
<tr>
<th>Query Function</th>
<th>Defining DE-9IM Intersection Pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>geof:rcc8ec(geom1: ogc:geomLiteral, geom2: ogc:geomLiteral): xsd:boolean</code></td>
<td><code>(F***** F**T***** F***T****)</code></td>
</tr>
<tr>
<td><code>geof:rcc8ntpp(geom1: ogc:geomLiteral, geom2: ogc:geomLiteral): xsd:boolean</code></td>
<td><code>(TFF*FFT**)</code></td>
</tr>
</tbody>
</table>

67
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

<table>
<thead>
<tr>
<th>Query Function</th>
<th>Defining DE-9IM Intersection Pattern</th>
</tr>
</thead>
</table>
Chapter 10. RDFS Entailment Extension

This clause establishes the *RDFS Entailment Extension* parameterized Requirements class with base 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. These Requirements have a single corresponding conformance class *RDFS Entailment Extension*, 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 51** Basic graph pattern matching shall use the semantics defined by the RDFS Entailment Regime [SPARQLENT].

A.6.1.1 /conf/rdfsentailmentextension/bgp-rdfs-ent

10.3. WKT Serialization

This section establishes the requirements for representing geometry data in RDF based on WKT as defined by Simple Features [OGCSFACA] [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 (sibling resource to this specification) does this. The following list gives the class hierarchy with each indented item being a subclass of the item in the line above. The class hierarchy 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 52 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 [OGCSFACA] [ISO19125-1].

A.6.2.1 /conf/rdfs-entailment-extension/wkt-geometry-types

10.4. GML Serialization

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

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 ;
```
Req 53 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 [GML].
This clause establishes the *Query Rewrite Extension* parameterized Requirements class with base IRI `/req/query-rewrite-extension`, which has a single corresponding conformance class *Query Rewrite Extension*, with IRI `/conf/query-rewrite-extension`. These Requirements define 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] and shown in *Presentation Syntax* 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. `ogc:asGeomLiteral` is used to indicate one of the properties that link `geo:Geometry` instances to serialisations, such as `geo:asWKT` or `geo:asGeoJSON`. The variables `?so1` & `?so2` represent `geo:SpatialObject` instances (either `geo:Feature` or `geo:Geometry` instances), `?g1` & `?g2` `geo:Geometry` instances only and `?g1Serial` & `?g2Serial` represent `geo:Geometry` instance serializations, e.g. `geo:asWKT` etc. literals.

```riffml
Forall ?so1 ?so2 ?g1 ?g2 ?g1Serial ?g2Serial ( 
  ?so1[ogc:relation->?so2] :- Or ( 
    And ( 
      # feature - feature rule 
      ?so1[geo:hasDefaultGeometry->?g1] 
      ?so2[geo:hasDefaultGeometry->?g2] 
      ?g1[ogc:asGeomLiteral->?g1Serial] 
      ?g2[ogc:asGeomLiteral->?g2Serial] 
      External(ogc:function(?g1Serial, ?g2Serial)) 
    ) 
    And ( 
      # feature - geometry rule 
      ?so1[geo:hasDefaultGeometry->?g1] 
      ?g1[ogc:asGeomLiteral->?g1Serial] 
      ?so2[ogc:asGeomLiteral->?g2Serial] 
      External(ogc:function(?g1Serial, ?g2Serial)) 
    ) 
    And ( 
      # geometry - feature rule 
      ?so1[ogc:asGeomLiteral->?g1Serial] 
      ?so2[geo:hasDefaultGeometry->?g2] 
      ?g2[ogc:asGeomLiteral->?g2Serial] 
      External(ogc:function(?g1Serial, ?g2Serial)) 
    ) 
    And ( 
      # geometry - geometry rule 
      ?so1[ogc:asGeomLiteral->?g1Serial] 
      ?so2[ogc:asGeomLiteral->?g2Serial] 
      External(ogc:function(?g1Serial, ?g2Serial)) 
  ) 
) 
```
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

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.


A.7.1.1 /conf/query-rewrite-extension/sf-query-rewrite-rewrite

**Table 9. Simple Features Query Transformation Rules**

<table>
<thead>
<tr>
<th>Rule</th>
<th>ogc:relation</th>
<th>ogc:function</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>geor:sfEquals</code></td>
<td><code>geo:sfEquals</code></td>
<td><code>geof:sfEquals</code></td>
</tr>
<tr>
<td><code>geor:sfDisjoint</code></td>
<td><code>geo:sfDisjoint</code></td>
<td><code>geof:sfDisjoint</code></td>
</tr>
<tr>
<td><code>geor:sfIntersects</code></td>
<td><code>geo:sfIntersects</code></td>
<td><code>geof:sfIntersects</code></td>
</tr>
<tr>
<td><code>geor:sfTouches</code></td>
<td><code>geo:sfTouches</code></td>
<td><code>geof:sfTouches</code></td>
</tr>
<tr>
<td><code>geor:sfCrosses</code></td>
<td><code>geo:sfCrosses</code></td>
<td><code>geof:sfCrosses</code></td>
</tr>
<tr>
<td><code>geor:sfWithin</code></td>
<td><code>geo:sfWithin</code></td>
<td><code>geof:sfWithin</code></td>
</tr>
<tr>
<td><code>geor:sfContains</code></td>
<td><code>geo:sfContains</code></td>
<td><code>geof:sfContains</code></td>
</tr>
<tr>
<td><code>geor:sfOverlaps</code></td>
<td><code>geo:sfOverlaps</code></td>
<td><code>geof:sfOverlaps</code></td>
</tr>
</tbody>
</table>

11.3. Egenhofer Relation Family

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.


A.7.2.1 /conf/query-rewrite-extension/eh-query-rewrite-rewrite

**Table 10. Egenhofer Query Transformation Rules**

<table>
<thead>
<tr>
<th>Rule</th>
<th>ogc:relation</th>
<th>ogc:function</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>geor:ehEquals</code></td>
<td><code>geo:ehEquals</code></td>
<td><code>geof:ehEquals</code></td>
</tr>
<tr>
<td><code>geor:ehDisjoint</code></td>
<td><code>geo:ehDisjoint</code></td>
<td><code>geof:ehDisjoint</code></td>
</tr>
<tr>
<td><code>geor:ehMeet</code></td>
<td><code>geo:ehMeet</code></td>
<td><code>geof:ehMeet</code></td>
</tr>
<tr>
<td><code>geor:ehOverlap</code></td>
<td><code>geo:ehOverlap</code></td>
<td><code>geof:ehOverlap</code></td>
</tr>
<tr>
<td><code>geor:ehCovers</code></td>
<td><code>geo:ehCovers</code></td>
<td><code>geof:ehCovers</code></td>
</tr>
<tr>
<td><code>geor:ehCoveredBy</code></td>
<td><code>geo:ehCoveredBy</code></td>
<td><code>geof:ehCoveredBy</code></td>
</tr>
<tr>
<td><code>geor:ehInside</code></td>
<td><code>geo:ehInside</code></td>
<td><code>geof:ehInside</code></td>
</tr>
<tr>
<td><code>geor:ehContains</code></td>
<td><code>geo:ehContains</code></td>
<td><code>geof:ehContains</code></td>
</tr>
</tbody>
</table>
11.4. RCC8 Relation Family

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 56
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:rcc8tp, geor:rcc8ntpp and geor:rcc8ntppi.

### A.7.3.1
/conf/query-rewrite-extension/rcc8-query-rewrite

Table 11. RCC8 Query Transformation Rules

<table>
<thead>
<tr>
<th>Rule</th>
<th>ogc:relation</th>
<th>ogc:function</th>
</tr>
</thead>
<tbody>
<tr>
<td>geor:rcc8eq</td>
<td>geor:rcc8eq</td>
<td>geof:rcc8eq</td>
</tr>
<tr>
<td>geor:rcc8dc</td>
<td>geor:rcc8dc</td>
<td>geof:rcc8dc</td>
</tr>
<tr>
<td>geor:rcc8ec</td>
<td>geor:rcc8ec</td>
<td>geof:rcc8ec</td>
</tr>
<tr>
<td>geor:rcc8po</td>
<td>geor:rcc8po</td>
<td>geof:rcc8po</td>
</tr>
<tr>
<td>geor:rcc8tppi</td>
<td>geor:rcc8tppi</td>
<td>geof:rcc8tppi</td>
</tr>
<tr>
<td>geor:rcc8tp</td>
<td>geor:rcc8tp</td>
<td>geof:rcc8tp</td>
</tr>
<tr>
<td>geor:rcc8ntpp</td>
<td>geor:rcc8ntpp</td>
<td>geof:rcc8ntpp</td>
</tr>
<tr>
<td>geor:rcc8ntppi</td>
<td>geor:rcc8ntppi</td>
<td>geof:rcc8ntppi</td>
</tr>
</tbody>
</table>

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.
Chapter 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 suggested but won't be articulated here. Instead they will be discussed and decided upon 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. Conformance classes may be used to signify the compatibility of a given implementation to parts of the GeoSPARQL standard. They may be stated as part of a SPARQL 1.1 Service Description [SPARQLSERVDESC].
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: [SPARQL]

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

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

A.2.1 Simple Features Relation Family

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 Egenhofer Relation Family

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 RCC8 Relation Family

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

This Conformance Class applies to non-DGGS geometries. See A.4. Conformance Class: Geometry Extension - DGGS for DGGS geometries.

Conformance Class IRI: /conf/geometry-extension

A.3.1 Tests for all Serializations except DGGS

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: geo:Geometry
d. Test Type: Capabilities

A.3.1.2 /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: Geometry Collection
d. Test Type: Capabilities

A.3.1.3 /conf/core/feature-properties

Requirement: /req/core/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.3.1.4 /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.3.1.5 /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 [OGCSFACA] [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.3.1.6 /conf/geometry-extension/query-functions-non-sf**  

**Requirement:** /req/geometry-extension/query-functions-non-sf  

Implementations shall support the functions `geof:metricLength`, `geof:length`, `geof:metricPerimeter`, `geof:perimeter`, `geof:metricArea`, `geof:area`, `geof:geometryN`, `geof:maxX`, `geof:maxY`, `geof:maxZ`, `geof:minX`, `geof:minY`, `geof:minZ` and `geof:numGeometries` as SPARQL extension functions which are
defined in this standard, for non-DGGS geometry literals.

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:


c. **Reference**: Section 8.9

d. **Test Type**: Capabilities

### A.3.1.7 /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 8.9.19

d. **Test Type**: Capabilities

### A.3.1.8 /conf/geometry-extension/sa-functions

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

Implementations shall support *geof:aggBoundingBox*, *geof:aggBoundingCircle*, *geof:aggCentroid*, *geof:aggConcaveHull*, *geof:aggConvexHull* and *geof:aggUnion* 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**: Section 8.10

d. **Test Type**: Capabilities

### A.3.2 WKT Serialization

#### 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.
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 8.8.2
d. **Test Type:** Capabilities

A.3.2.6 /conf/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:** Section 8.8.2.1
d. **Test Type:** Capabilities

A.3.3 GML Serialization

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 8.8.3.1
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 8.8.3.1
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 8.8.3.1
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
d. **Test Type:** Capabilities

**A.3.3.5 /conf/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:** Section 8.8.4.1
d. **Test Type:** Capabilities
A.3.4 GeoJSON Serialization

A.3.4.1 /conf/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 8.8.5.1
d. Test Type: Capabilities

A.3.4.2 /conf/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 8.8.5.1
d. Test Type: Capabilities

A.3.4.3 /conf/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 8.8.5.1
d. Test Type: Capabilities

A.3.4.4 /conf/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 8.8.6

d. **Test Type:** Capabilities

### A.3.4.5 /conf/geometry-extension/asGeoJSON-function

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

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

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

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

c. **Reference:** Section 8.8.6.1

d. **Test Type:** Capabilities

### A.3.5 KML Serialization

### A.3.5.1 /conf/geometry-extension/kmlLiteral

**Requirement:** /req/geometry-extension/kmlLiteral

All `geo:kmlLiteral` instances shall consist of a valid element from the KML schema that implements `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:** Section 8.8.7.1

d. **Test Type:** Capabilities

### A.3.5.2 /conf/geometry-extension/kmlLiteral-srs

**Requirement:** /req/geometry-extension/kmlLiteral-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:** Section 8.8.7.1
d. **Test Type**: Capabilities

**A.3.5.3 /conf/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**: Section 8.8.7.1

d. **Test Type**: Capabilities

**A.3.5.4 /conf/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 8.8.8

d. **Test Type**: Capabilities

**A.3.5.5 /conf/geometry-extension/asKML-function**

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

Implementations shall support `geof:asKML`, 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**: Section 8.8.8.1

d. **Test Type**: Capabilities
A.4. Conformance Class: Geometry Extension - DGGS

This conformance Class applies only to DGGS geometries. See A.3 Conformance Class: Geometry Extension for other geometries.

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

A.4.1 Tests for DGGS Serializations

A.4.1.1 /conf/geometry-extension-dggs/query-functions

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

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

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.4.1.2 /conf/geometry-extension-dggs/query-functions-non-sf

Requirement: /req/geometry-extension-dggs/geometry-properties-non-sf

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

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.4.1.3 /conf/geometry-extension-dggs/srid-function

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

Implementations shall support `geof:getSRID` as a SPARQL extension function for DGGS geometry
literals.

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

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

c. **Reference:** Section 8.9.19

d. **Test Type:** Capabilities

### A.4.1.4 /conf/geometry-extension-dggs/sa-functions

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

Implementations shall support the functions of Requirement A.3.1.8 /conf/geometry-extension/sa-functions as SPARQL extension functions which are defined in this standard, for DGGS geometry literals, in a manner which is consistent with definitions of these functions in Simple Features [OGCSFACA] [ISO19125-1].

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:** Section 8.10

d. **Test Type:** Capabilities

### A.4.2 DGGS Serialization

### A.4.2.1 /conf/geometry-extension-dggs/dggs-literal

**Requirement:** /req/geometry-extension-dggs/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:** Section 8.8.9.1

d. **Test Type:** Capabilities

### A.4.2.2 /conf/geometry-extension-dggs/dggs-literal-empty

**Requirement:** /req/geometry-extension-dggs/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:** Section 8.8.9.1

d. **Test Type:** Capabilities

### A.4.2.3 /conf/geometry-extension-dggs/geometry-as-dggs-literal

**Requirement:** /req/geometry-extension-dggs/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 8.8.10

d. **Test Type:** Capabilities

### A.4.2.4 /conf/geometry-extension-dggs/asDGGS-function

**Requirement:** /req/geometry-extension-dggs/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:** Section 8.8.10.1

d. **Test Type:** Capabilities
A.5 Conformance Class: Geometry Topology Extension

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

A.5.1 Tests for all relation families

A.5.1.1 /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 [OGCSFACA] [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.5.2 Simple Features Relation Family

A.5.2.1 /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 [OGCSFACA] [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.5.3 Egenhofer Relation Family
A.5.3.1 /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 [OGCSFACA] [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.5.4 RCC8 Relation Family

A.5.4.1 /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 [OGCSFACA] [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.6 Conformance Class: RDFS Entailment Extension

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

A.6.1 Tests for all implementations

A.6.1.1 /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**: [req_rdfs_entailment_extension_bgp_rdfs_ent]

d. **Test Type**: Capabilities

A.6.2 WKT Serialization

A.6.2.1 /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 [OGCSFACA] [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.6.3 GML Serialization

A.6.3.1 /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 [GML].

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.7 Conformance Class: Query Rewrite Extension

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

A.7.1 Simple Features Relation Family

A.7.1.1 /conf/query-rewrite-extension/sf-query-rewrite

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


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 7.2
d. Test Type: Capabilities

A.7.2 Egenhofer Relation Family

A.7.2.1 /conf/query-rewrite-extension/eh-query-rewrite

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


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 7.3
d. Test Type: Capabilities
A.7.3 RCC8 Relation Family

A.7.3.1 /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 7.4

d. Test Type: Capabilities
Annex B - Functions Summary (normative)
B.0 Overview

This annex summarizes all the functions defined in GeoSPARQL, providing descriptions of their parameters and return types.

The value `ogc:geomLiteral` indicates any one of the specific geometry serializations datatypes defined in this Specification, for example `geo:wktLiteral`.

The geometry subtypes - Polygon, Point, CellList etc. - are the Simple Features specification [OGCSFACA] [ISO19125-1] or DGGS types, as implemented by the various geometry serialization specifications referenced here. See Section 8.8 for the individual specification references.
## B.1 Functions Summary Table

### Simple Features Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Input Datatypes</th>
<th>Input Subtypes</th>
<th>Output Datatype</th>
<th>Output Subtype</th>
</tr>
</thead>
<tbody>
<tr>
<td>sfContains</td>
<td>2x ogc:geomLiteral</td>
<td>1x Polygon, 1x Geometry</td>
<td>xsd:boolean</td>
<td></td>
</tr>
<tr>
<td>sfCrosses</td>
<td>2x ogc:geomLiteral</td>
<td>1x Point or LineString, 1x LineString or Polygon</td>
<td>xsd:boolean</td>
<td></td>
</tr>
<tr>
<td>sfDisjoint</td>
<td>2x ogc:geomLiteral</td>
<td>2x Geometry</td>
<td>xsd:boolean</td>
<td></td>
</tr>
<tr>
<td>sfEquals</td>
<td>2x ogc:geomLiteral</td>
<td>2x Geometry</td>
<td>xsd:boolean</td>
<td></td>
</tr>
<tr>
<td>sfIntersects</td>
<td>2x ogc:geomLiteral</td>
<td>2x Geometry</td>
<td>xsd:boolean</td>
<td></td>
</tr>
<tr>
<td>sfOverlaps</td>
<td>2x ogc:geomLiteral</td>
<td>2x Point or 2x LineString or 2x Polygon</td>
<td>xsd:boolean</td>
<td></td>
</tr>
<tr>
<td>sfTouches</td>
<td>2x ogc:geomLiteral</td>
<td>2x Geometry but not Point</td>
<td>xsd:boolean</td>
<td></td>
</tr>
<tr>
<td>sfWithin</td>
<td>2x ogc:geomLiteral</td>
<td>1x Geometry, 1x Polygon</td>
<td>xsd:boolean</td>
<td></td>
</tr>
</tbody>
</table>

### Egenhofer Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Input Datatypes</th>
<th>Input Subtypes</th>
<th>Output Datatype</th>
<th>Output Subtype</th>
</tr>
</thead>
<tbody>
<tr>
<td>ehContains</td>
<td>2x ogc:geomLiteral</td>
<td>1x Polygon, 1x Geometry</td>
<td>xsd:boolean</td>
<td></td>
</tr>
<tr>
<td>ehCoveredBy</td>
<td>2x ogc:geomLiteral</td>
<td>1x Polygon, 1x Geometry</td>
<td>xsd:boolean</td>
<td></td>
</tr>
<tr>
<td>ehCovers</td>
<td>2x ogc:geomLiteral</td>
<td>1x Polygon, 1x Geometry</td>
<td>xsd:boolean</td>
<td></td>
</tr>
<tr>
<td>ehDisjoint</td>
<td>2x ogc:geomLiteral</td>
<td>2x Geometry</td>
<td>xsd:boolean</td>
<td></td>
</tr>
<tr>
<td>ehEquals</td>
<td>2x ogc:geomLiteral</td>
<td>2x Geometry</td>
<td>xsd:boolean</td>
<td></td>
</tr>
<tr>
<td>ehMeet</td>
<td>2x ogc:geomLiteral</td>
<td>2x Geometry but not Point</td>
<td>xsd:boolean</td>
<td></td>
</tr>
<tr>
<td>ehOverlap</td>
<td>2x ogc:geomLiteral</td>
<td>2x Geometry</td>
<td>xsd:boolean</td>
<td></td>
</tr>
<tr>
<td>Function</td>
<td>Input Datatypes</td>
<td>Input Subtypes</td>
<td>Output Datatype</td>
<td>Output Subtype</td>
</tr>
<tr>
<td>-----------</td>
<td>----------------</td>
<td>----------------</td>
<td>----------------</td>
<td>------------------------------</td>
</tr>
<tr>
<td>rcc8dcc</td>
<td>2x</td>
<td>2x Polygon</td>
<td>xsd:boolean</td>
<td></td>
</tr>
<tr>
<td>rcc8ecc</td>
<td>2x</td>
<td>2x Polygon</td>
<td>xsd:boolean</td>
<td></td>
</tr>
<tr>
<td>rcc8eq</td>
<td>2x</td>
<td>2x Polygon</td>
<td>xsd:boolean</td>
<td></td>
</tr>
<tr>
<td>rcc8ntpp</td>
<td>2x</td>
<td>2x Polygon</td>
<td>xsd:boolean</td>
<td></td>
</tr>
<tr>
<td>rcc8ntppi</td>
<td>2x</td>
<td>2x Polygon</td>
<td>xsd:boolean</td>
<td></td>
</tr>
<tr>
<td>rcc8po</td>
<td>2x</td>
<td>2x Polygon</td>
<td>xsd:boolean</td>
<td></td>
</tr>
<tr>
<td>rcc8tppp</td>
<td>2x</td>
<td>2x Polygon</td>
<td>xsd:boolean</td>
<td></td>
</tr>
<tr>
<td>rcc8tppi</td>
<td>2x</td>
<td>2x Polygon</td>
<td>xsd:boolean</td>
<td></td>
</tr>
</tbody>
</table>

**Spatial Aggregate Functions**

<table>
<thead>
<tr>
<th>Function</th>
<th>Input Datatypes</th>
<th>Input Subtypes</th>
<th>Output Datatype</th>
<th>Output Subtype</th>
</tr>
</thead>
<tbody>
<tr>
<td>aggBoundingBox</td>
<td>1 or more</td>
<td>ogc:geomLiteral</td>
<td>ogc:geomLiteral</td>
<td>square Polygon (not DGGS), CellList (DGGS)</td>
</tr>
<tr>
<td>aggBoundingCircle</td>
<td>1 or more</td>
<td>ogc:geomLiteral</td>
<td>ogc:geomLiteral</td>
<td>Polygon (not DGGS), CellList (DGGS)</td>
</tr>
<tr>
<td>aggCentroid</td>
<td>1 or more</td>
<td>ogc:geomLiteral</td>
<td>ogc:geomLiteral</td>
<td>Point (not DGGS), Cell (DGGS)</td>
</tr>
<tr>
<td>aggConcaveHull</td>
<td>1 or more</td>
<td>ogc:geomLiteral</td>
<td>ogc:geomLiteral</td>
<td>Polygon (not DGGS), CellList (DGGS)</td>
</tr>
<tr>
<td>aggConvexHull</td>
<td>1 or more</td>
<td>ogc:geomLiteral</td>
<td>ogc:geomLiteral</td>
<td>Polygon (not DGGS), CellList (DGGS)</td>
</tr>
</tbody>
</table>
### AggUnion

- **Input Datatypes**: 1 or more `ogc:geomLiteral`
- **Output Datatype**: `ogc:geomLiteral`
- **Output Subtype**: Polygon (not DGGS), CellList (DGGS)

### Non-topological Query Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Input Datatypes</th>
<th>Input Subtypes</th>
<th>Output Datatype</th>
<th>Output Subtype</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>metricArea</strong></td>
<td>1x <code>ogc:geomLiteral</code></td>
<td>Polygon</td>
<td><code>xsd:double</code></td>
<td></td>
</tr>
<tr>
<td><strong>area</strong></td>
<td>1x <code>ogc:geomLiteral</code></td>
<td>Polygon</td>
<td><code>xsd:double</code></td>
<td></td>
</tr>
<tr>
<td><strong>boundary</strong></td>
<td>1x <code>ogc:geomLiteral</code></td>
<td>Geometry</td>
<td><code>ogc:geomLiteral</code></td>
<td>LineString (not DGGS), OrderedCellList (DGGS)</td>
</tr>
<tr>
<td><strong>buffer</strong></td>
<td>1x <code>ogc:geomLiteral</code>, 1x <code>xsd:double</code>, 1x <code>xsd:anyURI</code></td>
<td>any</td>
<td><code>ogc:geomLiteral</code></td>
<td>(Multi)Polygon (not DGGS), CellList (DGGS)</td>
</tr>
<tr>
<td><strong>convexHull</strong></td>
<td>1x <code>ogc:geomLiteral</code></td>
<td>Geometry</td>
<td><code>ogc:geomLiteral</code></td>
<td>LineString (not DGGS)</td>
</tr>
<tr>
<td><strong>coordinateDimension</strong></td>
<td>1x <code>ogc:geomLiteral</code></td>
<td>Geometry</td>
<td><code>xsd:integer</code></td>
<td></td>
</tr>
<tr>
<td><strong>difference</strong></td>
<td>2x <code>ogc:geomLiteral</code></td>
<td>2x Geometry</td>
<td><code>ogc:geomLiteral</code></td>
<td>(Multi)Polygon (not DGGS), CellList (DGGS)</td>
</tr>
<tr>
<td><strong>dimension</strong></td>
<td>1x <code>ogc:geomLiteral</code></td>
<td>Geometry</td>
<td><code>xsd:double</code></td>
<td></td>
</tr>
<tr>
<td><strong>metricDistance</strong></td>
<td>2x <code>ogc:geomLiteral</code>, 1x <code>xsd:anyURI</code></td>
<td>2x Geometry</td>
<td><code>xsd:double</code></td>
<td></td>
</tr>
<tr>
<td><strong>distance</strong></td>
<td>2x <code>ogc:geomLiteral</code>, 1x <code>xsd:anyURI</code></td>
<td>2x Geometry</td>
<td><code>rdfs:Resource</code></td>
<td></td>
</tr>
<tr>
<td><strong>envelope</strong></td>
<td>1x</td>
<td>1x</td>
<td>1x</td>
<td>1x</td>
</tr>
<tr>
<td>----------</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
</tr>
<tr>
<td><strong>geometryN</strong></td>
<td>1x</td>
<td>1x</td>
<td>1x</td>
<td>1x</td>
</tr>
<tr>
<td><strong>geometryType</strong></td>
<td>1x</td>
<td>1x</td>
<td>1x</td>
<td>1x</td>
</tr>
<tr>
<td><strong>getSRID</strong></td>
<td>1x</td>
<td>1x</td>
<td>1x</td>
<td>1x</td>
</tr>
<tr>
<td><strong>intersection</strong></td>
<td>2x</td>
<td>2x</td>
<td>2x</td>
<td>2x</td>
</tr>
<tr>
<td><strong>is3D</strong></td>
<td>1x</td>
<td>1x</td>
<td>1x</td>
<td>1x</td>
</tr>
<tr>
<td><strong>isEmpty</strong></td>
<td>1x</td>
<td>1x</td>
<td>1x</td>
<td>1x</td>
</tr>
<tr>
<td><strong>isMeasured</strong></td>
<td>1x</td>
<td>1x</td>
<td>1x</td>
<td>1x</td>
</tr>
<tr>
<td><strong>isSimple</strong></td>
<td>1x</td>
<td>1x</td>
<td>1x</td>
<td>1x</td>
</tr>
<tr>
<td><strong>metricLength</strong></td>
<td>1x</td>
<td>1x</td>
<td>1x</td>
<td>1x</td>
</tr>
<tr>
<td><strong>length</strong></td>
<td>1x</td>
<td>1x</td>
<td>1x</td>
<td>1x</td>
</tr>
<tr>
<td><strong>numGeometries</strong></td>
<td>1x</td>
<td>1x</td>
<td>1x</td>
<td>1x</td>
</tr>
<tr>
<td><strong>metricPerimeter</strong></td>
<td>1x</td>
<td>1x</td>
<td>1x</td>
<td>1x</td>
</tr>
<tr>
<td><strong>perimeter</strong></td>
<td>1x</td>
<td>1x</td>
<td>1x</td>
<td>1x</td>
</tr>
<tr>
<td><strong>spatialDimension</strong></td>
<td>1x</td>
<td>1x</td>
<td>1x</td>
<td>1x</td>
</tr>
<tr>
<td><strong>symDifference</strong></td>
<td>2x</td>
<td>2x</td>
<td>2x</td>
<td>2x</td>
</tr>
<tr>
<td><strong>transform</strong></td>
<td>1x</td>
<td>1x</td>
<td>1x</td>
<td>1x</td>
</tr>
</tbody>
</table>
### Serialization Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Input Datatypes</th>
<th>Input Subtypes</th>
<th>Output Datatype</th>
<th>Output Subtype</th>
</tr>
</thead>
<tbody>
<tr>
<td>asDGGS</td>
<td>1x ogc:geomLiteral</td>
<td>Geometry</td>
<td>geo:dggsLiteral</td>
<td></td>
</tr>
<tr>
<td>asGeoJSON</td>
<td>1x ogc:geomLiteral</td>
<td>Geometry</td>
<td>geo:geoJSONLiteral</td>
<td></td>
</tr>
<tr>
<td>asGML</td>
<td>1x ogc:geomLiteral, 1x xsd:string</td>
<td>Geometry</td>
<td>geo:gmlLiteral</td>
<td></td>
</tr>
<tr>
<td>asKML</td>
<td>1x ogc:geomLiteral</td>
<td>Geometry</td>
<td>geo:kmlLiteral</td>
<td></td>
</tr>
<tr>
<td>asWKT</td>
<td>1x ogc:geomLiteral</td>
<td>Geometry</td>
<td>geo:wktLiteral</td>
<td></td>
</tr>
</tbody>
</table>

### Extent Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Input Datatypes</th>
<th>Input Subtypes</th>
<th>Output Datatype</th>
<th>Output Subtype</th>
</tr>
</thead>
<tbody>
<tr>
<td>getSRID</td>
<td>1x ogc:geomLiteral</td>
<td>Geometry</td>
<td>xsd:anyURI</td>
<td></td>
</tr>
<tr>
<td>maxX</td>
<td>1x ogc:geomLiteral</td>
<td>Geometry</td>
<td>xsd:double</td>
<td></td>
</tr>
<tr>
<td>maxY</td>
<td>1x ogc:geomLiteral</td>
<td>Geometry</td>
<td>xsd:double</td>
<td></td>
</tr>
<tr>
<td>maxZ</td>
<td>1x ogc:geomLiteral</td>
<td>Geometry</td>
<td>xsd:double</td>
<td></td>
</tr>
<tr>
<td>minX</td>
<td>1x ogc:geomLiteral</td>
<td>Geometry</td>
<td>xsd:double</td>
<td></td>
</tr>
<tr>
<td>minY</td>
<td>1x ogc:geomLiteral</td>
<td>Geometry</td>
<td>xsd:double</td>
<td></td>
</tr>
<tr>
<td>minZ</td>
<td>1x ogc:geomLiteral</td>
<td>Geometry</td>
<td>xsd:double</td>
<td></td>
</tr>
</tbody>
</table>

### Other Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Input Datatypes</th>
<th>Input Subtypes</th>
<th>Output Datatype</th>
<th>Output Subtype</th>
</tr>
</thead>
<tbody>
<tr>
<td>relate</td>
<td>2x ogc:geomLiteral</td>
<td></td>
<td>xsd:string</td>
<td></td>
</tr>
</tbody>
</table>
B.2 GeoSPARQL to SFA Functions Mapping

The following table indicates which GeoSPARQL non-topological query functions map to Simple Features Access ([OGCSFACA] [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.

<table>
<thead>
<tr>
<th>GeoSPARQL Function</th>
<th>in 1.0</th>
<th>in 1.1</th>
<th>SFA</th>
</tr>
</thead>
<tbody>
<tr>
<td>metricArea</td>
<td></td>
<td>x</td>
<td>Area</td>
</tr>
<tr>
<td>area</td>
<td></td>
<td>x</td>
<td>Area</td>
</tr>
<tr>
<td>asWKT*</td>
<td>x</td>
<td>x</td>
<td>AsBinary</td>
</tr>
<tr>
<td>boundary</td>
<td>x</td>
<td>x</td>
<td>Boundary</td>
</tr>
<tr>
<td>buffer</td>
<td>x</td>
<td>x</td>
<td>Buffer</td>
</tr>
<tr>
<td>convexHull</td>
<td>x</td>
<td>x</td>
<td>ConvexHull</td>
</tr>
<tr>
<td>coordinateDimension</td>
<td></td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>difference</td>
<td>x</td>
<td>x</td>
<td>Difference</td>
</tr>
<tr>
<td>dimension</td>
<td></td>
<td>x</td>
<td>Dimension</td>
</tr>
<tr>
<td>metricDistance</td>
<td>x</td>
<td></td>
<td>Distance</td>
</tr>
<tr>
<td>distance</td>
<td>x</td>
<td>x</td>
<td>EndPoint</td>
</tr>
<tr>
<td>envelope</td>
<td>x</td>
<td>x</td>
<td>Envelope</td>
</tr>
<tr>
<td>geometryN</td>
<td></td>
<td>x</td>
<td>GeometryN</td>
</tr>
<tr>
<td>geometryType</td>
<td></td>
<td>x</td>
<td>GeometryType</td>
</tr>
<tr>
<td>getSRID</td>
<td>x</td>
<td>x</td>
<td>SRID</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>InteriorRingN</td>
</tr>
<tr>
<td>intersection</td>
<td>x</td>
<td>x</td>
<td>Intersection</td>
</tr>
<tr>
<td>is3D</td>
<td></td>
<td>x</td>
<td>IsClosed</td>
</tr>
<tr>
<td>isEmpty</td>
<td></td>
<td>x</td>
<td>IsEmpty</td>
</tr>
<tr>
<td>isMeasured</td>
<td></td>
<td>x</td>
<td>IsRing</td>
</tr>
<tr>
<td>isSimple</td>
<td></td>
<td>x</td>
<td>IsSimple</td>
</tr>
<tr>
<td>metricLength</td>
<td></td>
<td>x</td>
<td>Length</td>
</tr>
<tr>
<td>GeoSPARQL Function</td>
<td>in 1.0</td>
<td>in 1.1</td>
<td>SFA</td>
</tr>
<tr>
<td>--------------------</td>
<td>--------</td>
<td>--------</td>
<td>-----</td>
</tr>
<tr>
<td>length</td>
<td></td>
<td>x</td>
<td>Length</td>
</tr>
<tr>
<td>maxX</td>
<td></td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>maxY</td>
<td></td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>maxZ</td>
<td></td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>minX</td>
<td></td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>minY</td>
<td></td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>minZ</td>
<td></td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>numGeometries</td>
<td></td>
<td>x</td>
<td>NumGeometries</td>
</tr>
<tr>
<td>perimeterLength</td>
<td></td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>perimeter</td>
<td></td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>spatialDimension</td>
<td></td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>symDifference</td>
<td></td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>transform</td>
<td></td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>union</td>
<td></td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Y</td>
</tr>
</tbody>
</table>

* 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 separately 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)

```xml
<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.

```xml
@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[^1] 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

```sparql
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*

```sparql
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)

```sparql
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 ;
```

[^1]: 

This example and the example below (B 1.1.2.4) show the same geo:Feature, but with a different specification of its area. This example shows the recommended way to express size: by using a subproperty of geo:hasMetricSize (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

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

eg:x
  a geo:Feature ;
  skos:prefLabel "Feature X";
  geo:hasGeometry [ geo:hasArea [ qudt:numericValue "2.2E5"^^xsd:double ; qudt:unit <http://qudt.org/vocab/unit/AC> ; # international acre ] ] .
```

Here a geo:Feature 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

```turtle
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 difference explained in annotation properties. No GeoSPARQL ontology properties are used to indicate a
difference in these Geometry instances thus machine use of this Feature would not be easily able to differentiate them.

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

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 geo:hasMetricSpatialResolution.

C.1.1.2.7 Feature with non-metric size

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
Here a Feature instance has two geometries, one indicated with the general property hasGeometry and a second indicated with the specialized 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 absolute URI and data.

```xml
<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

```xml
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 "<https://w3id.org/dggs/auspix> CELLLIST ((R1234 R1235 R1236 ... R1256))"^^geo:dggsLiteral;
  ];
.
```

Here a single Geometry, linked to a Feature instance, is expressed using two different serializations: Well-known Text and the DGGS with the AusPIX DGGS indicated by its IRI.

C.1.1.3.3 Geometry with scalar spatial property

```xml
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 serialization (WKT) indicated with the predicate geo: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.

```xml
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 subclasses 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` instance labels, their areas, geometries or any other property.

### C.1.1.6 GeometryCollection

This example shows a `GeometryCollection` instance containing 3 `Geometry` instances.

```xml
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` itself 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 serializations used in GeoSPARQL define the geometry type - point, polygon etc. within the literal, e.g. WKT can encode \texttt{POLYGON(())} or \texttt{POINT()}, however the Simple Features Vocabulary resource within GeoSPARQL 1.1 contains specialised Geometry RDF classes such as \texttt{sf:Polygon}, \texttt{sf:PolyhedralSurface} and others.

It may be appropriate to use these specialized forms of Geometry in circumstances when geometry type differentiation is required within RDF and not withing specialized literal handling. This is the case when type differentiation must occur within plain SPARQL, not GeoSPARQL.

The following example shows a Feature instance with two Geometry instances where the Simple Features Vocabulary classes are used to indicate the Geometry type:

```sparql
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, \texttt{geo:hasCentroid} indicates is could be used with a \texttt{sf:Point} and \texttt{geo:hasBoundingBox} indicates use with an \texttt{sf:Envelope}.

C.1.2 Properties

C.1.2.1 Spatial Object Properties

See the section C.1.1.2 Size Properties above.

C.1.2.2 Feature Properties

This example shows a \texttt{geo:Feature} instance with each of the properties defined in Section 6.4 used, except for the properties \texttt{geo:hasMetricSize} and \texttt{geo:hasSize}, that are intended to be used through their subproperties and \texttt{geo:hasMetricPerimeterLength} and \texttt{geo:hasPerimeterLength} which are examplified in C.1.1.1.2 Size Properties.
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 estimated 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.

```qudt
@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 properties defined for a Geometry instance has realistic values. For example, the isEmpty property 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 serializations. 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.

```qudt
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,
```
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:asXML """"<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]
    ]
}"""
geo:asDGGS """"https://w3id.org/dggs/auspix"""" 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
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.

```
@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 ;
```

Figure 4. Illustration of spatial data
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.
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 \texttt{my:hasPointGeometry}.

\begin{verbatim}
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)
        )
}
\end{verbatim}

Result:

\begin{verbatim}
?f
my:D
\end{verbatim}

C.2.2.3

Find all features that touch the union of feature \texttt{my:A} and feature \texttt{my:D}, where computations are based on \texttt{my:hasExactGeometry}.

\begin{verbatim}
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(\end{verbatim}
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/>

WHERE {
  
```

```
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:

```sparql
PREFIX geo: <http://www.opengis.net/ont/geosparql#>
SELECT ?f
WHERE { ?f geo:sfOverlaps my:A }
```

Transformed Query (application of transformation rule geor:sfOverlaps):

```sparql
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
    }
}
```
FILTER (geof:sfOverlaps(?fSerial, ?aSerial))
}
UNION
{
  # feature – geometry
  my:A geo:asWKT ?aSerial .
  FILTER (geof:sfOverlaps(?fSerial, ?aSerial))
}
UNION
{
  # geometry – feature
  ?f geo:asWKT ?fSerial .
  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.
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 only in very few cases 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 serialization 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

<table>
<thead>
<tr>
<th>SHACL Shape ID</th>
<th>Severity</th>
<th>Test purpose</th>
<th>Requirements tested</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shape 1a</td>
<td>Violation</td>
<td>Each node with an incoming <code>geo:hasGeometry</code>, or a specialization of it, should have at minimum one outgoing relation that is either <code>geo:hasSerialization</code>, or a specialization of it.</td>
<td>[req_geometry-extension_feature_properties], [req_geometry-extension_geometry_properties]</td>
</tr>
<tr>
<td>Shape 1b</td>
<td>Violation</td>
<td>Each node with an incoming <code>geo:hasGeometry</code>, or a specialization of it, can have a maximum of one outgoing <code>geo:asWKT</code> relation.</td>
<td>[req_geometry-extension_geometry_properties], [req_geometry-extension_geometry-as-wkt-literal]</td>
</tr>
<tr>
<td>Shape 1c</td>
<td>Violation</td>
<td>Each node with an incoming <code>geo:hasGeometry</code>, or a specialization of it, can have a maximum of one outgoing <code>geo:asGML</code> relation.</td>
<td>[req_geometry-extension_geometry_properties], [req_geometry-extension_geometry-as-gml-literal]</td>
</tr>
<tr>
<td>Shape 1d</td>
<td>Violation</td>
<td>Each node with an incoming <code>geo:hasGeometry</code>, or a specialization of it, can have a maximum of one outgoing <code>geo:asGeoJSON</code> relation.</td>
<td>[req_geometry-extension_geometry_properties], [req_geometry-extension_geometry-as-geojson-literal]</td>
</tr>
<tr>
<td>Shape 1e</td>
<td>Violation</td>
<td>Each node with an incoming <code>geo:hasGeometry</code>, or a specialization of it, can have a maximum of one outgoing <code>geo:asKML</code> relation.</td>
<td>[req_geometry-extension_geometry_properties], [req_geometry-extension_geometry-as-kml-literal]</td>
</tr>
<tr>
<td>SHACL Shape ID</td>
<td>Severity</td>
<td>Test purpose</td>
<td>Requirements tested</td>
</tr>
<tr>
<td>----------------</td>
<td>----------</td>
<td>--------------</td>
<td>---------------------</td>
</tr>
<tr>
<td>Shape 2</td>
<td>Violation</td>
<td>Each node with one or more outgoing relations that are either <code>geo:hasSerialization</code>, or a specialization of it, should have at least one incoming <code>geo:hasGeometry</code> relation or a specialization of it.</td>
<td>[req_geometry-extension_geometry-properties]</td>
</tr>
<tr>
<td>Shape 3a-c</td>
<td>Violation</td>
<td>A node that has an incoming <code>geo:hasGeometry</code> property, or specialization of it, cannot have an outgoing <code>geo:hasGeometry</code> property, or a specialization of, it at the same time (a <code>geo:Feature</code> cannot be a <code>geo:Geometry</code> at the same time)</td>
<td>[req_geometry-extension_feature-properties]</td>
</tr>
<tr>
<td>Shape 4</td>
<td>Violation</td>
<td>The target of a <code>geo:hasSerialization</code> property, or a specialization of, it should be an RDF literal</td>
<td>[req_geometry-extension_geometry-properties]</td>
</tr>
<tr>
<td>Shape 5</td>
<td>Violation</td>
<td>The target of a <code>geo:asWKT</code> property should be an RDF literal with datatype <code>geo:wktLiteral</code></td>
<td>[req_geometry-extension_wkt-literal]</td>
</tr>
<tr>
<td>Shape 6</td>
<td>Violation</td>
<td>The target of a <code>geo:asGML</code> property should be an RDF literal with datatype <code>geo:gmlLiteral</code></td>
<td>[req_geometry-extension_gml-literal]</td>
</tr>
<tr>
<td>SHACL Shape ID</td>
<td>Severity</td>
<td>Test purpose</td>
<td>Requirements tested</td>
</tr>
<tr>
<td>----------------</td>
<td>----------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>--------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Shape 7</td>
<td>Violation</td>
<td>The target of a <code>geo:asGeoJSON</code> property should be an RDF literal with datatype <code>geo:geoJSONLiteral</code></td>
<td>[req_geometry-extension_geojson-literal]</td>
</tr>
<tr>
<td>Shape 8</td>
<td>Violation</td>
<td>The target of a <code>geo:asKML</code> property should be an RDF literal with datatype <code>geo:kmlLiteral</code></td>
<td>[req_geometry-extension_kml-literal]</td>
</tr>
<tr>
<td>Shape 9</td>
<td>Violation</td>
<td>A <code>geo:Geometry</code> node should have a maximum of one outgoing <code>geo:coordinateDimension</code> property</td>
<td>[req_geometry-extension_geometry-properties]</td>
</tr>
<tr>
<td>Shape 10</td>
<td>Violation</td>
<td>A <code>geo:Geometry</code> node should have a maximum of one outgoing <code>geo:dimension</code> property</td>
<td>[req_geometry-extension_geometry-properties]</td>
</tr>
<tr>
<td>Shape 11</td>
<td>Violation</td>
<td>A <code>geo:Geometry</code> node should have a maximum of one outgoing <code>geo:isEmpty</code> property</td>
<td>[req_geometry-extension_geometry-properties]</td>
</tr>
<tr>
<td>Shape 12</td>
<td>Violation</td>
<td>A <code>geo:Geometry</code> node should have a maximum one outgoing <code>geo:isSimple</code> property</td>
<td>[req_geometry-extension_geometry-properties]</td>
</tr>
<tr>
<td>Shape 13</td>
<td>Violation</td>
<td>A <code>geo:Geometry</code> node should have maximum of one outgoing <code>geo:spatialDimension</code> property</td>
<td>[req_geometry-extension_geometry-properties]</td>
</tr>
<tr>
<td>Shape 14a</td>
<td>Violation</td>
<td>A <code>geo:Geometry</code> node should have maximum of one outgoing <code>geo:hasSpatialResolution</code> property</td>
<td>[req_geometry-extension_geometry-properties]</td>
</tr>
<tr>
<td>SHACL Shape ID</td>
<td>Severity</td>
<td>Test purpose</td>
<td>Requirements tested</td>
</tr>
<tr>
<td>----------------</td>
<td>------------</td>
<td>------------------------------------------------------------------------------</td>
<td>-----------------------------------------------------------</td>
</tr>
<tr>
<td>Shape 14b</td>
<td>Violation</td>
<td>A <code>geo:Geometry</code> node should have maximum of one outgoing <code>geo:hasSpatialAccuracy</code> property</td>
<td>[req_geometry-extension_geometry-properties]</td>
</tr>
<tr>
<td>Shape 14c</td>
<td>Violation</td>
<td>A <code>geo:Geometry</code> node should have maximum of one outgoing <code>geo:hasMetricSpatialAccuracy</code> property</td>
<td>[req_geometry-extension_geometry-properties]</td>
</tr>
<tr>
<td>Shape 14d</td>
<td>Violation</td>
<td>A <code>geo:Geometry</code> node should have maximum of one outgoing <code>geo:hasMetricSpatialResolution</code> property</td>
<td>[req_geometry-extension_geometry-properties]</td>
</tr>
<tr>
<td>Shape 15</td>
<td>Violation</td>
<td>The content of an RDF literal with an incoming <code>geo:asWKT</code> relation must conform to a well-formed WKT string, as defined by its official specification (Simple Features Access)</td>
<td>[req_geometry-extension_wkt-literal]</td>
</tr>
<tr>
<td>Shape 16</td>
<td>Violation</td>
<td>The content of an RDF literal with an incoming <code>geo:asWKT</code> relation must conform to a well-formed WKT string, as defined by its official specification (Simple Features Access)</td>
<td>[req_geometry-extension_gml-literal]</td>
</tr>
<tr>
<td>Shape 17</td>
<td>Violation</td>
<td>The content of an RDF literal with an incoming <code>geo:asGeoJSON</code> relation must conform to a well-formed GeoJSON geometry string, as defined by its official specification</td>
<td>[req_geometry-extension_geojson-literal]</td>
</tr>
<tr>
<td>SHACL Shape ID</td>
<td>Severity</td>
<td>Test purpose</td>
<td>Requirements tested</td>
</tr>
<tr>
<td>----------------</td>
<td>----------</td>
<td>--------------</td>
<td>---------------------</td>
</tr>
<tr>
<td>Shape 18</td>
<td>Violation</td>
<td>The content of an RDF literal with an incoming <code>geo:asKML</code> relation must conform to a well-formed KML geometry XML string, as defined by its official specification</td>
<td><code>[req_geometry-extension_kml-literal]</code></td>
</tr>
<tr>
<td>Shape 20</td>
<td>Violation</td>
<td>If both <code>geo:dimension</code> and <code>geo:coordinateDimension</code> properties are asserted, the value of <code>geo:dimension</code> should be less than or equal to the value of <code>geo:coordinateDimension</code></td>
<td><code>[req_geometry-extension_geometry-properties]</code></td>
</tr>
<tr>
<td>Shape 21a</td>
<td>Violation</td>
<td>An instance of <code>geo:FeatureCollection</code> should have at least one outgoing <code>rdfs:member</code> relation</td>
<td><code>[req_core_spatial-feature-collection-class]</code></td>
</tr>
<tr>
<td>Shape 21b</td>
<td>Violation</td>
<td>An instance of <code>geo:FeatureCollection</code> should only have outgoing <code>rdfs:member</code> relations going to <code>geo:Feature</code> instances</td>
<td><code>[req_core_spatial-feature-collection-class]</code></td>
</tr>
<tr>
<td>Shape 22a</td>
<td>Violation</td>
<td>An instance of <code>geo:GeometryCollection</code> should have at least one outgoing <code>rdfs:member</code> relation</td>
<td><code>[req_core_spatial-geometry-collection-class]</code></td>
</tr>
<tr>
<td>Shape 22b</td>
<td>Violation</td>
<td>An instance of <code>geo:GeometryCollection</code> should only have outgoing <code>rdfs:member</code> relations to <code>geo:Geometry</code> instances</td>
<td><code>[req_core_spatial-geometry-collection-class]</code></td>
</tr>
<tr>
<td>SHACL Shape ID</td>
<td>Severity</td>
<td>Test purpose</td>
<td>Requirements tested</td>
</tr>
<tr>
<td>---------------</td>
<td>----------</td>
<td>--------------</td>
<td>---------------------</td>
</tr>
<tr>
<td>Shape 23a</td>
<td>Violation</td>
<td>An instance of <code>geo:SpatialObjectCollection</code> should have at least one outgoing <code>rdfs:member</code> relation</td>
<td>[req_core_spatial-object-collection-class]</td>
</tr>
<tr>
<td>Shape 23b</td>
<td>Violation</td>
<td>An instance of <code>geo:SpatialObjectCollection</code> should only have outgoing <code>rdfs:member</code> relations going to <code>geo:SpatialObject</code> instances, or subclasses of them</td>
<td>[req_core_spatial-object-collection-class]</td>
</tr>
</tbody>
</table>
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.

<table>
<thead>
<tr>
<th>Prefix</th>
<th>URI</th>
</tr>
</thead>
<tbody>
<tr>
<td>as:</td>
<td><a href="https://www.w3.org/ns/activitystreams#">https://www.w3.org/ns/activitystreams#</a></td>
</tr>
<tr>
<td>dcterms:</td>
<td><a href="http://purl.org/dc/terms/">http://purl.org/dc/terms/</a></td>
</tr>
<tr>
<td>geo:</td>
<td><a href="http://www.opengis.net/ont/geosparql#">http://www.opengis.net/ont/geosparql#</a></td>
</tr>
<tr>
<td>geom:</td>
<td><a href="http://geovocab.org/geometry#">http://geovocab.org/geometry#</a></td>
</tr>
<tr>
<td>gn:</td>
<td><a href="https://www.geonames.org/ontology#">https://www.geonames.org/ontology#</a></td>
</tr>
<tr>
<td>juso:</td>
<td><a href="http://rdfs.co/juso/">http://rdfs.co/juso/</a></td>
</tr>
<tr>
<td>lgd:</td>
<td><a href="http://linkedgeodata.org/ontology/">http://linkedgeodata.org/ontology/</a></td>
</tr>
<tr>
<td>locn:</td>
<td><a href="https://www.w3.org/ns/locn">https://www.w3.org/ns/locn</a></td>
</tr>
<tr>
<td>obo:</td>
<td><a href="http://purl.obolibrary.org/obo/">http://purl.obolibrary.org/obo/</a></td>
</tr>
<tr>
<td>osm:</td>
<td><a href="https://w3id.org/openstreetmap/terms#">https://w3id.org/openstreetmap/terms#</a></td>
</tr>
<tr>
<td>osmm:</td>
<td><a href="https://www.openstreetmap.org/meta/">https://www.openstreetmap.org/meta/</a></td>
</tr>
<tr>
<td>osmt:</td>
<td><a href="https://wiki.openstreetmap.org/wiki/Key">https://wiki.openstreetmap.org/wiki/Key</a>:</td>
</tr>
<tr>
<td>pos:</td>
<td><a href="http://www.w3.org/2003/01/geo/wgs84_pos#">http://www.w3.org/2003/01/geo/wgs84_pos#</a></td>
</tr>
<tr>
<td>prov:</td>
<td><a href="http://www.w3.org/ns/prov#">http://www.w3.org/ns/prov#</a></td>
</tr>
<tr>
<td>rdf:</td>
<td><a href="http://www.w3.org/1999/02/22-rdf-syntax-ns#">http://www.w3.org/1999/02/22-rdf-syntax-ns#</a></td>
</tr>
<tr>
<td>rdfs:</td>
<td><a href="http://www.w3.org/2000/01/rdf-schema#">http://www.w3.org/2000/01/rdf-schema#</a></td>
</tr>
<tr>
<td>sdo:</td>
<td><a href="https://schema.org">https://schema.org</a></td>
</tr>
<tr>
<td>sosa:</td>
<td><a href="http://www.w3.org/ns/sosa/">http://www.w3.org/ns/sosa/</a></td>
</tr>
<tr>
<td>spatial:</td>
<td><a href="http://data.ordnancesurvey.co.uk/ontology/spatialrelations/">http://data.ordnancesurvey.co.uk/ontology/spatialrelations/</a></td>
</tr>
<tr>
<td>spatialuk:</td>
<td><a href="http://data.ordnancesurvey.co.uk/ontology/spatialrelations/">http://data.ordnancesurvey.co.uk/ontology/spatialrelations/</a></td>
</tr>
<tr>
<td>spatialukgeom:</td>
<td><a href="http://data.ordnancesurvey.co.uk/ontology/geometry/">http://data.ordnancesurvey.co.uk/ontology/geometry/</a></td>
</tr>
<tr>
<td>spatial:</td>
<td><a href="http://geovocab.org/spatial#">http://geovocab.org/spatial#</a></td>
</tr>
<tr>
<td>ssn:</td>
<td><a href="http://www.w3.org/ns/ssn/">http://www.w3.org/ns/ssn/</a></td>
</tr>
<tr>
<td>time:</td>
<td><a href="http://www.w3.org/2006/time#">http://www.w3.org/2006/time#</a></td>
</tr>
<tr>
<td>wdt:</td>
<td><a href="http://www.wikidata.org/entity/">http://www.wikidata.org/entity/</a></td>
</tr>
</tbody>
</table>
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).

<table>
<thead>
<tr>
<th>From Element</th>
<th>Mapping relation</th>
<th>To Element</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>geo:Feature</td>
<td>rdfs:subClassOf</td>
<td>dcterms:Location</td>
<td>LOCN states that dcterms:Location &quot;represents any location, irrespective of size or other restriction&quot;. As such, it can be considered as a superclass of geo:Feature.</td>
</tr>
<tr>
<td>locn:Address</td>
<td>rdfs:subClassOf</td>
<td>geo:Feature</td>
<td>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.</td>
</tr>
<tr>
<td>geo:Geometry</td>
<td>rdfs:subClassOf</td>
<td>locn:Geometry</td>
<td>In LOCN, class locn:Geometry &quot;[...] defines the notion of geometry at the conceptual level, and it shall be encoded by using different formats&quot;. 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.</td>
</tr>
<tr>
<td>From Element</td>
<td>Mapping relation</td>
<td>To Element</td>
<td>Notes</td>
</tr>
<tr>
<td>--------------------</td>
<td>------------------------</td>
<td>-----------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>geo:hasGeometry</td>
<td>rdfs:subPropertyOf</td>
<td>locn:geometry</td>
<td>In LOCN, the usage note to property locn:geometry states that &quot;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 [...].&quot; 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.</td>
</tr>
</tbody>
</table>
## E.2 WGS84 Geo Positioning: an RDF vocabulary (POS)

**POS Source:** [http://www.w3.org/2003/01/geo/](http://www.w3.org/2003/01/geo/)

<table>
<thead>
<tr>
<th>From Element</th>
<th>Mapping relation</th>
<th>To Element</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>geo:SpatialObject</td>
<td>owl:equivalentClass</td>
<td>pos:SpatialThing</td>
<td>Both classes are unrestricted, essentially abstract classes</td>
</tr>
<tr>
<td>pos:Point</td>
<td>rdfs:subClassOf</td>
<td>geo:Geometry</td>
<td>Via pos:Point rdfs:subClassOf pos:SpatialThing but since pos:Point usage notes indicates direct positioning, it is a form of geometry</td>
</tr>
<tr>
<td>pos:Point</td>
<td>owl:equivalentClass</td>
<td>sf:Point</td>
<td></td>
</tr>
<tr>
<td>pos:lat_long</td>
<td>rdfs:subPropertyOf</td>
<td>geo:hasSerialization</td>
<td>A special datatype is not indicated for use with this property by POS, unlike GeoSPARQL’s geo:hasSerialization object literals</td>
</tr>
<tr>
<td>pos:location</td>
<td>rdfs:subPropertyOf</td>
<td>geo:hasGeometry</td>
<td></td>
</tr>
</tbody>
</table>
## E.3 W3C Activity Streams Vocabulary

AS Source: [https://www.w3.org/TR/activitystreams-vocabulary/](https://www.w3.org/TR/activitystreams-vocabulary/)

<table>
<thead>
<tr>
<th>From Element</th>
<th>Mapping relation</th>
<th>To Element</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>as:Place</td>
<td>owl:equivalentClass</td>
<td>geo:Feature</td>
<td>AS places are only defined for point geometries</td>
</tr>
<tr>
<td>as:accuracy</td>
<td>rdfs:subPropertyOf</td>
<td>geo:hasSpatialAccuracy</td>
<td>AS expresses the accuracy in percent</td>
</tr>
<tr>
<td>as:altitude</td>
<td></td>
<td></td>
<td>The altitude property can be expressed as a Z coordinate in GeoSPARQL-compatible literals</td>
</tr>
<tr>
<td>as:latitude</td>
<td>rdfs:subPropertyOf</td>
<td>geo:hasSerialization</td>
<td>AS defines the range of this property as xsd:float</td>
</tr>
<tr>
<td>as:longitude</td>
<td>rdfs:subPropertyOf</td>
<td>geo:hasSerialization</td>
<td>AS defines the range of this property as xsd:float</td>
</tr>
</tbody>
</table>
### E.4 Geonames Ontology (GN)

Geonames source: [http://www.geonames.org/ontology/documentation.html](http://www.geonames.org/ontology/documentation.html)

<table>
<thead>
<tr>
<th>From Element</th>
<th>Mapping relation</th>
<th>To Element</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>gn:Feature</td>
<td>owl:equivalentClass</td>
<td>geo:Feature</td>
<td>The GN class is defined as &quot;A feature described in geonames database...&quot;</td>
</tr>
<tr>
<td>gn:GeonamesFeature</td>
<td>rdfs:subClassOf</td>
<td>geo:Feature</td>
<td>The GN class’ definition reads &quot;A class of features&quot;</td>
</tr>
<tr>
<td>geo:Feature</td>
<td>rdfs:subClassOf</td>
<td>gn:Class</td>
<td></td>
</tr>
<tr>
<td>gn:locatedIn</td>
<td>owl:equivalentProperty</td>
<td>geo:sfWithin</td>
<td></td>
</tr>
<tr>
<td>gn:nearby</td>
<td>rdfs:subPropertyOf</td>
<td>geo:sfDisjoint</td>
<td>A gn:nearby B means A is not within or touching B. The only close SF property is disjoint</td>
</tr>
<tr>
<td>gn:neighbour</td>
<td>owl:equivalentProperty</td>
<td>geo:sfTouches</td>
<td></td>
</tr>
</tbody>
</table>
# E.5 NeoGeo Vocabulary


<table>
<thead>
<tr>
<th>From Element</th>
<th>Mapping relation</th>
<th>To Element</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>spatial:Feature</td>
<td>owl:equiventClass</td>
<td>geo:Feature</td>
<td></td>
</tr>
<tr>
<td>spatial:C</td>
<td>rdfs:subPropertyOf</td>
<td>geo:rcc8ec</td>
<td>Sub property not equivalent property since the NeoGeo property has more restrictive domain &amp; range</td>
</tr>
<tr>
<td>spatial:DR</td>
<td>rdfs:subPropertyOf</td>
<td>geo:rcc8dc</td>
<td></td>
</tr>
<tr>
<td>spatial:EC</td>
<td>rdfs:subPropertyOf</td>
<td>geo:rcc8ec</td>
<td></td>
</tr>
<tr>
<td>spatial:EQ</td>
<td>rdfs:subPropertyOf</td>
<td>geo:rcc8eq</td>
<td></td>
</tr>
<tr>
<td>spatial:NTPP</td>
<td>rdfs:subPropertyOf</td>
<td>geo:rcc8ntpp</td>
<td></td>
</tr>
<tr>
<td>spatial:NTPPi</td>
<td>rdfs:subPropertyOf</td>
<td>geo:rcc8ntppi</td>
<td></td>
</tr>
<tr>
<td>spatial:O</td>
<td>rdfs:subPropertyOf</td>
<td>geo:sfOverlaps</td>
<td></td>
</tr>
<tr>
<td>spatial:P</td>
<td>rdfs:subPropertyOf</td>
<td>geo:sfWithin</td>
<td></td>
</tr>
<tr>
<td>spatial:PO</td>
<td>rdfs:subPropertyOf</td>
<td>geo:rcc8po</td>
<td></td>
</tr>
<tr>
<td>spatial:PP</td>
<td>rdfs:subPropertyOf</td>
<td>geo:sfWithin</td>
<td></td>
</tr>
<tr>
<td>spatial:PPi</td>
<td>rdfs:subPropertyOf</td>
<td>geo:sfContains</td>
<td></td>
</tr>
<tr>
<td>spatial:Pi</td>
<td>rdfs:subPropertyOf</td>
<td>geo:sfContains</td>
<td></td>
</tr>
<tr>
<td>spatial:TPP</td>
<td>rdfs:subPropertyOf</td>
<td>geo:rcc8tpp</td>
<td></td>
</tr>
<tr>
<td>spatial:TPPi</td>
<td>rdfs:subPropertyOf</td>
<td>geo:rcc8tppi</td>
<td></td>
</tr>
<tr>
<td>geom:Geometry</td>
<td>owl:equiventClass</td>
<td>geo:Geometry</td>
<td></td>
</tr>
<tr>
<td>geom:BoundingBox</td>
<td>rdfs:subClassOf</td>
<td>geo:Geometry</td>
<td></td>
</tr>
<tr>
<td>geom:GeometryCollection</td>
<td>owl:equiventClass</td>
<td>geo:GeometryCollection</td>
<td></td>
</tr>
<tr>
<td>geom:LineString</td>
<td>owl:equiventClass</td>
<td>sf:LineString</td>
<td></td>
</tr>
<tr>
<td>geom:LinearRing</td>
<td>owl:equiventClass</td>
<td>sf:LinearRing</td>
<td></td>
</tr>
<tr>
<td>geom:MultiLineString</td>
<td>owl:equiventClass</td>
<td>sf:MultiLineString</td>
<td></td>
</tr>
<tr>
<td>geom:MultiPoint</td>
<td>owl:equiventClass</td>
<td>sf:MultiPoint</td>
<td></td>
</tr>
<tr>
<td>geom:MultiPolygon</td>
<td>owl:equiventClass</td>
<td>sf:MultiPolygon</td>
<td></td>
</tr>
<tr>
<td>geom:Polygon</td>
<td>owl:equiventClass</td>
<td>sf:Polygon</td>
<td></td>
</tr>
<tr>
<td>geom:Point</td>
<td>owl:equiventClass</td>
<td>sf:Point</td>
<td></td>
</tr>
</tbody>
</table>

*GeoSPARQL doesn’t have a BoundingBox class but has a generic Geometry class that is the range of the geo:hasBoundingBox property.*
• 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`
Juso contains mappings to GeoSPARQL but uses `owl:sameAs` which it should instead use `owl:equivalentClass`.

<table>
<thead>
<tr>
<th>From Element</th>
<th>Mapping relation</th>
<th>To Element</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>juso:SpatialThing</code></td>
<td><code>owl:equivalentClass</code></td>
<td><code>geo:SpatialObject</code></td>
</tr>
<tr>
<td><code>juso:Feature</code></td>
<td><code>owl:equivalentClass</code></td>
<td><code>geo:Feature</code></td>
</tr>
<tr>
<td><code>juso:Geometry</code></td>
<td><code>owl:equivalentClass</code></td>
<td><code>geo:Geometry</code></td>
</tr>
<tr>
<td><code>juso:Point</code></td>
<td><code>owl:equivalentClass</code></td>
<td><code>sf:Point</code></td>
</tr>
<tr>
<td><code>juso:geometry</code></td>
<td><code>owl:equivalentProperty</code></td>
<td><code>geo:hasGeometry</code></td>
</tr>
<tr>
<td><code>juso:parent</code></td>
<td><code>rdfs:subPropertyOf</code></td>
<td><code>geo:sfWithin</code></td>
</tr>
<tr>
<td><code>juso:political_division</code></td>
<td><code>rdfs:subPropertyOf</code></td>
<td><code>geo:sfContains</code></td>
</tr>
<tr>
<td><code>juso:within</code></td>
<td><code>owl:equivalentProperty</code></td>
<td><code>geo:sfWithin</code></td>
</tr>
</tbody>
</table>
E.7 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
- GeoSPARQL 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 spatio-temporal Feature may be indicated like this:

```owl
:fllooded-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, :fllooded-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 specialized 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:

```owl
:fllooded-area-x
   a geo:Feature;
   geo:hasGeometry [ a geo:Geometry;
                     ];
```

In the above example, :fllooded-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 specialized form of `time:TemporalEntity`.
In contrast to the first example, \texttt{:flooded-area-x} is inferred to be a spatio-temporal Feature but since it is the Geometry of \texttt{:flooded-area-x} that has a temporality, it is possible to describe other Geometries of \texttt{:flooded-area-x} with other temporalities.
## E.8 schema.org

**schema.org Source:** [https://schema.org](https://schema.org)

<table>
<thead>
<tr>
<th>From Element</th>
<th>Mapping relation</th>
<th>To Element</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>geo:Geometry</td>
<td>rdfs:subClassOf</td>
<td>sdo:GeoShape</td>
<td>A GeoShape can have various literal geometry representations</td>
</tr>
<tr>
<td>sdo:GeospatialGeometry</td>
<td>owl:equivalentClass</td>
<td>geo:SpatialObject</td>
<td>Since sdo:GeospatialGeometry is the domain of SimpleFeature-like properties and a superclass of GeoShape</td>
</tr>
<tr>
<td>sdo:GeoCoordinates</td>
<td>rdfs:subClassOf</td>
<td>geo:Geometry</td>
<td>GoCoordinates uses direct lat, long, elevation etc properties to indicate position, not a geometry serialization but it is nevertheless a form of a Geometry</td>
</tr>
<tr>
<td>sdo:geo</td>
<td>rdfs:subPropertyOf</td>
<td>geo:hasGeometry</td>
<td></td>
</tr>
<tr>
<td>sdo:geoCoveredBy</td>
<td>owl:equivalentProperty</td>
<td>geo:ehCoveredBy</td>
<td></td>
</tr>
<tr>
<td>sdo:geoCovers</td>
<td>owl:equivalentProperty</td>
<td>geo:ehCovers</td>
<td></td>
</tr>
<tr>
<td>sdo:geoCrosses</td>
<td>owl:equivalentProperty</td>
<td>geo:sfCrosses</td>
<td></td>
</tr>
<tr>
<td>sdo:geoDisjoint</td>
<td>owl:equivalentProperty</td>
<td>geo:sfDisjoint</td>
<td></td>
</tr>
<tr>
<td>sdo:geoEquals</td>
<td>owl:equivalentProperty</td>
<td>geo:sfEquals</td>
<td></td>
</tr>
<tr>
<td>sdo:geoIntersects</td>
<td>owl:equivalentProperty</td>
<td>geo:sfIntersects</td>
<td></td>
</tr>
<tr>
<td>sdo:geoOverlaps</td>
<td>owl:equivalentProperty</td>
<td>geo:sfOverlaps</td>
<td></td>
</tr>
<tr>
<td>sdo:geoTouches</td>
<td>owl:equivalentProperty</td>
<td>geo:sfTouches</td>
<td></td>
</tr>
<tr>
<td>sdo:geoWithin</td>
<td>owl:equivalentProperty</td>
<td>geo:sfWithin</td>
<td></td>
</tr>
<tr>
<td>sdo:geoMidpoint</td>
<td>owl:equivalentProperty</td>
<td>geo:hasCentroid</td>
<td></td>
</tr>
<tr>
<td>sdo:Landform</td>
<td>rdfs:subClassOf</td>
<td>geo:Feature</td>
<td></td>
</tr>
</tbody>
</table>
E.9 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.10 DCMI Metadata Terms (DCTERMS)

DCTERMS Source: https://www.dublincore.org/specifications/dublin-core/dcми-terms/

<table>
<thead>
<tr>
<th>From Element</th>
<th>Mapping relation</th>
<th>To Element</th>
<th>Notes</th>
</tr>
</thead>
</table>
| 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:spatial: "Spatial characteristics of the resource". The range of this property includes a dcterms:Location, so it is a property for indicating a geo:Feature, for which GeoSPARQL has no equivalent, but perhaps also for indicating a geo:Geometry, thus the subPropertyOf mapping above.
- dcterms:coverage: "The spatial or temporal topic of the resource, spatial applicability of the resource, or jurisdiction under which the resource is relevant". This is a more generic form of dcterms:spatial but, since there is no direct GeoSPARQL mapping for dcterms:spatial, there is no direct mapping for this property either.

DCTERMS-related geometry literals, such as the DCMI Box Encoding Scheme [1] and the DCMI Point Encoding Scheme [2] 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:

```sparql
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 ;
    .
    ] ;
```

E.11 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 characterizes things - spatial objects - which are a kind of Entity but does not deal with events (Activity) or things with agency (Agent).

<table>
<thead>
<tr>
<th>From Element</th>
<th>Mapping relation</th>
<th>To Element</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>geo:SpatialObjectCollection</td>
<td>rdfs:subClassOf</td>
<td>prov:Collection</td>
<td>PROV’s class is a generic collection class and GeoSPARQL’s property is clearly a specialized form of it that may only consist of certain class instances (geo:SpatialObject)</td>
</tr>
<tr>
<td>geo:SpatialObject</td>
<td>rdfs:subClassOf</td>
<td>prov:Entity</td>
<td>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.”</td>
</tr>
<tr>
<td>geo:Feature</td>
<td>rdfs:subClassOf</td>
<td>prov:Location</td>
<td>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”</td>
</tr>
</tbody>
</table>

* The PROV property prov:atLocation indicates prov:Location instances, which may be geo:Feature instances, but GeoSPARQL has no property to indicate a geo:Feature, so no mapping is possible. Indicating features is commonly done in ontologies which use GeoSPARQL but not
within GeoSPARQL.

- Derivative relations between GeoSPARQL objects could be modelled using PROV, for instance a BoundingBox may be indicated as having been derived from a Polygon like this:

```
@prefix bounding-box-y : http://example.org/bounding-box-y .
@prefix polygon-x : http://example.org/polygon-x .

bounding-box-y prov:wasDerivedFrom polygon-x .
```
### E.12 WikiData

<table>
<thead>
<tr>
<th>From Element</th>
<th>Mapping relation</th>
<th>To Element</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>wdt:P625</td>
<td>owl:equivalentProperty</td>
<td>geo:asWKT</td>
<td>The Wikidata description of this property labeled &quot;coordinate location&quot; note that &quot;For Earth, please note that only WGS84 coordinating system is supported at the moment&quot; but that is a system limit, not an ontological one.</td>
</tr>
<tr>
<td>wdt:P3896</td>
<td>owl:propertyChainAxiom</td>
<td>(geo:hasGeometry geo:asGeoJSON)</td>
<td>This Wikidata property labeled &quot;geoshape&quot; indicated GeoJSON geometry literal content for a Feature, but it allows information other than just Geometry in the GeoJSON whereas GeoSPARQL does not.</td>
</tr>
<tr>
<td>wdt:P3096</td>
<td>owl:propertyChainAxiom</td>
<td>(geo:hasGeometry geo:asKML)</td>
<td>This Wikidata property labeled &quot;KML File&quot; 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.</td>
</tr>
<tr>
<td>From Element</td>
<td>Mapping relation</td>
<td>To Element</td>
<td>Notes</td>
</tr>
<tr>
<td>--------------</td>
<td>------------------</td>
<td>--------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>wd:Q82794</td>
<td>rdfs:subClassOf</td>
<td>geo:Feature</td>
<td>The Wikidata class is labeled &quot;geographic region&quot; 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.</td>
</tr>
<tr>
<td>wd:Q618123</td>
<td>owl:equivalentClass</td>
<td>geo:Feature</td>
<td>The Wikidata class is labeled &quot;geographical feature&quot; and thus corresponds to geo:Feature.</td>
</tr>
<tr>
<td>wd:Q25404640</td>
<td>owl:equivalentClass</td>
<td>geo:SpatialObject</td>
<td>The Wikidata class is labeled &quot;spatial object&quot; and thus corresponds to geo:SpatialObject.</td>
</tr>
<tr>
<td>wdt:P150</td>
<td>rdfs:subPropertyOf</td>
<td>geo:sfContains</td>
<td>The Wikidata property is labeled &quot;contains administrative territorial entity&quot; but also alternatively labeled &quot;contains&quot;, &quot;has districts&quot; and others. There are likely many other specialized forms of geo:sfContains and geo:sfWithin in Wikidata.</td>
</tr>
<tr>
<td>From Element</td>
<td>Mapping relation</td>
<td>To Element</td>
<td>Notes</td>
</tr>
<tr>
<td>----------------------</td>
<td>------------------</td>
<td>--------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>geo:sfWithin</td>
<td>rdfs:subPropertyOf</td>
<td>wdt:P361</td>
<td>The Wikidata property is labeled &quot;part of&quot; and is sometimes used to indicate Feature parthood. There are likely 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 &quot;has part&quot;</td>
</tr>
<tr>
<td>geo:sfContains</td>
<td>rdfs:subPropertyOf</td>
<td>wdt:Q65964571</td>
<td>The property labeled &quot;has part&quot; is the inverse of wdt:P361 (see above)</td>
</tr>
<tr>
<td>wdt:P131</td>
<td>rdfs:subPropertyOf</td>
<td>geo:sfContains</td>
<td>The Wikidata property is labeled &quot;located in the administrative territorial entity&quot; and is essentially the inverse of wdt:P150 (described above)</td>
</tr>
<tr>
<td>wdt:P706</td>
<td>rdfs:subPropertyOf</td>
<td>geo:sfWithin</td>
<td>The Wikidata property is labeled &quot;located in/on physical feature&quot; and is indicated for use with a &quot;(geo)physical feature&quot; and not to be used for administrative features where wdt:P131 (see above) should be</td>
</tr>
<tr>
<td>From Element</td>
<td>Mapping relation</td>
<td>To Element</td>
<td>Notes</td>
</tr>
<tr>
<td>--------------</td>
<td>------------------</td>
<td>------------</td>
<td>-------</td>
</tr>
<tr>
<td>wdt:P4688</td>
<td>rdfs:subClassOf</td>
<td>geo:Feature</td>
<td>The Wikidata class is labeled &quot;geomorphological unit&quot; and is one of many Wikidata feature classes that could be expressed as a subclass of geo:Feature. More specialized geological unit examples are wd:Q5107 &quot;continent&quot; and wdt:P4552 &quot;mountain range&quot;.</td>
</tr>
<tr>
<td>wdt:P2046</td>
<td>owl:equivalentProperty</td>
<td>geo:hasArea</td>
<td>The Wikidata property is labeled &quot;area&quot;. It indicates a microformat - NUMBER + SPACE + ALLOWED_UNIT_LABEL - with a fixed set of ALLOWED_UNIT_LABELs to present values and units of measure.</td>
</tr>
</tbody>
</table>
E.13 OpenStreetMap Ontologies

There are several approaches to make OpenStreetMap data accessible in the Linked Open Data cloud.

E.13.1 LinkedGeoData

LinkedGeoData emerged from a research project connecting OpenStreetMap representations to an ontology model. In this model, specific values of OpenStreetMap tags, e.g. the values of amenity tags are converted to `owl:Class` representations using an automated process. Every class defined in this way represented a `geo:Feature` and is linked to either a Geometry or a latitude longitude representation. Hence, every linked geodata class can be considered a `geo:Feature` in the sense of GeoSPARQL.

<table>
<thead>
<tr>
<th>From Element</th>
<th>Mapping relation</th>
<th>To Element</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Any LGD Class</td>
<td><code>rdfs:subClassOf</code></td>
<td><code>geo:Feature</code></td>
<td>Any class defined in the LinkedGeoData ontology is a subclass of <code>geo:Feature</code></td>
</tr>
</tbody>
</table>

E.13.2 OpenStreetMap RDF (Sophox)

https://wiki.openstreetmap.org/wiki/Sophox#How_OSM_data_is_stored

<table>
<thead>
<tr>
<th>From Element</th>
<th>Mapping relation</th>
<th>To Element</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>osmm:loc</code></td>
<td><code>owl:equivalentProperty</code></td>
<td><code>geo:asWKT</code></td>
<td>The OpenStreetMap RDF property <code>osmm:loc</code> includes WKTliterals which depending on the type of the subject instance describe an OSM node or the centroid of a way or OSM relation</td>
</tr>
<tr>
<td><code>osmm:type 'n'</code></td>
<td><code>owl:equivalentClass</code></td>
<td><code>sf:Point</code></td>
<td>The OpenStreetMap RDF property <code>osmm:type</code> with value 'n' describes an OSM Node which is equivalent to a <code>sf:Point</code></td>
</tr>
<tr>
<td><code>osmm:type 'w'</code></td>
<td><code>owl:equivalentClass</code></td>
<td><code>sf:LineString</code></td>
<td>The OpenStreetMap RDF property <code>osmm:type</code> with value 'w' describes an OSM Way which is equivalent to a <code>sf:LineString</code></td>
</tr>
<tr>
<td>From Element</td>
<td>Mapping relation</td>
<td>To Element</td>
<td>Notes</td>
</tr>
<tr>
<td>--------------</td>
<td>------------------</td>
<td>-----------------</td>
<td>----------------------------------------------------------------------</td>
</tr>
<tr>
<td>osmm:type 'r'</td>
<td>owl:equivalentClass</td>
<td>sf:GeometryCollection</td>
<td>The OpenStreetMap RDF property osmm:type with value 'r' describes an OSM relation Way which is equivalent to a sf:GeometryCollection</td>
</tr>
<tr>
<td>osmm:has</td>
<td>owl:equivalentProperty</td>
<td>geo:sfContains, geo:ehContains, geo:rcc8ntpp</td>
<td>The OpenStreetMap RDF property osmm:has describes that a relation contains a way or that a way contains a node</td>
</tr>
<tr>
<td>osmm:isClosed true</td>
<td>owl:equivalentClass</td>
<td>sf:Polygon</td>
<td>The OpenStreetMap RDF property osmm:isClosed indicates whether a Way is closed, i.e. if it constitutes a Polygon</td>
</tr>
<tr>
<td>osmm:isClosed false</td>
<td>owl:equivalentClass</td>
<td>sf:LineString</td>
<td>The OpenStreetMap RDF property osmm:isClosed indicates whether a Way is closed, i.e. if it constitutes a Polygon</td>
</tr>
</tbody>
</table>

### E.13.3 Routable Tiles Ontology

[https://github.com/openplannerteam/routable-tiles-ontology](https://github.com/openplannerteam/routable-tiles-ontology)

<table>
<thead>
<tr>
<th>From Element</th>
<th>Mapping relation</th>
<th>To Element</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>osm:Element</td>
<td>owl:equivalentClass</td>
<td>geo:Geometry</td>
<td>The class osm:Element is equivalent to a geo:Geometry</td>
</tr>
<tr>
<td>osm:Node</td>
<td>owl:equivalentClass</td>
<td>sf:Point</td>
<td>The class osm:Node is equivalent to a sf:Point</td>
</tr>
<tr>
<td>osm:Way</td>
<td>owl:equivalentClass</td>
<td>sf:LineString</td>
<td>The class osm:Way is equivalent to a sf:LineString</td>
</tr>
<tr>
<td>osm:Relation</td>
<td>owl:equivalentClass</td>
<td>sf:GeometryCollection</td>
<td>The class osm:Relation is equivalent to a sf:GeometryCollection</td>
</tr>
</tbody>
</table>
E.14 Ordnance Survey UK Spatial Ontology

http://www.ordnancesurvey.co.uk/legacy/ontologies/spatialrelations.owl &
http://www.ordnancesurvey.co.uk/legacy/ontologies/geometry.owl

**NOTE** 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 recognize 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."

<table>
<thead>
<tr>
<th>From Element</th>
<th>Mapping relation</th>
<th>To Element</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>spatialuk:contains</td>
<td>owl:equivalentProperty</td>
<td>geo:sfContains</td>
<td></td>
</tr>
<tr>
<td>spatialuk:disjoint</td>
<td>owl:equivalentProperty</td>
<td>geo:sfDisjoint</td>
<td></td>
</tr>
<tr>
<td>spatialuk:easting</td>
<td>owl:equivalentProperty</td>
<td>-</td>
<td>Distance in metres east of National Grid origin</td>
</tr>
<tr>
<td>spatialuk:equals</td>
<td>owl:equivalentProperty</td>
<td>geo:sfEquals</td>
<td></td>
</tr>
<tr>
<td>spatialuk:northing</td>
<td>owl:equivalentProperty</td>
<td>-</td>
<td>Distance in metres north of National Grid origin</td>
</tr>
<tr>
<td>spatialuk:touches</td>
<td>owl:equivalentProperty</td>
<td>geo:sfTouches</td>
<td></td>
</tr>
<tr>
<td>spatialuk:within</td>
<td>owl:equivalentProperty</td>
<td>geo:sfWithin</td>
<td></td>
</tr>
<tr>
<td>spatialukgeom:Abstract Geometry</td>
<td>owl:equivalentProperty</td>
<td>geo:Geometry</td>
<td></td>
</tr>
<tr>
<td>spatialukgeom:extent</td>
<td>owl:equivalentProperty</td>
<td>geo:hasGeometry</td>
<td>The range of spatialukgeom:extent is constrained to 2D geometries</td>
</tr>
<tr>
<td>spatialukgeom:asGML</td>
<td>owl:equivalentProperty</td>
<td>geo:asGML</td>
<td>The properties are equivalent, but the range of <code>spatialukgeom:asGML</code> is more general: An rdf:XMLLiteral</td>
</tr>
</tbody>
</table>

- `spatialuk:easting` describes a latitude coordinate east of the national UK grid and GeoSPARQL does not contain modelling of individual coordinate reference system elements
- `spatialuk:northing` describes a longitude coordinate north of the national UK grid so, as above, has not GeoSPARQL equivalent
E.15 CIDOC CRM Geo

<table>
<thead>
<tr>
<th>From Element</th>
<th>Mapping relation</th>
<th>To Element</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>cidoc:SP1_PhenomenalSpaceTimeVolume</td>
<td>rdfs:subClassOf</td>
<td>geo:Feature</td>
<td>The CIDOC CRMgeo class SP1_PhenomenalSpaceTimeVolume is a subclass of geo:Feature as described in the CRMgeo 1.2 specification document.</td>
</tr>
<tr>
<td>cidoc:SP2_PhenomenalPlace</td>
<td>rdfs:subClassOf</td>
<td>geo:Feature</td>
<td>The CIDOC CRMgeo class SP2_PhenomenalPlace is a subclass of geo:Feature as described in the CRMgeo 1.2 specification document.</td>
</tr>
<tr>
<td>cidoc:SP5_GeometricPlaceExpression</td>
<td>rdfs:subClassOf</td>
<td>geo:Geometry</td>
<td>The CIDOC CRMgeo class SP5_GeometricPlaceExpression is a subclass of geo:Geometry as described in the CRMgeo 1.2 specification document.</td>
</tr>
<tr>
<td>cidoc:SP6_DeclarativePlace</td>
<td>rdfs:subClassOf</td>
<td>geo:Geometry</td>
<td>The CIDOC CRMgeo class SP6_DeclarativePlace is a subclass of geo:Geometry as described in the CRMgeo 1.2 specification document.</td>
</tr>
<tr>
<td>cidoc:SP7_DeclarativePlace</td>
<td>rdfs:subClassOf</td>
<td>geo:Geometry</td>
<td>The CIDOC CRMgeo class SP7_DeclarativePlace is a subclass of geo:Geometry as described in the CRMgeo 1.2 specification document.</td>
</tr>
<tr>
<td>From Element</td>
<td>Mapping relation</td>
<td>To Element</td>
<td>Notes</td>
</tr>
<tr>
<td>------------------------------</td>
<td>------------------</td>
<td>--------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>cidoc:SP10_DeclarativeTimeSpan</td>
<td>rdfs:subClassOf</td>
<td>geo:Geometry</td>
<td>The CIDOC CRMgeo class SP10_DeclarativeTimeSpan is a subclass of geo:Geometry as described in the CRMgeo 1.2 specification document.</td>
</tr>
<tr>
<td>cidoc:SP14_TimeExpression</td>
<td>rdfs:subClassOf</td>
<td>geo:Geometry</td>
<td>The CIDOC CRMgeo class SP14_TimeExpression is a subclass of geo:Geometry as described in the CRMgeo 1.2 specification document.</td>
</tr>
<tr>
<td>cidoc:SP15_Geometry</td>
<td>rdfs:subClassOf</td>
<td>geo:Geometry</td>
<td>The CIDOC CRMgeo class SP15_Geometry is a subclass of geo:Geometry as described in the CRMgeo 1.2 specification document.</td>
</tr>
</tbody>
</table>
## E.16 Basic Formal Ontology (BFO)

BFO Source: [https://basic-formal-ontology.org/bfo-2020.html](https://basic-formal-ontology.org/bfo-2020.html), and from there, an OWL ontology of BFO2020 at [https://github.com/BFO-ontology/BFO-2020](https://github.com/BFO-ontology/BFO-2020)

<table>
<thead>
<tr>
<th>From Element</th>
<th>Mapping relation</th>
<th>To Element</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>geo:SpatialObject</td>
<td>rdfs:subClassOf</td>
<td>obo:BFO_000004</td>
<td>BFO's &quot;independent continuant&quot; is the superclass of &quot;material entity&quot; &amp; &quot;immaterial entity&quot; which are mapped to Feature &amp; Geometry respectively, so at least some independent continuants must be Spatial Objects</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&quot;independent continuant&quot;</td>
<td></td>
</tr>
<tr>
<td>geo:Geometry</td>
<td>rdfs:subClassOf</td>
<td>obo:BFO_000006</td>
<td>BFO's &quot;spatial region&quot; class is described as a &quot;spatial projection of a portion of spacetime&quot; so Geometry appears to be a subclass of this as it's &quot;A coherent set of direct positions in space&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&quot;spatial region&quot;</td>
<td></td>
</tr>
<tr>
<td>geo:Geometry</td>
<td>rdfs:subClassOf</td>
<td>obo:IAO_000003</td>
<td>BFO's &quot;information content entity&quot; class is described as &quot;an entity that represents information about some other entity&quot;, so Geometry appears to be subclass of this as well as &quot;spatial region&quot; since in GeoSPARQL, Geometry gives the details of the spatial projection of a Feature.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&quot;information content entity&quot;</td>
<td></td>
</tr>
<tr>
<td>obo:BFO_000004</td>
<td>rdfs:subClassOf</td>
<td>geo:Feature</td>
<td>A BFO &quot;material entity&quot; is something that &quot;has some portion of matter as continuant part&quot; and some Features are such, however Features may be imaginary too</td>
</tr>
<tr>
<td>&quot;material entity&quot;</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>obo:BFO_000029</td>
<td>rdfs:subClassOf</td>
<td>geo:Feature</td>
<td>BFO's sites either cover the same areas as, or have locations determined in relation to, material entities, so sites are Features but not necessarily the other way around</td>
</tr>
<tr>
<td>&quot;site&quot;</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>geo:hasGeometry</td>
<td>rdfs:subPropertyOf</td>
<td>obo:BFO_000021</td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&quot;occupies spatial region at all times&quot;</td>
<td></td>
</tr>
<tr>
<td>geo:hasGeometry</td>
<td>rdfs:subPropertyOf</td>
<td>obo:BFO_000021</td>
<td>A transitive mapping from the mapping above. Temporal qualification can be used with GeoSPARQL, see the OWL TIME alignment.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&quot;occupies spatial region at some time&quot;</td>
<td></td>
</tr>
<tr>
<td>From Element</td>
<td>Mapping relation</td>
<td>To Element</td>
<td>Notes</td>
</tr>
<tr>
<td>--------------</td>
<td>------------------</td>
<td>------------</td>
<td>-------</td>
</tr>
<tr>
<td>geo:sfWithin</td>
<td>rdfs:subPropertyOf</td>
<td>obo:BFO_0000082 &quot;located in at all times&quot;</td>
<td>The BFO property &quot;located in at all times&quot; 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: &quot;...at all times&quot;</td>
</tr>
<tr>
<td>geo:sfWithin</td>
<td>rdfs:subPropertyOf</td>
<td>obo:BFO_0000171 &quot;located in at some time&quot;</td>
<td>A transitive mapping from the mapping above. Temporal qualification can be used with GeoSPARQL, see the OWL TIME alignment.</td>
</tr>
<tr>
<td>obo:BFO_0000066 &quot;occurs in&quot;</td>
<td>rdfs:range</td>
<td>geo:SpatialObject</td>
<td>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 &quot;spatial region&quot; (geo:Geometry) range value but from GeoSPARQL’s point of view, it could also be used with a geo:Feature where the &quot;in&quot; would be taken to be within the feature’s geometry, so the superclass of feature and geometry is given as the range</td>
</tr>
<tr>
<td>obo:BFO_0000216 &quot;spatially projects onto at some time&quot;</td>
<td>rdfs:range</td>
<td>geo:SpatialObject</td>
<td>The reasoning is the same as for &quot;occurs in&quot;</td>
</tr>
</tbody>
</table>

- BFO distinguishes between continuants & occurrants, which spatial region & spatiotemporal region are subclasses of, respectively. GeoSPARQL has no handling of temporality, so cannot yet map to any continuants

  - a future version of GeoSPARQL that handled spatio-temporal Features could perhaps claim that geo:Feature is a rdfs:subClassOf obo:BFO_0000011 "spatiotemporal region", however inconsistencies from this mapping will occur due to the current Feature/"spatial region" mapping above and this will need to be handled
Annex B: 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 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 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.

```
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.

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:

```
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.

<table>
<thead>
<tr>
<th>Query Parameter</th>
<th>Example</th>
<th>SPARQL Expression</th>
<th>Example</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>limit</td>
<td>limit=5</td>
<td>LIMIT</td>
<td>LIMIT 5</td>
<td></td>
</tr>
<tr>
<td>offset</td>
<td>offset=10</td>
<td>OFFSET</td>
<td>OFFSET 10</td>
<td></td>
</tr>
<tr>
<td>bbox</td>
<td>bbox=160.6,-55.95,-170,-25.89</td>
<td>FILTER(geo:sfIntersects())</td>
<td>FILTER(geo:sfIntersects(&quot;POLYGON((160.6-55.95,160.6-25.89,-170-25.89,-170-55.95,160.6-55.95))&quot;^^geo:wktLiteral))</td>
<td>WKT does not define a type boundingbox, therefore a bbox is converted to a Polygon</td>
</tr>
<tr>
<td>datetime</td>
<td>datetime=2018-02-12T23:20:52Z</td>
<td>-</td>
<td>-</td>
<td>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]</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>CQL2 literal</th>
<th>Examples</th>
<th>(Geo)SPARQL literal</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>String</td>
<td>&quot;This is a string&quot;</td>
<td>xsd:string</td>
<td>&quot;This is a string&quot;^^xsd:string</td>
</tr>
<tr>
<td>CQL2 literal</td>
<td>Examples</td>
<td>(Geo)SPARQL literal</td>
<td>Examples</td>
</tr>
<tr>
<td>--------------</td>
<td>----------</td>
<td>---------------------</td>
<td>----------</td>
</tr>
<tr>
<td>Number</td>
<td>-100 3.14159</td>
<td>xsd:int, xsd:integer, xsd:double</td>
<td>&quot;-100&quot;^^xsd:integer &quot;3.14159&quot;^^xsd:double</td>
</tr>
<tr>
<td>Boolean</td>
<td>true false</td>
<td>xsd:boolean</td>
<td>&quot;true&quot;^^xsd:boolean &quot;false&quot;^^xsd:boolean</td>
</tr>
<tr>
<td>Spatial Geometry (WKT)</td>
<td>POINT(1 1)</td>
<td>WKT Literal</td>
<td>&quot;POINT(1 1)&quot;^^geo:wktLiteral</td>
</tr>
<tr>
<td>Spatial Geometry (JSON)</td>
<td>{&quot;type&quot;: &quot;Point&quot;, &quot;coordinates&quot;:[1,1]}</td>
<td>GeoJSON Literal</td>
<td>&quot;{&quot;type&quot;: &quot;Point&quot;, &quot;coordinates&quot;:[1,1]}&quot;^^geo:geoJSONLiteral</td>
</tr>
</tbody>
</table>

### 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.

<table>
<thead>
<tr>
<th>Property Reference</th>
<th>Triple pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td>name=&quot;OGC&quot;</td>
<td>?item my:name &quot;OGC&quot;^^xsd:string</td>
</tr>
<tr>
<td>number=5</td>
<td>?item my:number &quot;5&quot;^^xsd:integer</td>
</tr>
<tr>
<td>number&gt;5</td>
<td>?item my:number ?number . FILTER(?number&gt;5)</td>
</tr>
</tbody>
</table>

### 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.

<table>
<thead>
<tr>
<th>Comparison predicate</th>
<th>Triple pattern</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>name=&quot;OGC&quot;</td>
<td>?item my:name &quot;OGC&quot;^^xsd:string</td>
<td>Equality statements can be converted to a triple pattern</td>
</tr>
<tr>
<td>number=5</td>
<td>?item my:number &quot;5&quot;^^xsd:integer</td>
<td></td>
</tr>
<tr>
<td>number&gt;5</td>
<td>?item my:number ?number . FILTER(?number&gt;5)</td>
<td>Arithmetic comparisons (&lt;,&gt;,&gt;=,⇐) are converted to filter expressions</td>
</tr>
<tr>
<td>Comparison predicate</td>
<td>Triple pattern</td>
<td>Comment</td>
</tr>
<tr>
<td>----------------------</td>
<td>--------------------------------------------------------------------------------</td>
<td>--------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>number BETWEEN 5 AND 10</td>
<td><code>?item my:number ?number . FILTER(?number&gt;=5 &amp;&amp; ?number&lt;=10)</code></td>
<td>BETWEEN statements are converted to arithmetic expressions</td>
</tr>
<tr>
<td>name IN (&quot;OGC&quot;,&quot;W3C&quot;)</td>
<td><code>?item my:name IN (&quot;OGC&quot;, &quot;W3C&quot;)</code></td>
<td>IN statements may also be expressed using SPARQL VALUES statements</td>
</tr>
<tr>
<td>name IS NOT NULL</td>
<td>EXISTS {?item my:name ?name { } }</td>
<td>NOT NULL statements are converted to EXIST statements</td>
</tr>
<tr>
<td>name LIKE &quot;OGC.&quot;</td>
<td><code>?item my:name ?name . FILTER(regex(?name, &quot;OGC.&quot; , &quot;i&quot; ))</code></td>
<td>LIKE statements are converted to SPARQL regex filters</td>
</tr>
<tr>
<td>INTERSECTS(geometry1, geometry2)</td>
<td><code>FILTER(geof:sfIntersects(?geometry1,?geometry2))</code></td>
<td>The INTERSECTS filter statement is converted to a GeoSPARQL FILTER statement</td>
</tr>
</tbody>
</table>

There is no direct GeoSPARQL equivalent to a CRS-based CQL filter, however certain GeoSPARQL geometry literals have explicit 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.

<table>
<thead>
<tr>
<th>CQL2 Filter Expression</th>
<th>GeoSPARQL Filter Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONTAINS(geometry1,geometry2)</td>
<td><code>FILTER(geof:sfContains(?geometry1,?geometry2))</code></td>
</tr>
<tr>
<td>CROSSES(geometry1,geometry2)</td>
<td><code>FILTER(geof:sfCloses(?geometry1,?geometry2))</code></td>
</tr>
<tr>
<td>DISJOINT(geometry1,geometry2)</td>
<td><code>FILTER(geof:sfDisjoint(?geometry1,?geometry2))</code></td>
</tr>
<tr>
<td>EQUALS(geometry1,geometry2)</td>
<td><code>FILTER(geof:sfEquals(?geometry1,?geometry2))</code></td>
</tr>
<tr>
<td>INTERSECTS(geometry1,geometry2)</td>
<td><code>FILTER(geof:sfIntersects(?geometry1,?geometry2))</code></td>
</tr>
<tr>
<td>OVERLAPS(geometry1,geometry2)</td>
<td><code>FILTER(geof:sfOverlaps(?geometry1,?geometry2))</code></td>
</tr>
<tr>
<td>TOUCHES(geometry1,geometry2)</td>
<td><code>FILTER(geof:sfTouches(?geometry1,?geometry2))</code></td>
</tr>
<tr>
<td>WITHIN(geometry1,geometry2)</td>
<td><code>FILTER(geof:sfWithin(?geometry1,?geometry2))</code></td>
</tr>
</tbody>
</table>

### F.2.6 Temporal Operators

Temporal operators are not part of the GeoSPARQL standard.
<table>
<thead>
<tr>
<th>CQL2 Filter Expression</th>
<th>GeoSPARQL Filter Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>beginTime AFTER 1969-07-16T13:32:00Z</td>
<td>N/A</td>
</tr>
<tr>
<td>beginTime BEFORE 1969-07-16T13:32:00Z</td>
<td>N/A</td>
</tr>
<tr>
<td>beginTime BEGINS 1969-07-16T13:32:00Z</td>
<td>N/A</td>
</tr>
<tr>
<td>beginTime BEGUNBY 1969-07-16T13:32:00Z</td>
<td>N/A</td>
</tr>
<tr>
<td>beginTime DURING 1969-07-16T13:32:00Z</td>
<td>N/A</td>
</tr>
<tr>
<td>beginTime ENDEDBY 1969-07-16T13:32:00Z</td>
<td>N/A</td>
</tr>
<tr>
<td>beginTime ENDS 1969-07-16T13:32:00Z</td>
<td>N/A</td>
</tr>
<tr>
<td>beginTime MEETS 1969-07-16T13:32:00Z</td>
<td>N/A</td>
</tr>
<tr>
<td>beginTime METBY 1969-07-16T13:32:00Z</td>
<td>N/A</td>
</tr>
<tr>
<td>beginTime OVERLAPPEDBY 1969-07-16T13:32:00Z</td>
<td>N/A</td>
</tr>
<tr>
<td>beginTime TCONTAINS 1969-07-16T13:32:00Z</td>
<td>N/A</td>
</tr>
<tr>
<td>beginTime TEQUALS 1969-07-16T13:32:00Z</td>
<td>N/A</td>
</tr>
<tr>
<td>beginTime TOVERLAPS 1969-07-16T13:32:00Z</td>
<td>N/A</td>
</tr>
</tbody>
</table>

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 [OGCSFACA][ISO19125-1] to GeoSPARQL.

<table>
<thead>
<tr>
<th>Simple Features for SQL</th>
<th>GeoSPARQL Equivalent</th>
<th>Since GeoSPARQL</th>
<th>Related Property Available</th>
<th>Since GeoSPARQL</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>2.1.1.1 Basic Methods on Geometry</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dimension(): Double</td>
<td>geof:dimension</td>
<td>-</td>
<td>geo:dimension</td>
<td>1.0</td>
</tr>
<tr>
<td>GeometryType(): Integer</td>
<td>Class of geometry instance</td>
<td>1.0</td>
<td>N/A</td>
<td>-</td>
</tr>
<tr>
<td>SRID(): Integer</td>
<td>geof:getSRID</td>
<td>1.0</td>
<td>N/A</td>
<td>-</td>
</tr>
<tr>
<td>Envelope(): Geometry</td>
<td>geof:envelope</td>
<td>1.0</td>
<td>geo:hasBoundingBox</td>
<td>1.1</td>
</tr>
<tr>
<td>AsText(): String</td>
<td>geof:asWKT</td>
<td>1.1</td>
<td>geo:asWKT</td>
<td>1.0</td>
</tr>
<tr>
<td>AsBinary(): Binary</td>
<td>N/A</td>
<td>-</td>
<td>N/A</td>
<td>-</td>
</tr>
<tr>
<td>IsEmpty(): Integer</td>
<td>geof:isEmpty</td>
<td>-</td>
<td>geo:isEmpty</td>
<td>1.0</td>
</tr>
<tr>
<td>IsSimple(): Integer</td>
<td>geof:isEmpty</td>
<td>-</td>
<td>geo:isSimple</td>
<td>1.0</td>
</tr>
<tr>
<td>Boundary(): Geometry</td>
<td>geof:boundary</td>
<td>1.0</td>
<td>N/A</td>
<td>-</td>
</tr>
<tr>
<td><strong>2.1.1.2 Spatial Relations</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Equals(anotherGeometry: Geometry): Integer</td>
<td>geof:sfEquals</td>
<td>1.0</td>
<td>geo:sfEquals</td>
<td>1.0</td>
</tr>
<tr>
<td>Disjoint(anotherGeometry: Geometry): Integer</td>
<td>geof:sfDisjoint</td>
<td>1.0</td>
<td>geo:sfDisjoint</td>
<td>1.0</td>
</tr>
<tr>
<td>Intersects(anotherGeometry: Geometry): Integer</td>
<td>geof:sfIntersects</td>
<td>1.0</td>
<td>geo:sfIntersects</td>
<td>1.0</td>
</tr>
<tr>
<td>Touches(anotherGeometry: Geometry): Integer</td>
<td>geof:sfTouches</td>
<td>1.0</td>
<td>geo:sfTouches</td>
<td>1.0</td>
</tr>
<tr>
<td>Simple Features for SQL</td>
<td>GeoSPARQL Equivalent</td>
<td>Since GeoSPARQL</td>
<td>Related Property Available</td>
<td>Since GeoSPARQL</td>
</tr>
<tr>
<td>------------------------</td>
<td>-----------------------</td>
<td>----------------</td>
<td>---------------------------</td>
<td>----------------</td>
</tr>
<tr>
<td>Crosses(anotherGeometry: Geometry): Integer</td>
<td>geof:sfCrosses</td>
<td>1.0</td>
<td>geo:sfCrosses</td>
<td>1.0</td>
</tr>
<tr>
<td>Within(anotherGeometry: Geometry): Integer</td>
<td>geof:sfWithin</td>
<td>1.0</td>
<td>geo:sfWithin</td>
<td>1.0</td>
</tr>
<tr>
<td>Contains(anotherGeometry: Geometry): Integer</td>
<td>geof:sfContains</td>
<td>1.0</td>
<td>geo:sfContains</td>
<td>1.0</td>
</tr>
<tr>
<td>Overlaps(anotherGeometry: Geometry): Integer</td>
<td>geof:sfOverlaps</td>
<td>1.0</td>
<td>geo:sfOverlaps</td>
<td>1.0</td>
</tr>
<tr>
<td>Relate(anotherGeometry: Geometry, IntersectionPatternMatrix: String): Integer</td>
<td>geof:relate</td>
<td>1.0</td>
<td>N/A</td>
<td>-</td>
</tr>
</tbody>
</table>

2.1.1.3 Spatial Analysis

<p>| 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 | - |</p>
<table>
<thead>
<tr>
<th>Simple Features for SQL</th>
<th>GeoSPARQL Equivalent</th>
<th>Since GeoSPARQL</th>
<th>Related Property Available</th>
<th>Since GeoSPARQL</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>2.1.2.1 GeometryCollection</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NumGeometries(): Integer</td>
<td><code>geof:numGeometries</code></td>
<td>-</td>
<td>N/A</td>
<td>-</td>
</tr>
<tr>
<td>GeometryN(N: Integer): Geometry</td>
<td><code>geof:geometryN</code></td>
<td>-</td>
<td>N/A</td>
<td>-</td>
</tr>
<tr>
<td><strong>2.1.3.1 Point</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>X(): Double</td>
<td>N/A</td>
<td>-</td>
<td>N/A</td>
<td>-</td>
</tr>
<tr>
<td>Y(): Double</td>
<td>N/A</td>
<td>-</td>
<td>N/A</td>
<td>-</td>
</tr>
<tr>
<td>Z(): Double (not in the SQL spec, but a logical extension)</td>
<td>N/A</td>
<td>-</td>
<td>N/A</td>
<td>-</td>
</tr>
<tr>
<td>M(): Double (not in the SQL spec, but a logical extension)</td>
<td>N/A</td>
<td>-</td>
<td>N/A</td>
<td>-</td>
</tr>
<tr>
<td><strong>2.1.5.1 Curve</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Length(): Double</td>
<td><code>geof:length</code></td>
<td>-</td>
<td><code>geo:hasLength</code></td>
<td>1.1</td>
</tr>
<tr>
<td>startPoint(): Point</td>
<td>N/A</td>
<td>-</td>
<td>N/A</td>
<td>-</td>
</tr>
<tr>
<td>EndPoint(): Point</td>
<td>N/A</td>
<td>-</td>
<td>N/A</td>
<td>-</td>
</tr>
<tr>
<td>IsClosed(): Integer</td>
<td>N/A</td>
<td>-</td>
<td>N/A</td>
<td>-</td>
</tr>
<tr>
<td>IsRing(): Integer</td>
<td>N/A</td>
<td>-</td>
<td>N/A</td>
<td>-</td>
</tr>
<tr>
<td><strong>2.1.6.1 LineString</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NumPoints(): Integer</td>
<td>N/A</td>
<td>-</td>
<td>N/A</td>
<td>-</td>
</tr>
<tr>
<td>PointN(N: Integer): Point</td>
<td>N/A</td>
<td>-</td>
<td>N/A</td>
<td>-</td>
</tr>
<tr>
<td><strong>2.1.7.1 MultiCurve</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IsClosed(): Integer</td>
<td>N/A</td>
<td>-</td>
<td>N/A</td>
<td>-</td>
</tr>
<tr>
<td>Length(): Double</td>
<td><code>geof:length</code></td>
<td>-</td>
<td><code>geo:hasLength</code></td>
<td>1.1</td>
</tr>
<tr>
<td><strong>2.1.9.1 Surface</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Area(): Double</td>
<td><code>geof:area</code></td>
<td>-</td>
<td><code>geo:hasArea</code></td>
<td>1.1</td>
</tr>
<tr>
<td>Centroid(): Point</td>
<td><code>geof:centroid</code></td>
<td>1.1</td>
<td><code>geo:hasCentroid</code></td>
<td>1.1</td>
</tr>
<tr>
<td>PointOnSurface(): Point</td>
<td>N/A</td>
<td>-</td>
<td>N/A</td>
<td>-</td>
</tr>
<tr>
<td><strong>2.1.10.1 Polygon</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Simple Features for SQL</td>
<td>GeoSPARQL Equivalent</td>
<td>Since GeoSPARQL</td>
<td>Related Property Available</td>
<td>Since GeoSPARQL</td>
</tr>
<tr>
<td>---------------------------------------------</td>
<td>----------------------</td>
<td>----------------</td>
<td>----------------------------</td>
<td>----------------</td>
</tr>
<tr>
<td>ExteriorRing(): LineString</td>
<td>N/A</td>
<td>-</td>
<td>N/A</td>
<td>-</td>
</tr>
<tr>
<td>NumInteriorRing(): Integer</td>
<td>N/A</td>
<td>-</td>
<td>N/A</td>
<td>-</td>
</tr>
<tr>
<td>InteriorRingN(N: Integer): LineString</td>
<td>N/A</td>
<td>-</td>
<td>N/A</td>
<td>-</td>
</tr>
</tbody>
</table>

### 2.11.1 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                        | -              |
### Annex C: Annex G - Revision History

<table>
<thead>
<tr>
<th>Date</th>
<th>Release</th>
<th>Author</th>
<th>Paragraph modified</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>11 Nov. 2009</td>
<td>Draft</td>
<td>John R. Herring</td>
<td>All</td>
<td>Creation</td>
</tr>
<tr>
<td>06 Jan. 2010</td>
<td>Draft</td>
<td>John R. Herring</td>
<td>All</td>
<td>Comment responses</td>
</tr>
<tr>
<td>30 March 2010</td>
<td>Draft</td>
<td>Matthew Perry</td>
<td>All</td>
<td>Comment responses</td>
</tr>
<tr>
<td>26 Oct. 2010</td>
<td>Draft</td>
<td>Matthew Perry</td>
<td>All</td>
<td>Revision based on working group discussion</td>
</tr>
<tr>
<td>28 Jan. 2011</td>
<td>Draft</td>
<td>Matthew Perry</td>
<td>All</td>
<td>Revision based on working group discussion</td>
</tr>
<tr>
<td>18 April 2011</td>
<td>Draft</td>
<td>Matthew Perry</td>
<td>All</td>
<td>Restructure with multiple conformance classes</td>
</tr>
<tr>
<td>02 May 2011</td>
<td>Draft</td>
<td>Matthew Perry</td>
<td>Clause 6 and Clause 8</td>
<td>Move Geometry Class from core to geometryExtension</td>
</tr>
<tr>
<td>05 May 2011</td>
<td>Draft</td>
<td>Matthew Perry</td>
<td>All</td>
<td>Update URIs</td>
</tr>
<tr>
<td>13 Jan. 2012</td>
<td>Draft</td>
<td>Matthew Perry</td>
<td>All</td>
<td>Revision based on Public RFC</td>
</tr>
<tr>
<td>16 April 2012</td>
<td>Draft</td>
<td>Matthew Perry</td>
<td>All</td>
<td>Revision based on adoption vote comments</td>
</tr>
<tr>
<td>19 July 2012</td>
<td>1.0</td>
<td>Matthew Perry</td>
<td>All</td>
<td>Revision of URIs based on OGC Naming Authority recommendations</td>
</tr>
<tr>
<td>09 Oct. 2020</td>
<td>1.1 Draft</td>
<td>Joseph Abhayaratna</td>
<td>All</td>
<td>Establishment of the 1.1 Specification</td>
</tr>
<tr>
<td>Date</td>
<td>Release</td>
<td>Author</td>
<td>Paragraph modified</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>--------------------------</td>
<td>---------------------------------------</td>
<td>--------------------</td>
<td>-------------------------------------------------------</td>
</tr>
<tr>
<td>10 Oct. 2020 to 02 June. 2022</td>
<td>1.1 Draft</td>
<td>GeoSPARQL 1.1 SWG</td>
<td>All</td>
<td>Addition of GeoSPARQL 1.1 elements</td>
</tr>
<tr>
<td>23 Oct. 2022</td>
<td>1.1 For Public Comment</td>
<td>Karl Reed, Joseph Abhayaratna</td>
<td>All</td>
<td>Final review prior to public comment</td>
</tr>
</tbody>
</table>
Bibliography

[DE-9IM]

[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]

[CATEG]

[AUSPIX]

[ISO13249]

[ISO19105]

[ISO19107]

[ISO19109]
[ISO19125-1]

[ISO19156]

[IETF3987]

[IETF5234]

[GEOJSON]

[GML]

[OGCKML]

[OGCOM]

[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]

[OGCAPIF]
[DGGAS]

[CQLDEF]

[WFS]

[LOGIC]

[XSD1]

[XSD2]

[XML]

[SKOS]

[XMLNS]

[OWL2]

[TIME]

[RIF]
February 2013). [RIFCORE](https://www.w3.org/TR/rif-overview/)

[RIFCORE]

[Sparql]

[SPARQLENT]

[SPARQLPROT]

[SPARQLRESX]

[SPARQLRESJ]

[TURTLE]

[RDFXML]

[RDF]

[RDFSEM]

[RDFS]

[SHACL]
[PROF]

[JSON-LD]

[SPARQLSERVDESC]