International Standard

ISO/OGC 19000-1

GeoSPARQL — GeoSPARQL - A Geographic Query Language for RDF Data

GeoSPARQL — GeoSPARQL - Un langage de requête géographique pour les données RDF

Reference number
ISO/OGC 19000-1:2025(en)

Contents

Page

Abstract

GeoSPARQL contains a small spatial domain ontology that allow literal representations of geometries to be associated with spatial features and for features to be associated with other features using spatial relations.

GeoSPARQL also contains SPARQL extension function definitions that can be used to calculate relations between spatial objects.

Several other supporting assets are also contained within GeoSPARQL such as vocabularies of Simple Feature types and data validators.

ISO (the International Organization for Standardization) is a worldwide federation of national standards bodies (ISO member bodies). The work of preparing International Standards is normally carried out through ISO technical committees. Each member body interested in a subject for which a technical committee has been established has the right to be represented on that committee. International organizations, governmental and non-governmental, in liaison with ISO, also take part in the work. ISO collaborates closely with the Open Geospatial Consortium on matters of spatial data standardization.

Any trade name used in this document is information given for the convenience of users and does not constitute an endorsement.

For an explanation of the voluntary nature of standards, the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the World Trade Organization (WTO) principles in the Technical Barriers to Trade (TBT), see www.iso.org/iso/foreword.html.

Introduction

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

0.1 RDF

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

— Subject (start node, instance, entity, feature)

— Predicate (verb, property, attribute, relation, member, link, reference)

— Object (value, end node, non-literal values can be used as a Subject)

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

Subjects and objects within an RDF triple are called nodes and can also be represented with a blank node (a local identifier with meaning outside the graph it is defined within). Objects can further be represented with a literal value. Basic literal values in RDF are those used in XML XSD2 but the basic types can be extended for specialised purposes and in this specification are, for geometry data. The figure below, Figure 1 shows a basic triple.

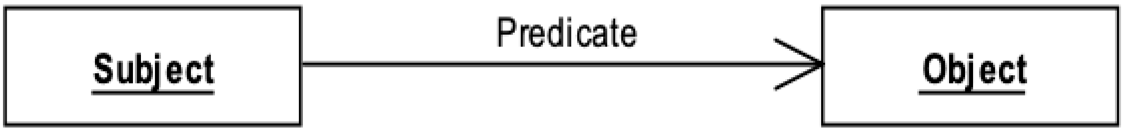

Figure 1 — An RDF Triple

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

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

0.2 SPARQL

From SPARQL:

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

and, from Wikipedia[1]:

> SPARQL (pronounced "sparkle" /ˈspɑːkəl/, a recursive acronym for SPARQL Protocol and RDF Query Language) is an RDF query language — that is, a semantic query language for databases

1) https://en.wikipedia.org/wiki/SPARQL

— able to retrieve and manipulate data stored in Resource Description Framework (RDF) format. It was made a standard by the RDF Data Access Working Group (DAWG) of the World Wide Web Consortium, and is recognized as one of the key technologies of the semantic web. On 15 January 2008, SPARQL 1.0 was acknowledged by W3C as an official recommendation, and SPARQL 1.1 in March 2013.

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

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

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

0.3 GeoSPARQL Standard structure

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

a) *profile definition*

   — http://www.opengis.net/def/geosparql

   — formally defined as an ontology, defined according to the *Profiles Vocabulary* PROF

   — this relates the parts in the standard together, provides access to them, and declares dependencies on other standards

b) **specification document**

   — *this document*

   — **defines many of the standard’s parts**

   — **includes normative RDF/OWL RDF,OWL2 ontology element definitions, conformance requirements and function signatures**

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

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

c) domain model RDF/OWL RDF,OWL2 ontology

   — http://www.opengis.net/ont/geosparql

   — for geographic information representation

   — based on the General Feature Model ISO19109, Simple Features Access ISO19125-1, Feature Geometry ISO19107 and SQL MM ISO13249

- defined within the specification document and delivered in RDF also

d) Functions and Rules vocabulary

- http://www.opengis.net/def/geosparql/funcsrules
- derived from the ontology
- presented as a SKOS taxonomy

e) Simple Features vocabulary

- http://www.opengis.net/ont/sf
- derived from Simple Features Access ISO19125-1's class model
- presented as an OWL2 ontology

f) SPARQL SPARQL extension functions

- defined within this specification document

g) RIF RIFCORE rules

- http://www.opengis.net/def/geosparql/rifrules
- templated within the specification document
- also delivered as a RIF document also

h) RDF data validator

- http://www.opengis.net/def/geosparql/validator
- defined using SHACL SHACL
- presented within a single RDF file

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

- a *core* component defining the top-level RDFS/OWL classes for spatial objects
- a *topology vocabulary* component defining the RDF properties for asserting and querying topological relations between spatial objects
- a *geometry* component defines RDFS data types for serializing geometry data, geometry-related RDF properties, and non-topological spatial query functions for geometry objects
- a *geometry topology* component defining topological query functions
- an *RDFS entailment* component defining mechanisms for matching implicit RDF triples that are derived based on RDF and RDFS semantics
- a *query rewrite* component defining rules for transforming a simple triple pattern that tests a topological relation between two features into an equivalent query involving concrete geometries and topological query functions

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

In addition, GeoSPARQL is designed to accommodate systems based on qualitative spatial reasoning and systems based on quantitative spatial computations. Systems based on qualitative spatial reasoning, (e.g. those based on the Region Connection Calculus QUAL, [36]) do not usually model explicit geometries, so queries in such systems will likely test for binary spatial relationships between features rather than between explicit geometries. To allow queries for spatial relations between features in quantitative systems, GeoSPARQL defines a series of query transformation rules that expand a feature-only query into a geometry-based query. With these transformation rules, queries about spatial relations between features will have the same specification in both qualitative systems and quantitative systems. The qualitative system will likely evaluate the query with a backward-chaining spatial “reasoner”, and the quantitative system can transform the query into a geometry-based query that can be evaluated with computational geometry.

GeoSPARQL — GeoSPARQL - A Geographic Query Language for RDF Data

1 Scope

The GeoSPARQL Standard comprises multiple parts. See the Introduction section [GeoSPARQL_Standard_structure] for details of the parts.

GeoSPARQL does not define a comprehensive vocabulary for representing spatial information. Instead, it 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.

2 Normative references

The following documents are referred to in the text in such a way that some or all of their content constitutes requirements of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

ISO19125-1, INTERNATIONAL ORGANIZATION FOR STANDARDIZATION. *Geographic information — Simple feature access*. Geneva, CH: 2005a. vol. 2004. Available from: https://www.iso.org/standard/40114.html

ISO19156, INTERNATIONAL ORGANIZATION FOR STANDARDIZATION. *Geographic information — Observations and measurements*. Geneva, CH: 2011. vol. 2000. Available from: https://www.iso.org/standard/32574.html

GML, OPEN GEOSPATIAL CONSORTIUM. *OpenGIS Geography Markup Language (GML) Encoding Standard*. 2007a. Available from: https://portal.ogc.org/files/?artifact_id=20509/

IETF3987, DÜRST, M. J. and M. SUIGNARD. *Internationalized Resource Identifiers (IRIs)*. RFC Editor. Available from: https://www.rfc-editor.org/info/rfc3987

OWL2, GROUP, W. O. W. *OWL 2 Web Ontology Language Document Overview (Second Edition)*. 2012. Available from: https://www.w3.org/TR/2012/REC-owl2-overview-20121211/

RDF, LANTHALER, M., R. CYGANIAK and D. WOOD. *RDF 1.1 Concepts and Abstract Syntax*. 2014. Available from: https://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/

RDFS, GUHA, R. and D. BRICKLEY. *RDF Schema 1.1*. 2014. Available from: https://www.w3.org/TR/2014/REC-rdf-schema-20140225/

RIFCORE, BOLEY, H., G. HALLMARK, D. REYNOLDS, A. PASCHKE, A. POLLERES and M. KIFER. *RIF Core Dialect (Second Edition)*. 2013a. Available from: https://www.w3.org/TR/2013/REC-rif-core-20130205/

SPARQL, BOLEY, H., G. HALLMARK, D. REYNOLDS, A. PASCHKE, A. POLLERES and M. KIFER. *RIF Core Dialect (Second Edition)*. 2013b. Available from: https://www.w3.org/TR/2013/REC-rif-core-20130205/

SPARQLENT, OGBUJI, C. and B. GLIMM. *SPARQL 1.1 Entailment Regimes*. 2013. Available from: https://www.w3.org/TR/2013/REC-sparql11-entailment-20130321/

SPARQLPROT, CLARK, K., G. WILLIAMS, E. TORRES and L. FEIGENBAUM. *SPARQL 1.1 Protocol*. 2013. Available from: https://www.w3.org/TR/2013/REC-sparql11-protocol-20130321/

SPARQLRESX, BECKETT, D. and J. BROEKSTRA. *SPARQL Query Results XML Format (Second Edition)*. 2013. Available from: https://www.w3.org/TR/2013/REC-rdf-sparql-XMLres-20130321/

SPARQLRESJ, SEABORNE, A. *SPARQL 1.1 Query Results JSON Format*. 2013. Available from: https://www.w3.org/TR/2013/REC-sparql11-results-json-20130321/

3 Conformance

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

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

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

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

Table 1 — Conformance Classes

Conformance class	Description	Subclause of the abstract test suite
Core	Defines top-level spatial vocabulary components	[_conf_core]
Topology Vocabulary Extension (relation_family)	Defines topological relation vocabulary	[_conf_topology-vocab-extension]
Geometry Extension (serialization, version)	Defines geometry vocabulary and non-topological query functions	[_conf_geometry-extension]
Geometry Extension — DGGS	Defines the properties and functions of the Geometry Extension Conformance Classes for use with Discrete Global Grid System geometry representations	[_conf_geometry-extension-dggs]
Geometry Topology Extension (serialization, version, relation_family)	Defines topological query functions for geometry objects	[_conf_geometry-topology-extension]
RDFS Entailment Extension (serialization, version , relation_family)	Defines a mechanism for matching implicit RDF triples that are derived based on RDF and RDFS semantics	[_conf_rdfs-entailment-extension]
Query Rewrite Extension (serialization, version, relation_family)	Defines query transformation rules for computing spatial relations between spatial objects based on their associated geometries	[_conf_query-rewrite-extension]

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

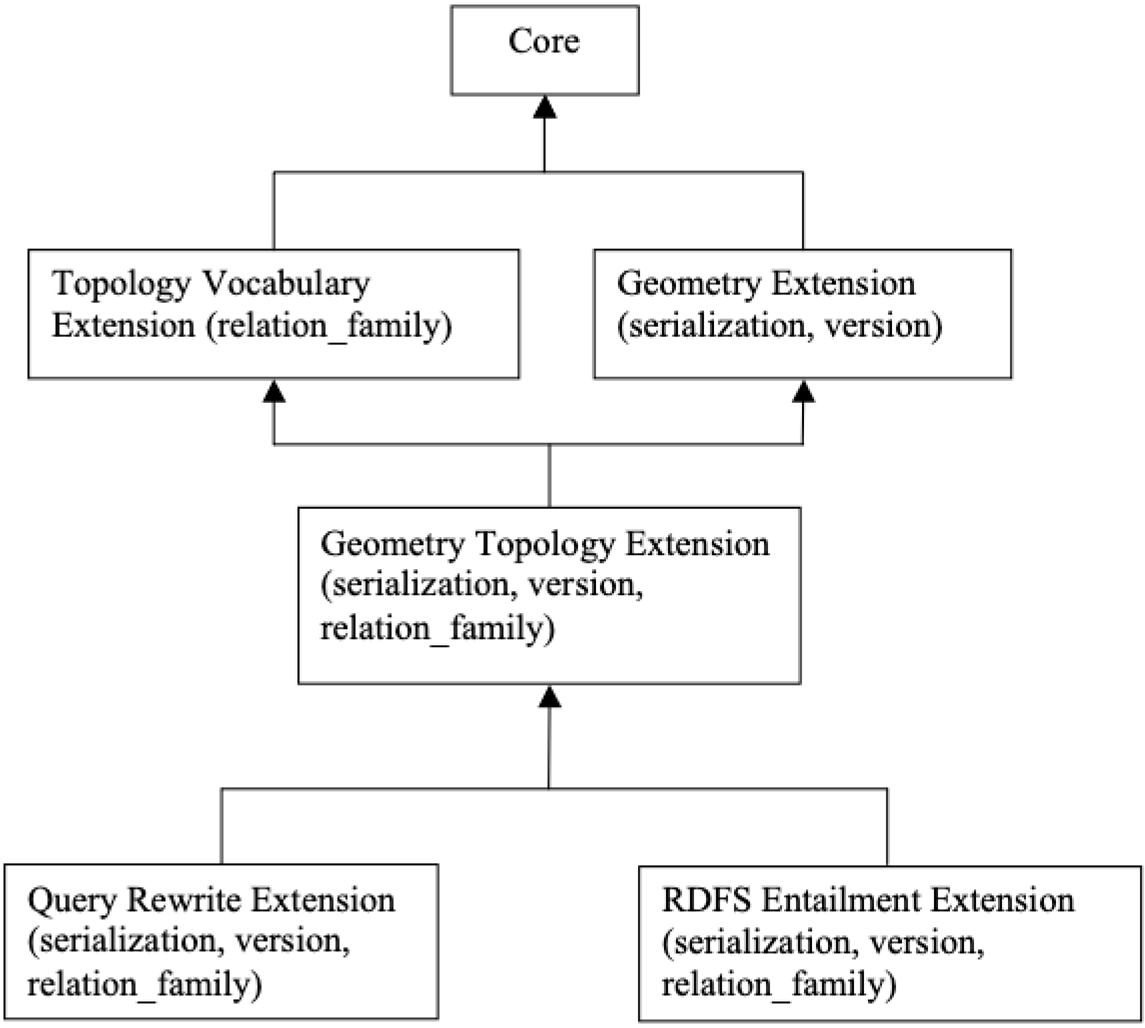

Figure 2 — Requirements Class Dependency Graph

4 Terms and definitions

For the purposes of this document, the following terms and definitions apply.

ISO and IEC maintain terminology databases for use in standardization at the following addresses:

— ISO Online browsing platform: available at https://www.iso.org/obp

— IEC Electropedia: available at https://www.electropedia.org

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).

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 (http://www.w3.org/)
XML	Extensible Markup Language

5.2 Namespaces

The following IRI namespace prefixes are used throughout this document:

Prefix	Namespace	Description
`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
`my:`	`http://example.org/ApplicationSchema#`	Example application schema
`ogc:`	`http://www.opengis.net/`	OGC placeholder
`owl:`	`http://www.w3.org/2002/07/owl#`	OWL2 ontology
`rdf:`	`http://www.w3.org/1999/02/22-rdf-syntax-ns#`	RDF ontology
`rdfs:`	`http://www.w3.org/2000/01/rdf-schema#`	RDFS ontology
`sf:`	`http://www.opengis.net/ont/sf#`	Simple Features vocabulary
`skos:`	`http://www.w3.org/2004/02/skos/core#`	SKOS ontology
`xsd:`	`http://www.w3.org/2001/XMLSchema#`	XSD2 datatypes vocabulary

5.3 Placeholder IRIs

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

5.4 RDF Serializations

Three RDF serializations are used in this document. Terse RDF Triple Language (turtle) TURTLE is used for RDF snippets placed within the main body of the document. Turtle, JSON-LD JSON-LD and RDF/XML RDFXML are used for the examples in Annex C.

6 Core

This clause establishes the **Core** requirements class, with IRI `/req/core`, which has a single corresponding conformance class, **Core**, with IRI `/conf/core`. This requirements class defines a set of classes and properties for representing geospatial data. The resulting vocabulary — an ontology — can be used to construct SPARQL graph patterns for querying appropriately modeled geospatial data. 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.

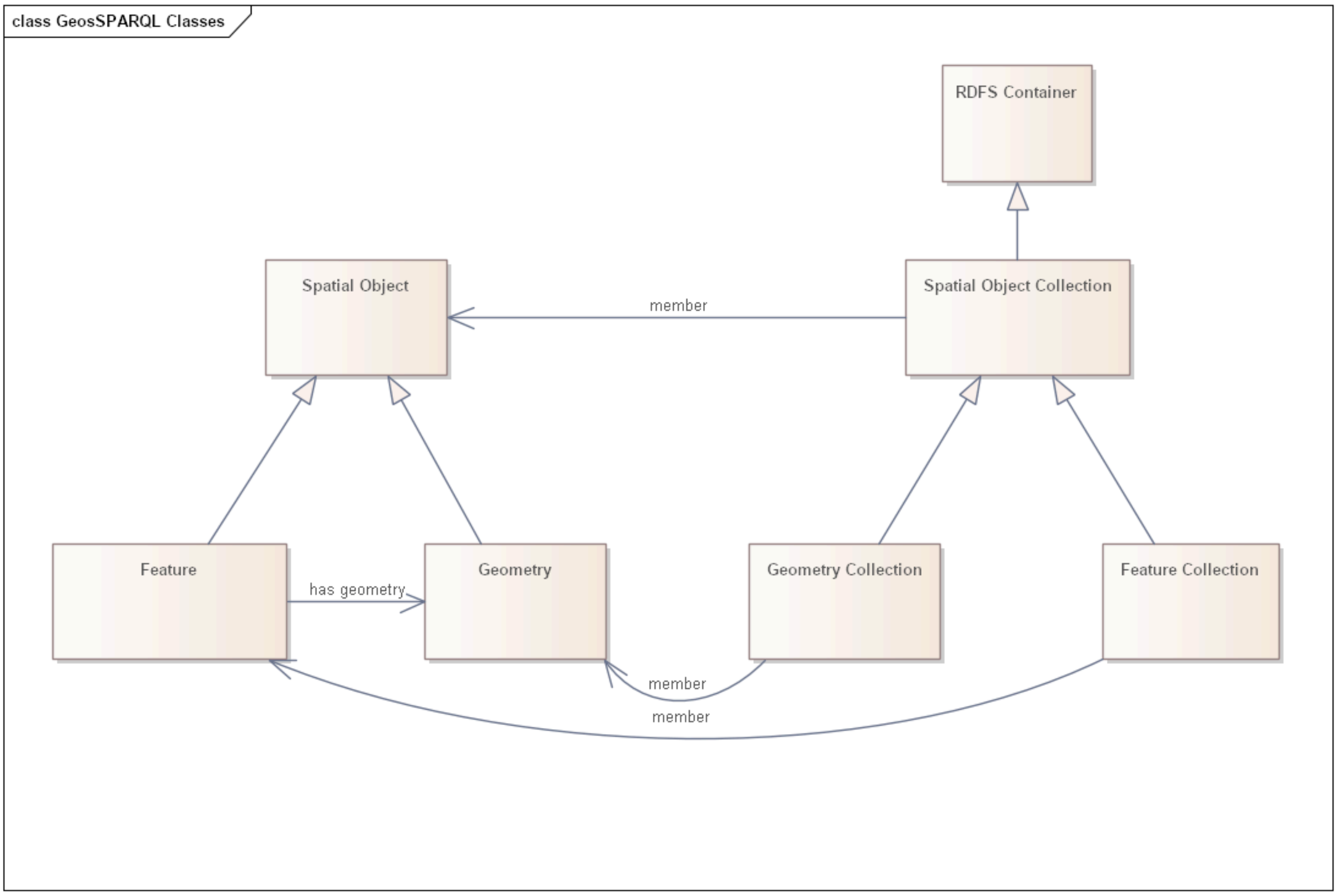

Figure 3 — An overview of GeoSPARQL's Classes and their main relations.

Requirement 1:
/req/core. Core Extension
Subject: Implementation Specification

requirement	/req/core/sparql-protocol
requirement	/req/core/spatial-object-class
requirement	/req/core/feature-class
requirement	/req/core/spatial-object-collection-class
requirement	/req/core/feature-collection-class

requirement	/req/core/geometry-collection-class
requirement	/req/core/spatial-object-properties

6.1 SPARQL

Requirement 2:
/req/core/sparql-protocol. SPARQL Protocol
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.

6.2 Classes

Two main classes are defined: `Spatial Object` and `Feature`. The class `Feature` is equivalent to the UML class `Feature` defined in ISO19109.

Two container classes are defined: `SpatialObjectCollection` and `FeatureCollection`.

6.2.1 SpatialObject

The class `SpatialObject` is defined by the following:

```
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 ;
.
```

Requirement 3:
/req/core/spatial-object-class. Spatial Object Class
Implementations shall allow the RDFS class `SpatialObject` to be used in SPARQL graph patterns.

Example:

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

6.2.2 Feature

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

```
geo:Feature
    a rdfs:Class, owl:Class ;
    rdfs:isDefinedBy geo: ;
    skos:prefLabel "Feature"@en ;
    rdfs:subClassOf geo:SpatialObject ;
    owl:disjointWith geo:Geometry ;
    skos:definition "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 ;
.
```

Requirement 4:
/req/core/feature-class. Feature Class
Implementations shall allow the RDFS class `Feature` to be used in SPARQL graph patterns.

6.2.3 SpatialObjectCollection

The class `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 `SpatialObject`. `SpatialObjectCollection` members are to be indicated with the `rdfs:member` property.

Requirement 5:
/req/core/spatial-object-collection-class. Spatial Object Collection Class
Implementations shall allow the RDFS class `SpatialObjectCollection` to be used in SPARQL graph patterns.

6.2.4 FeatureCollection

The class `FeatureCollection` is defined by the following:

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

Membership of the more general `SpatialObjectCollection` that defines this class is restricted to instances of `Feature`. `FeatureCollection` members are to be indicated with the `rdfs:member` property.

Requirement 6:
/req/core/feature-collection-class. Feature Collection Class
Implementations shall allow the RDFS class `FeatureCollection` to be used in SPARQL graph patterns.

6.3 Standard Properties for `SpatialObject`

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

Requirement 7:
/req/core/spatial-object-properties. Spatial Object Properties
Implementations shall allow the properties `hasSize`, `hasMetricSize`, `hasLength`, `hasMetricLength`, `hasPerimeterLength`, `hasMetricPerimeterLength`, `hasArea`, `hasMetricArea`, `hasVolume` and `hasMetricVolume` to be used in SPARQL graph patterns.

6.3.1 hasSize

The property 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 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: hasLength, hasPerimeterLength, hasArea and 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 hasMetricSize

The property 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 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: hasMetricLength, hasMetricPerimeterLength, hasMetricArea and hasMetricVolume.

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

6.3.3 hasLength

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

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

6.3.4 hasMetricLength

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

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

6.3.5 hasPerimeterLength

The property 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 hasMetricPerimeterLength. It is a subproperty of hasSize.

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

6.3.6 hasMetricPerimeterLength

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

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

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

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

        FILTER (?d < 2)
    }
```

6.3.7 hasArea

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

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

6.3.8 hasMetricArea

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

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

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

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

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

    FILTER (?d < 2)
}
```

6.3.9 hasVolume

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

The property hasMetricVolume can be used to indicate the volume of a Spatial Object in cubic meters (m^3). It is a subproperty of 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 ;
.
```

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

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

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

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

6.4 Standard Properties for `Feature`

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

Requirement 8:
/req/geometry-extension/feature-properties. Feature Properties
Implementations shall allow the properties `hasGeometry`, `hasDefaultGeometry`, `hasCentroid` and `hasBoundingBox` to be used in SPARQL graph patterns.

6.4.1 hasGeometry

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

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

6.4.2 hasDefaultGeometry

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

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

.

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

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

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

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

6.4.3 hasBoundingBox

The property `hasBoundingBox` is used to link a Feature with a simplified geometry-representation corresponding to the envelope of its 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
                    aligned with the axes of the coordinate reference system, which exactly
                    contains the geometry or Feature e.g. sf:Envelope"@en ;
.
```

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

6.4.4 hasCentroid

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

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

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

7 Topology Vocabulary Extension

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.

Requirement 9:
/req/topology-vocab-extension. Topology Vocabulary Extension
Subject: Implementation Specification

requirement	/req/topology-vocab-extension/sf-spatial-relations
requirement	/req/topology-vocab-extension/eh-spatial-relations
requirement	/req/topology-vocab-extension/rcc8-spatial-relations

A Dimensionally Extended 9-Intersection Model [35] 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.

Requirement 10:
/req/topology-vocab-extension/sf-spatial-relations. Simple Feature Spatial Relations
Implementations shall allow the properties `sfEquals`, `sfDisjoint`, `sfIntersects`, `sfTouches`, `sfCrosses`, `sfWithin`, `sfContains` and `sfOverlaps` to be used in SPARQL graph patterns.

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

Table 2 — Simple Features Topological Relations

Relation Name	Relation IRI	Domain/Range	Applies To Geometry Types	DE-9IM Intersection Pattern
equals	`sfEquals`	`SpatialObject`	`All`	`(TFFFTFFFT)`
disjoint	`sfDisjoint`	`SpatialObject`	`All`	`(FF*FF****)`
intersects	`sfIntersects`	`SpatialObject`	`All`	`(T******** *T******* ***T***** ****T****)`
touches	`sfTouches`	`SpatialObject`	`All except P/P`	`(FT******* F**T***** F***T****)`

Table 2 *(continued)*

Relation Name	Relation IRI	Domain/Range	Applies To Geometry Types	DE-9IM Intersection Pattern
within	sfWithin	SpatialObject	All	(T*F**F***)
contains	sfContains	SpatialObject	All	(T*****FF*)
overlaps	sfOverlaps	SpatialObject	A/A, P/P, L/L	(T*T***T**) for A/A, P/P; (1*T***T**) for L/L
crosses	sfCrosses	SpatialObject	P/L, P/A, L/A, L/L	(T*T***T**) for P/L, P/A, L/A; (0********) for L/L

7.3 Egenhofer Relation Family

This clause defines Requirements for the 9-intersection model for the binary topological relations (*Egenhofer*) relation family. The reader should consult references [34] and CATEG for a more detailed discussion of *Egenhofer* relations.

Requirement 11:
/req/topology-vocab-extension/eh-spatial-relations. Egenhofer Spatial Relations
Implementations shall allow the properties `ehEquals`, `ehDisjoint`, `ehMeet`, `ehOverlap`, `ehCovers`, `ehCoveredBy`, `ehInside` and `ehContains` to be used in SPARQL graph patterns.

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

Table 3 — Egenhofer Topological Relations

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

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 [36] for a more detailed discussion of *RCC8* relations.

Requirement 12:
/req/topology-vocab-extension/rcc8-spatial-relations. RCC8 Spatial Relations
Implementations shall allow the properties `rcc8eq`, `rcc8dc`, `rcc8ec`, `rcc8po`, `rcc8tppi`, `rcc8tpp`, `rcc8ntpp`, `rcc8ntppi` to be used in SPARQL graph patterns.

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

Table 4 — RCC8 Topological Relations

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

7.5 Equivalent RCC8, Egenhofer and Simple Features Topological Relations

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

Table 5 — Equivalent Simple Features, RCC8 and Egenhofer relations

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

8 Geometry Extension

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.

Requirement 13:
/req/geometry-extension. Geometry Extension
Subject: Implementation Specification

requirement	/req/geometry-extension/geometry-class
requirement	/req/geometry-extension/geometry-collection-class
requirement	/req/geometry-extension/feature-properties
requirement	/req/geometry-extension/geometry-properties
requirement	/req/geometry-extension/query-functions
requirement	/req/geometry-extension/srid-function
requirement	/req/geometry-extension/sa-functions

requirement	/req/geometry-extension/wkt-literal
requirement	/req/geometry-extension/wkt-literal-default-srs
requirement	/req/geometry-extension/wkt-axis-order
requirement	/req/geometry-extension/wkt-literal-empty
requirement	/req/geometry-extension/geometry-as-wkt-literal
requirement	/req/geometry-extension/asWKT-function
requirement	/req/geometry-extension/gml-literal
requirement	/req/geometry-extension/gml-literal-empty
requirement	/req/geometry-extension/gml-profile
requirement	/req/geometry-extension/geometry-as-gml-literal
requirement	/req/geometry-extension/asGML-function
requirement	/req/geometry-extension/geojson-literal
requirement	/req/geometry-extension/geojson-literal-srs
requirement	/req/geometry-extension/geojson-literal-empty
requirement	/req/geometry-extension/geometry-as-geojson-literal
requirement	/req/geometry-extension/asGeoJSON-function
requirement	/req/geometry-extension/kml-literal
requirement	/req/geometry-extension/kml-literal-srs
requirement	/req/geometry-extension/kml-literal-empty
requirement	/req/geometry-extension/geometry-as-kml-literal
requirement	/req/geometry-extension/asKML-function

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[2)] 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[3)] for "A spatial region or named place." and *schema.org* provides a number of spatial object and geometry classes, such as `GeoCoordinates`[4)] and `GeoShape`[5)].

2) http://www.w3.org/2003/01/geo/

3) http://purl.org/dc/terms/Location

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*[6] 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[7] 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 `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 `hasSpatialResolution` or the function `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

4) https://schema.org/GeoCoordinates

5) https://schema.org/GeoShape

6) https://www.w3.org/ns/locn

7) https://www.w3.org/TR/vocab-dcat/#spatial-properties

data interoperability. The recommended vocabulary for units of measurement for GeoSPARQL is the *Quantities, Units, Dimensions and Types (QUDT)* ontology[8] 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.

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: `Geometry`. In addition, properties are defined for describing geometry data and for associating geometries with features.

One container class is defined: `GeometryCollection`.

8.6.1 Geometry

The class `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". `Geometry` is defined by the following:

```
geo:Geometry
    a rdfs:Class, owl:Class ;
    rdfs:isDefinedBy geo: ;
    skos:prefLabel "Geometry"@en ;
    rdfs:subClassOf geo:SpatialObject ;
    owl:disjointWith geo:Feature;
    skos:definition "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 ;
```

8) http://www.qudt.org

.

Requirement 14:
/req/geometry-extension/geometry-class. Geometry Class
Implementations shall allow the RDFS class `Geometry` to be used in SPARQL graph patterns.

8.6.2 Geometry Collection

The class `GeometryCollection` is defined by the following:

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

Membership of the general `SpatialObjectCollection` that defines this class is restricted to instances of `Geometry`. `GeometryCollection` members are to be indicated with the `rdfs:member` property.

NOTE There is no RDF/ontology relationship between this `GeometryCollection` class and the Simple Features Vocabulary's `sf:GeometryCollection` class since the former is a collection of `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 `GeometryCollection` instances are more likely to be used for grouping `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 `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 `GeometryCollection` instances is expected to be different too.

Requirement 15:
/req/core/geometry-collection-class. Geometry Collection Class
Implementations shall allow the RDFS class `GeometryCollection` to be used in SPARQL graph patterns.

8.7 Standard Properties for `Geometry`

Properties are defined for describing geometry metadata.

Requirement 16:
/req/geometry-extension/geometry-properties. Geometry Properties
Implementations shall allow the properties `dimension`, `coordinateDimension`, `hasSpatialDimension`, `hasSpatialResolution`, `hasMetricSpatialResolution`, `hasSpatialAccuracy`, `hasMetricSpatialAccuracy`, `isEmpty`, `isSimple` and `hasSerialization` to be used in SPARQL graph patterns.

8.7.1 dimension

The property `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 coordinateDimension

The property 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 hasSpatialDimension

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

8.7.4 hasSpatialResolution

The property 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 Geometry, all literal representations of that Geometry instance must have the same spatial resolution.

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

NOTE See the 8.3.

8.7.5 hasMetricSpatialResolution

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

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

8.7.6 hasSpatialAccuracy

The property `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).

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

NOTE See the 8.3.

8.7.7 hasMetricSpatialAccuracy

The property `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).

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

8.7.8 isEmpty

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

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

8.7.9 isSimple

The property `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.

```
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 hasSerialization

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

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

NOTE this property is the generic property used to connect a Geometry with its serialization. GeoSPARQL also contains a number of sub properties of this 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 `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.

EXAMPLE 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 wktLiteral

The datatype `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 ;
.
```

Requirement 17:
/req/geometry-extension/wkt-literal. WKT Literal
All RDFS Literals of type `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 `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 (< and >) followed by whitespace as a separator, and a WKT string as defined in ISO13249.

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.

Requirement 18:
/req/geometry-extension/wkt-literal-default-srs. 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 `wktLiteral` instances that do not specify an explicit spatial reference system IRI.

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.

Requirement 19:
/req/geometry-extension/wkt-axis-order. WKT Literal Axis Order
Coordinate tuples within `wktLiteral` shall be interpreted using the axis order defined in the spatial reference system used.

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>
```

Requirement 20:
/req/geometry-extension/wkt-literal-empty. Empty WKT Literal
An empty RDFS Literal of type `wktLiteral` shall be interpreted as an empty Geometry.

8.8.1.2 asWKT

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

Requirement 21:
/req/geometry-extension/geometry-as-wkt-literal. asWKT Property
Implementations shall allow the RDF property `asWKT` to be used in SPARQL graph patterns.

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

8.8.1.3 asWKT

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

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

Requirement 22:
/req/geometry-extension/asWKT-function. asWKT Function
Implementations shall support `asWKT` as a SPARQL extension function.

8.8.2 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 OGC07-036. It defines one RDFS Datatype: GML Literal and one property, as GML.

8.8.2.1 gmlLiteral

The datatype 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 gmlLiteral 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`.

Requirement 23:
/req/geometry-extension/gml-literal. GML Literal
All `gmlLiteral` instances shall consist of a valid element from the GML schema that implements a subtype of `GM_Object` as defined in OGC07-036.

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

```
"""
<gml:Point
        srsName=\"http://www.opengis.net/def/crs/OGC/1.3/CRS84\"
        xmlns:gml=\"http://www.opengis.net/gml/3.2\">
```

```
    <gml:pos>-83.38 33.95</gml:pos>
</gml:Point>
"""^^<http://www.opengis.net/ont/geosparql#gmlLiteral>
```

Requirement 24:
/req/geometry-extension/gml-literal-empty. Empty GML Literal
An empty `gmlLiteral` shall be interpreted as an empty Geometry.

Requirement 25:
/req/geometry-extension/gml-profile. GML Profile
Implementations shall document supported GML profiles.

8.8.2.2 asGML

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

Requirement 26:
/req/geometry-extension/geometry-as-gml-literal. asGML Property
Implementations shall allow the RDF property `asGML` to be used in SPARQL graph patterns.

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

8.8.2.3 asGML

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

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

Requirement 27:
/req/geometry-extension/asGML-function. asGML Function
Implementations shall support `asGML` as a SPARQL extension function.

8.8.3 GeoJSON

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

8.8.3.1 geoJSONLiteral

The datatype geoJsonLiteral is used to contain the GeoJSON serialization of a Geometry.

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

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

Requirement 28:
/req/geometry-extension/geojson-literal. GeoJSON Literal

All geoJSONLiteral instances shall consist of the Geometry objects as defined in the GeoJSON specification GEOJSON.

Requirement 29:
/req/geometry-extension/geojson-literal-srs. GeoJSON Literal SRS
RDFS Literals of type 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).

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

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

Requirement 30:
/req/geometry-extension/geojson-literal-empty. Empty GeoJSON Literal
An empty RDFS Literal of type geoJSONLiteral shall be interpreted as an empty Geometry, i.e. `{"geometry": null}` in GeoJSON .

8.8.3.2 asGeoJSON

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

Requirement 31:
/req/geometry-extension/geometry-as-geojson-literal. asGeoJSON Property
Implementations shall allow the RDF property asGeoJSON to be used in SPARQL graph patterns.

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

8.8.3.3 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.

Requirement 32:
/req/geometry-extension/asGeoJSON-function. asGeoJSON Function
Implementations shall support asGeoJSON as a SPARQL extension function.

8.8.4 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, asKML.

8.8.4.1 kmlLiteral

The datatype 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 kmlLiteral instances are formed by encoding Geometry information as a Geometry object as defined in the KML specification [OGCKML].

Requirement 33:
/req/geometry-extension/kml-literal. KML Literal
All `kmlLiteral` instances shall consist of the Geometry objects as defined in the KML specification [OGCKML].

Requirement 34:
/req/geometry-extension/kml-literal-srs. KML Literal SRS
RDFS Literals of type `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).

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

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

Requirement 35:
/req/geometry-extension/kml-literal-empty. Empty KML Literal
An empty RDFS Literal of type `kmlLiteral` shall be interpreted as an empty Geometry .

8.8.4.2 asKML

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

Requirement 36:
/req/geometry-extension/geometry-as-kml-literal. asKML Property
Implementations shall allow the RDF property `asKML` to be used in SPARQL graph patterns.

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

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

8.8.4.3 asKML

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

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

Requirement 37:
/req/geometry-extension/asKML-function. asKML Function
Implementations shall support `asKML` as a SPARQL extension function.

8.8.5 Discrete Global Grid System

Requirement 38:
/req/geometry-extension-dggs. Geometry DGGS Extension

Subject: Implementation Specification

requirement	/req/geometry-extension-dggs/query-functions
requirement	/req/geometry-extension-dggs/query-functions-non-sf
requirement	/req/geometry-extension-dggs/srid-function
requirement	/req/geometry-extension-dggs/sa-functions
requirement	/req/geometry-extension-dggs/dggs-literal
requirement	/req/geometry-extension-dggs/dggs-literal-empty
requirement	/req/geometry-extension-dggs/geometry-as-dggs-literal
requirement	/req/geometry-extension-dggs/asDGGS-function

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.

8.8.5.1 dggsLiteral

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

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

Valid dggsLiteral 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 dggsLiteral datatype.

Requirement 39:
/req/geometry-extension-dggs/dggs-literal. DGGS Literal
All RDFS Literals of type `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 (< and >) followed by whitespace as a separator, and then the DGGS geometry data, formulated according to the identified DGGS.

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

```
dggsLiteral ::= iri-and-whitespace dggs-geomety-data

iri-and-whitespace = "<" IRI ">" LWSP
```

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.

Requirement 40:
/req/geometry-extension-dggs/dggs-literal-empty. Empty DGGS Literal
An empty RDFS Literal of type `dggsLiteral`, shall be interpreted as an empty `Geometry`.

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

```
dggsLiteral ::= iri-and-space dggs-geometry-data

iri-and-whitespace = "<" IRI ">" LWSP / ""
```

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

8.8.5.2 asDGGS

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

Requirement 41:
/req/geometry-extension-dggs/geometry-as-dggs-literal. asDGGS Property
Implementations shall allow the RDF property `asDGGS` to be used in SPARQL graph patterns.

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

8.8.5.3 asDGGS

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

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

Requirement 42:
/req/geometry-extension-dggs/asDGGS-function. asDGGS Function
Implementations shall support `asDGGS` as a SPARQL extension function.

8.9 Non-topological Query Functions

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

Requirement 43:
/req/geometry-extension/query-functions. Non-topological Query Functions (Simple Features)
Implementations shall support the functions `boundary` `boundingCircle`, `metricBuffer`, `buffer`, `centroid`, `convexHull`, `concaveHull`, `coordinateDimension`, `difference`, `dimension`, `metricDistance`, `distance`, `envelope`, `geometryType`, `intersection`, `is3D`, `isEmpty`, `isMeasured`, `isSimple`, `hasSpatialDimension`, `symDifference`, `transform` and `union` as SPARQL extension functions, consistent with definitions of these functions in Simple Features OGCSFACA ISO19125-1, for non-DGGS geometry literals.

Requirement 44:
/req/geometry-extension/query-functions-non-sf. Non-topological Query Functions (Non Simple Features)

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

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.

Requirement 45:
/req/geometry-extension-dggs/query-functions. DGGS Query Functions (Simple Features)
Implementations shall support the functions of Requirement 43 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.

Requirement 46:
/req/geometry-extension-dggs/query-functions-non-sf. DGGS Query Functions (Non Simple Features)
Implementations shall support the functions of Requirement 44 for DGGS geometry literals as SPARQL extension functions which are defined in this standard, for non-DGGS geometry literals.

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 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; or

— 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[9] of the SPARQL specification SPARQL.

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

9) <https://www.w3.org/TR/sparql11-query/#expressions>

From Section 17.3.1[10] of the SPARQL specification SPARQL:

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

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

NOTE Several non-topological query functions use a unit of measure IRI. See the 8.3. Also, the OGC has recommended units of measure vocabularies for use, see the OGC Definitions Server[11].

8.9.2 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 `area` but does not need a specification of measurement unit.

8.9.3 area

```
geof:area (geom: ogc:geomLiteral, units: rdfs:Resource): xsd:double
```

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

NOTE See the 8.3.

8.9.4 boundary

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

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

8.9.5 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 metricBuffer

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

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 `buffer`, but does not need a specification of measurement unit.

10) <https://www.w3.org/TR/sparql11-query/#operatorExtensibility>

11) https://www.ogc.org/def-server

8.9.7 buffer

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

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 metricBuffer, which does not need a specification of measurement unit.

NOTE See the 8.3.

8.9.8 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 convexHull

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

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

8.9.10 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 coordinateDimension

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

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

8.9.12 difference

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

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 dimension

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

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

8.9.14 metricDistance

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

```
                geom2: ogc:geomLiteral): xsd:double
```

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 distance, but does not need a specification of measurement unit.

8.9.15 distance

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

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 metricDistance, which does not need a specification of measurement unit.

NOTE See the 8.3.

8.9.16 envelope

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

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 geometryN

```
geof:geometryN (geom: ogc:geomLiteral, geomindex: xsd:integer): ogc:geomLiteral
```

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 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 intersection

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

8.9.20 is3D

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

The function geof:is3D Returns true if `geom` has z coordinate values. See [GeoSPARQL and Simple Features SFA-CA] for a description of GeoSPARQL's limitations: currently it supports 2.5D and geometric functions ignore Z values in calculations.

8.9.21 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.

8.9.22 isMeasured

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

The function geof:isMeasured returns true if `geom` has m coordinate values, as used e.g. in linear referencing. See [GeoSPARQL_and_Simple_Features__SFA-CA_] for a description of GeoSPARQL's limitations: at the moment it does not support linear referencing and M values are ignored.

8.9.23 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.

8.9.24 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 length but does not need a specification of measurement unit.

8.9.25 length

`geof:length (geom: ogc:geomLiteral, units: rdfs:Resource): xsd:double`

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

NOTE See the 8.3.

8.9.26 maxX

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

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

8.9.27 maxY

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

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

8.9.28 maxZ

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

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

8.9.29 minX

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

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

8.9.30 minY

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

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

8.9.31 minZ

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

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

8.9.32 numGeometries

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

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

NOTE This function returns 1 except in cases when it receives a collection type, such as an `sf:MultiPoint`, a `sf:MultiLineString`, a `sf:MultiPolygon`, or a `sf:GeometryCollection`. The function does not apply to `GeometryCollection` instances, as members of `GeometryCollection` instances are not ordered.

8.9.33 perimeter

```
geof:perimeter (geom: ogc:geomLiteral, unit: rdfs:Resource): xsd:double
```

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 `length`.

8.9.34 metricPerimeter

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

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

8.9.35 hasSpatialDimension

```
geof:hasSpatialDimension (geom: ogc:geomLiteral): xsd:integer
```

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

8.9.36 symDifference

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

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`.

8.9.37 transform

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

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.

8.9.38 union

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

This 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`.

Requirement 47:
/req/geometry-extension/srid-function. SRID Function
Implementations shall support `getSRID` as a SPARQL extension function.

8.9.39 getSRID

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

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

8.10 Spatial Aggregate Functions

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

Requirement 48:
/req/geometry-extension/sa-functions. Spatial Aggregate Functions
Implementations shall support `aggBoundingBox`, `aggBoundingCircle`, `aggCentroid`, `aggConcaveHull`, `aggConvexHull` and `aggUnion` as a SPARQL extension functions.

8.10.1 aggBoundingBox

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

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

NOTE This function is similar in nature to `envelope` used to calculate the bounding box of just one geometry.

8.10.2 aggBoundingCircle

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

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

NOTE This function is similar in name to `boundingCircle` used to calculate the bounding circle of just one geometry.

8.10.3 aggCentroid

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

The function `aggCentroid` calculates the centroid of the set of given geometries.

NOTE This function is similar in name to `centroid` used to calculate the centroid of just one geometry.

8.10.4 aggConcaveHull

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

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

NOTE This function is similar in name to `concaveHull` used to calculate the concave hull of just one geometry.

8.10.5 aggConvexHull

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

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

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

8.10.6 aggUnion

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

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

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

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) [35] 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.

Requirement 49:
/req/geometry-topology-extension. Geometry Topology Extension
Subject: Implementation Specification

requirement	/req/geometry-topology-extension/relate-query-function
requirement	/req/geometry-topology-extension/sf-query-functions
requirement	/req/geometry-topology-extension/eh-query-functions

requirement	/req/geometry-topology-extension/rcc8-query-functions

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

9.2.1 relate

Requirement 50:
/req/geometry-topology-extension/relate-query-function. Relate Query Function
Implementations shall support `relate` as a SPARQL extension function, consistent with the relate operator defined in Simple Features OGCSFACA ISO19125-1.

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

Figure 4

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.

Requirement 51:
/req/geometry-topology-extension/sf-query-functions. Simple Features Query Functions
Implementations shall support `sfEquals`, `sfDisjoint`, `sfIntersects`, `sfTouches`, `sfCrosses`, `sfWithin`, `sfContains` and `sfOverlaps` as SPARQL extension functions, consistent with their corresponding DE-9IM intersection patterns, as defined by Simple Features OGCSFACA ISO19125-1.

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

Table 6 — Simple Features Query Functions

Query Function	Defining DE-9IM Intersection Pattern
`geof:sfEquals(geom1: ogc:geomLiteral, geom2: ogc:geomLiteral): xsd:boolean`	`(TFFFTFFFT)`
`geof:sfDisjoint(geom1: ogc:geomLiteral, geom2: ogc:geomLiteral): xsd:boolean`	`(FF*FF****)`
`geof:sfIntersects(geom1: ogc:geomLiteral, geom2: ogc:geomLiteral): xsd:boolean`	`(FT******* F**T***** F***T****)`
`geof:sfTouches(geom1: ogc:geomLiteral, geom2: ogc:geomLiteral): xsd:boolean`	`(FT******* F**T***** F***T****)`

Table 6 *(continued)*

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

9.4 Egenhofer Relation Family

This clause establishes Requirements for the *Egenhofer* relation family. Consult references [34] and CATEG for a more detailed discussion of *Egenhofer* relations.

Requirement 52:
/req/geometry-topology-extension/eh-query-functions. Egenhofer Query Functions
Implementations shall support `ehEquals`, `ehDisjoint`, `ehMeet`, `ehOverlap`, `ehCovers`, `ehCoveredBy`, `ehInside` and `ehContains` as SPARQL extension functions, consistent with their corresponding DE-9IM intersection patterns, as defined by Simple Features OGCSFACA ISO19125-1.

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

Table 7 — Egenhofer Query Functions

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

9.5 RCC8 Relation Family

This clause establishes Requirements for the *RCC8* relation family. Consult references QUAL and [36] for a more detailed discussion of *RCC8* relations.

Requirement 53:
/req/geometry-topology-extension/rcc8-query-functions. RCC8 Query Functions

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

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

Table 8 — RCC8 Query Functions

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

10 RDFS Entailment Extension

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`.

Requirement 54:
/req/rdfs-entailment-extension. RDFS Entailment Extension
Subject: Implementation Specification

requirement	/req/rdfs-entailment-extension/bgp-rdfs-ent
requirement	/req/rdfs-entailment-extension/wkt-geometry-types
requirement	/req/rdfs-entailment-extension/gml-geometry-types

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.

Requirement 55:
/req/rdfs-entailment-extension/bgp-rdfs-ent. Basic Graph Pattern
Basic graph pattern matching shall use the semantics defined by the RDFS Entailment Regime SPARQLENT.

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 `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 ;
.
```

Requirement 56:
/req/rdfs-entailment-extension/wkt-geometry-types. 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.

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 OGC07-036.

10.4.1 Geometry Class Hierarchy

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

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

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

Requirement 57:
/req/rdfs-entailment-extension/gml-geometry-types. GML Geometry Types
Implementations shall support graph patterns involving terms from an RDFS/OWL class hierarchy of geometry types consistent with the GML schema that implements `GM_Object` using the specified *version* of GML OGC07-036.

11 Query Rewrite Extension

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 [Geometry_Extension] to establish the existence of direct topological predicates defined in [Topology_Vocabulary_Extension]. 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.

Requirement 58:
/req/query-rewrite-extension. Query Rewrite Extension
Subject: Implementation Specification

requirement	/req/query-rewrite-extension/sf-query-rewrite
requirement	/req/query-rewrite-extension/eh-query-rewrite
requirement	/req/query-rewrite-extension/rcc8-query-rewrite

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 [Geometry_Extension]. `ogc:asGeomLiteral` is used to indicate one of the properties that link `Geometry` instances to serializations, such as `asWKT` or `asGeoJSON`. The variables `?so1` and `?so2` represent `SpatialObject` instances (`Feature` or `Geometry` instances), `?g1` and `?g2` `Geometry` instances only and `?g1Serial` and `?g2Serial` represent `Geometry` instance serializations, e.g. `asWKT` etc. literals.

```
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.

Requirement 59:
/req/query-rewrite-extension/sf-query-rewrite. Simple Features Query Transformation Rules
Basic graph pattern matching shall use the semantics defined by the RIF Core Entailment Regime SPARQLENT for the RIF rules RIFCORE `sfEquals`, `sfDisjoint`, `sfIntersects`, `sfTouches`, `sfCrosses`, `sfWithin`, `sfContains` and `sfOverlaps`.

Table 9 — Simple Features Query Transformation Rules

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

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.

Requirement 60:
/req/query-rewrite-extension/eh-query-rewrite. Egenhofer Query Transformation Rules
Basic graph pattern matching shall use the semantics defined by the RIF Core Entailment Regime SPARQLENT for the RIF rules RIFCORE `ehEquals`, `ehDisjoint`, `ehMeet`, `ehOverlap`, `ehCovers`, `ehCoveredBy`, `ehInside` and `ehContains`.

Table 10 — Egenhofer Query Transformation Rules

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

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.

Requirement 61:
/req/query-rewrite-extension/rcc8-query-rewrite. RCC8 Query Transformation Rules
Basic graph pattern matching shall use the semantics defined by the RIF Core Entailment Regime SPARQLENT for the RIF rules RIFCORE `rcc8eq`, `rcc8dc`, `rcc8ec`, `rcc8po`, `rcc8tppi`, `rcc8tpp`, `rcc8ntpp` and `rcc8ntppi`.

Table 11 — RCC8 Query Transformation Rules

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

11.5 Special Considerations

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

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

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

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

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

12 Future Work

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

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

Plans for future GeoSPARQL releases have been mooted but won't be articulated here, instead they will be discussed and decided 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.

Future versions of GeoSPARQL published by the OGC will be proposed for ISO co-adoption link this version.

Annex A
(normative)

Abstract Test Suite

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

Requirement A.1:
/conf/core. Core
Target: /req/core
Abstract-test: /conf/core/sparql-protocol
Abstract-test: /conf/core/spatial-object-class
Abstract-test: /conf/core/feature-class
Abstract-test: /conf/core/spatial-object-collection-class
Abstract-test: /conf/core/feature-collection-class
Abstract-test: /conf/core/spatial-object-properties

A.1.1 SPARQL

Requirement A.2:
/conf/core/sparql-protocol
Target: /req/core/sparql-protocol
Test-purpose: Check conformance with this requirement
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.
Test-method-type: Capabilities
Reference: SPARQLPROT

A.1.2 RDF Classes and Properties

Requirement A.3:
/conf/core/spatial-object-class
Target: /req/core/spatial-object-class
Test-purpose: Check conformance with this requirement
Test-method: Verify that queries involving `SpatialObject` return the correct result on a test dataset.
Test-method-type: Capabilities
Reference: Class/ geo

Requirement A.4:
/conf/core/feature-class
Target: /req/core/feature-class

Test-purpose: Check conformance with this requirement
Test-method: Verify that queries involving `Feature` return the correct result on a test dataset.
Test-method-type: Capabilities
Reference: Class/ geo

Requirement A.5:
/conf/core/spatial-object-collection-class
Target: /req/core/spatial-object-collection-class
Test-purpose: Check conformance with this requirement
Test-method: Verify that queries involving `SpatialObjectCollection` return the correct result on a test dataset.
Test-method-type: Capabilities
Reference: Class/ geo

Requirement A.6:
/conf/core/feature-collection-class
Target: /req/core/feature-collection-class
Test-purpose: Check conformance with this requirement
Test-method: Verify that queries involving `FeatureCollection` return the correct result on a test dataset.
Test-method-type: Capabilities
Reference: Class/ geo

Requirement A.7:
/conf/core/spatial-object-properties
Target: /req/core/spatial-object-properties
Test-purpose: Check conformance with this requirement
Test-method: Verify that queries involving these following properties return the correct result for a test dataset: `hassize`, `hasmetricsize`, `haslength`, `hasmetriclength`, `hasperimeterlength`, `hasmetricperimeterlength`, `hasarea`, `hasmetricarea`, `hasvolume` and `hasmetricvolume`
Test-method-type: Capabilities
Reference: 6.3

A.2 Conformance Class: Topology Vocabulary Extension

Requirement A.8:
/conf/topology-vocab-extension. Topology Vocabulary Extension
Target: /req/topology-vocab-extension
Abstract-test: /conf/topology-vocab-extension/sf-spatial-relations
Abstract-test: /conf/topology-vocab-extension/eh-spatial-relations
Abstract-test: /conf/topology-vocab-extension/rcc8-spatial-relations

A.2.1 Simple Features Relation Family

Requirement A.9:
/conf/topology-vocab-extension/sf-spatial-relations
Target: /req/topology-vocab-extension/sf-spatial-relations
Test-purpose: Check conformance with this requirement
Test-method: Verify that queries involving the following properties return the correct result for a test dataset: `sfequals`, `sfdisjoint`, `sfintersects`, `sftouches`, `sfcrosses`, `sfwithin`, `sfcontains` and `sfoverlaps`
Test-method-type: Capabilities
Reference: Table 2

A.2.2 Egenhofer Relation Family

Requirement A.10:
/conf/topology-vocab-extension/eh-spatial-relations
Target: /req/topology-vocab-extension/eh-spatial-relations
Test-purpose: Check conformance with this requirement

Test-method: Verify that queries involving the following properties return the correct result for a test dataset: *ehequals*, *ehdisjoint*, *ehmeet*, *ehoverlap*, *ehcovers*, *ehcoveredby*, *ehinside* and *ehcontains*
Test-method-type: Capabilities
Reference: Table 3

A.2.3 RCC8 Relation Family

Requirement A.11:
/conf/topology-vocab-extension/rcc8-spatial-relations
Target: /req/topology-vocab-extension/rcc8-spatial-relations
Test-purpose: Check conformance with this requirement
Test-method: Verify that queries involving the following properties return the correct result for a test dataset: *rcc8eq*, *rcc8dc*, *rcc8ec*, *rcc8po*, *rcc8tppi*, *rcc8tpp*, *rcc8ntpp*, *rcc8ntppi*
Test-method-type: Capabilities
Reference: Table 4

A.3 Conformance Class: Geometry Extension

Requirement A.12:
/conf/geometry-extension. Geometry Extension
Subject: Geometry
Target: /req/geometry-extension
Abstract-test: /conf/geometry-extension/geometry-class
Abstract-test: /conf/geometry-extension/geometry-collection-class
Abstract-test: /conf/geometry-extension/feature-properties
Abstract-test: /conf/geometry-extension/geometry-properties
Abstract-test: /conf/geometry-extension/query-functions
Abstract-test: /conf/geometry-extension/srid-function
Abstract-test: /conf/geometry-extension/sa-functions
Abstract-test: /conf/geometry-extension/wkt-literal
Abstract-test: /conf/geometry-extension/wkt-literal-default-srs
Abstract-test: /conf/geometry-extension/wkt-axis-order
Abstract-test: /conf/geometry-extension/wkt-literal-empty
Abstract-test: /conf/geometry-extension/geometry-as-wkt-literal
Abstract-test: /conf/geometry-extension/asWKT-function
Abstract-test: /conf/geometry-extension/gml-literal
Abstract-test: /conf/geometry-extension/gml-literal-empty
Abstract-test: /conf/geometry-extension/gml-profile
Abstract-test: /conf/geometry-extension/geometry-as-gml-literal
Abstract-test: /conf/geometry-extension/asGML-function
Abstract-test: /conf/geometry-extension/geojson-literal
Abstract-test: /conf/geometry-extension/geojson-literal-srs
Abstract-test: /conf/geometry-extension/geojson-literal-empty
Abstract-test: /conf/geometry-extension/geometry-as-geojson-literal
Abstract-test: /conf/geometry-extension/asGeoJSON-function
Abstract-test: /conf/geometry-extension/kml-literal
Abstract-test: /conf/geometry-extension/kml-literal-srs
Abstract-test: /conf/geometry-extension/kml-literal-empty
Abstract-test: /conf/geometry-extension/geometry-as-kml-literal
Abstract-test: /conf/geometry-extension/asKML-function

This Conformance Class applies to non-DGGS geometries. See DGGS Conformance Class/ Geometry Extension — DGGS for DGGS geometries.

A.3.1 Tests for all Serializations except DGGS

Requirement A.13:

/conf/geometry-extension/geometry-class
Target: /req/geometry-extension/geometry-class
Test-purpose: Check conformance with this requirement
Test-method: Verify that queries involving `Geometry` *return the correct result on a test dataset*
Test-method-type: Capabilities
Reference: `Geometry`

Requirement A.14:
/conf/geometry-extension/geometry-collection-class
Target: /req/geometry-extension/geometry-collection-class
Test-purpose: Check conformance with this requirement
Test-method: Verify that queries involving `Geometry Collection` *return the correct result on a test dataset*
Test-method-type: Capabilities
Reference: `Geometry Collection`

Requirement A.15:
/conf/geometry-extension/feature-properties
Target: /req/geometry-extension/feature-properties
Test-purpose: Check conformance with this requirement
Test-method: Verify that queries involving the following properties return the correct result for a test dataset: `hasgeometry`, `hasdefaultgeometry`, `haslength`, `hasarea`, `hasvolume` `hascentroid`, `hasboundingbox` *and* `hasspatialresolution`
Test-method-type: Capabilities
Reference: 6.4

Requirement A.16:
/conf/geometry-extension/geometry-properties
Target: /req/geometry-extension/geometry-properties
Test-purpose: Check conformance with this requirement
Test-method: Verify that queries involving these properties return the correct result for a test dataset: `dimension`, `coordinatedimension`, `hasspatialdimension`, `hasspatialresolution`, `hasmetricspatialresolution`, `hasspatialaccuracy`, `hasmetricspatialaccuracy`, `isempty`, `issimple` *and* `hasserialization`
Test-method-type: Capabilities
Reference: 8.7

Requirement A.17:
/conf/geometry-extension/query-functions
Target: /req/geometry-extension/query-functions
Test-purpose: Check conformance with this requirement
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:boundary geof:boundingCircle, geof:metricBuffer, geof:buffer, geof:centroid, geof:convexHull, geof:concaveHull, geof:coordinateDimension, geof:difference, geof:dimension, geof:metricDistance, geof:distance, geof:envelope, geof:geometryType, geof:intersection, geof:is3D, geof:isEmpty, geof:isMeasured, geof:isSimple, geof:hasSpatialDimension, geof:symDifference, geof:transform and geof:union.
Test-method-type: Capabilities reference: 8.9

Requirement A.18:
/conf/geometry-extension/srid-function
Target: /req/geometry-extension/srid-function
Test-purpose: Check conformance with this requirement
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.
Test-method-type: Capabilities
Reference: Function/ geof

Requirement A.19:
/conf/geometry-extension/sa-functions

Target: /req/geometry-extension/sa-functions
Test-purpose: Check conformance with this requirement
Test-method: Verify that a set of SPARQL queries involving each of the following functions returns the correct result for a test dataset: geof:aggBoundingBox, geof:aggBoundingCircle, geof:aggCentroid, geof:aggConcaveHull, geof:aggConvexHull and geof:aggUnion
Test-method-type: Capabilities
Reference: 8.10

A.3.2 WKT Serialization

Requirement A.20:
/conf/geometry-extension/wkt-literal
Target: /req/geometry-extension/wkt-literal
Test-purpose: Check conformance with this requirement
Test-method: Verify that queries involving WKT Literal values return the correct result for a test dataset.
Test-method-type: Capabilities
Reference: [datatype-wktLiteral]

Requirement A.21:
/conf/geometry-extension/wkt-literal-default-srs
Target: /req/geometry-extension/wkt-literal-default-srs
Test-purpose: Check conformance with this requirement
Test-method: Verify that queries involving WKT Literal values without an explicit encoded SRS IRI return the correct result for a test dataset.
Test-method-type: Capabilities
Reference: Requirement 18

Requirement A.22:
/conf/geometry-extension/wkt-axis-order
Target: /req/geometry-extension/wkt-axis-order
Test-purpose: Check conformance with this requirement
Test-method: Verify that queries involving WKT Literal values return the correct result for a test dataset.
Test-method-type: Capabilities
Reference: Requirement 19

Requirement A.23:
/conf/geometry-extension/wkt-literal-empty
Target: /req/geometry-extension/wkt-literal-empty
Test-purpose: Check conformance with this requirement
Test-method: Verify that queries involving empty WKT Literal values return the correct result for a test dataset.
Test-method-type: Capabilities
Reference: Requirement 20

Requirement A.24:
/conf/geometry-extension/geometry-as-wkt-literal
Target: /req/geometry-extension/geometry-as-wkt-literal
Test-purpose: Check conformance with this requirement
Test-method: Verify that queries involving the `aswkt` property return the correct result for a test dataset.
Test-method-type: Capabilities
Reference: [geo_aswkt]

Requirement A.25:
/conf/geometry-extension/asWKT-function
Target: /req/geometry-extension/asWKT-function
Test-purpose: Check conformance with this requirement
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.
Test-method-type: Capabilities

Reference: Function/ geof

A.3.3 GML Serialization

Requirement A.26:
/conf/geometry-extension/gml-literal
Target: /req/geometry-extension/gml-literal
Test-purpose: Check conformance with this requirement
Test-method: Verify that queries involving `gmlliteral` *values return the correct result for a test dataset.*
Test-method-type: Capabilities
Reference: [datatype-gmlLiteral]

Requirement A.27:
/conf/geometry-extension/gml-literal-empty
Target: /req/geometry-extension/gml-literal-empty
Test-purpose: Check conformance with this requirement
Test-method: Verify that queries involving empty `gmlliteral` *values return the correct result for a test dataset.*
Test-method-type: Capabilities
Reference: Requirement 24

Requirement A.28:
/conf/geometry-extension/gml-profile
Target: /req/geometry-extension/gml-profile
Test-purpose: Check conformance with this requirement
Test-method: Examine the implementation's documentation to verify that the supported GML profiles are documented.
Test-method-type: Capabilities
Reference: Requirement 25

Requirement A.29:
/conf/geometry-extension/geometry-as-gml-literal
Target: /req/geometry-extension/geometry-as-gml-literal
Test-purpose: Check conformance with this requirement
Test-method: Verify that queries involving the `asgml` *property return the correct result for a test dataset.*
Test-method-type: Capabilities
Reference: Requirement 26

Requirement A.30:
/conf/geometry-extension/asGML-function
Target: /req/geometry-extension/asGML-function
Test-purpose: Check conformance with this requirement
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.
Test-method-type: Capabilities
Reference: Function/ geof

A.3.4 GeoJSON Serialization

Requirement A.31:
/conf/geometry-extension/geojson-literal
Target: /req/geometry-extension/geojson-literal
Test-purpose: Check conformance with this requirement
Test-method: Verify that queries involving `geojsonliteral` *values return the correct result for a test dataset.*
Test-method-type: Capabilities
Reference: [datatype-geoJSONLiteral]

Requirement A.32:
/conf/geometry-extension/geojson-literal-srs

Target: /req/geometry-extension/geojson-literal-srs
Test-purpose: Check conformance with this requirement
Test-method: Verify that queries involving `geojsonliteral` *values without an explicit encoded SRS IRI return the correct result for a test dataset.*
Test-method-type: Capabilities
Reference: Requirement 29

Requirement A.33:
/conf/geometry-extension/geojson-literal-empty
Target: /req/geometry-extension/geojson-literal-empty
Test-purpose: Check conformance with this requirement
Test-method: Verify that queries involving empty `geojsonliteral` *values return the correct result for a test dataset.*
Test-method-type: Capabilities
Reference: Requirement 30

Requirement A.34:
/conf/geometry-extension/geometry-as-geojson-literal
Target: /req/geometry-extension/geometry-as-geojson-literal
Test-purpose: Check conformance with this requirement
Test-method: Verify that queries involving the `asgeojson` *property return the correct result for a test dataset.*
Test-method-type: Capabilities
Reference: [req_geometry_extension_geojson-as-geojson-literal]

Requirement A.35:
/conf/geometry-extension/asGeoJSON-function
Target: /req/geometry-extension/asGeoJSON-function
Test-purpose: Check conformance with this requirement
Test-method: Verify that a set of SPARQL queries involving the geof:asGeoJSON function returns the correct result for a test dataset when using the specified serialization and version.
Test-method-type: Capabilities
Reference: Function/ geof

A.3.5 KML Serialization

Requirement A.36:
/conf/geometry-extension/kml-literal
Target: /req/geometry-extension/kml-literal
Test-purpose: Check conformance with this requirement
Test-method: Verify that queries involving `kmlliteral` *values return the correct result for a test dataset.*
Test-method-type: Capabilities
Reference: [datatype-kmlLiteral]

Requirement A.37:
/conf/geometry-extension/kml-literal-srs
Target: /req/geometry-extension/kml-literal-srs
Test-purpose: Check conformance with this requirement
Test-method: Verify that queries involving `kmlliteral` *values without an explicit encoded SRS IRI return the correct result for a test dataset.*
Test-method-type: Capabilities
Reference: Requirement 34

Requirement A.38:
/conf/geometry-extension/kml-literal-empty
Target: /req/geometry-extension/kml-literal-empty
Test-purpose: Check conformance with this requirement
Test-method: Verify that queries involving empty `kmlliteral` *values return the correct result for a test dataset.*
Test-method-type: Capabilities

Reference: Requirement 35

Requirement A.39:
/conf/geometry-extension/geometry-as-kml-literal
Target: /req/geometry-extension/geometry-as-kml-literal
Test-purpose: Check conformance with this requirement
Test-method: Verify that queries involving the `askml` property return the correct result for a test dataset.
Test-method-type: Capabilities
Reference: Requirement 36

Requirement A.40:
/conf/geometry-extension/asKML-function
Target: /req/geometry-extension/asKML-function
Test-purpose: Check conformance with this requirement
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.
Test-method-type: Capabilities
Reference: Function/ geof

A.4 DGGS Conformance Class: Geometry Extension — DGGS

This conformance Class applies only to DGGS geometries. See Conformance Class/ Geometry Extension for other geometries.

Requirement A.41:
/conf/geometry-extension-dggs. Geometry Extension DGGS
Target: /req/geometry-extension-dggs
Abstract-test: /conf/geometry-extension-dggs/query-functions
Abstract-test: /conf/geometry-extension-dggs/query-functions-non-sf
Abstract-test: /conf/geometry-extension-dggs/srid-function
Abstract-test: /conf/geometry-extension-dggs/sa-functions
Abstract-test: /conf/geometry-extension-dggs/dggs-literal
Abstract-test: /conf/geometry-extension-dggs/dggs-literal-empty
Abstract-test: /conf/geometry-extension-dggs/geometry-as-dggs-literal
Abstract-test: /conf/geometry-extension-dggs/asDGGS-function

A.4.1 Tests for DGGS Serializations

Requirement A.42:
/conf/geometry-extension-dggs/query-functions
Target: /req/geometry-extension-dggs/query-functions
Test-purpose: Check conformance with this requirement
Test-method: Verify that implementations 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.test-method-type:: Capabilities

Requirement A.43:
/conf/geometry-extension-dggs/query-functions-non-sf
Target: /req/geometry-extension-dggs/query-functions-non-sf
Test-purpose: Check conformance with this requirement
Test-method: Verify that implementations 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.

Requirement A.44:
/conf/geometry-extension-dggs/srid-function
Target: /req/geometry-extension-dggs/srid-function

Test-purpose: Check conformance with this requirement
Test-method: Verify that Implementations shall support geof:getSRID as a SPARQL extension function for DGGS geometry literals.
Reference: [[_function_geofgetsrid]]

Requirement A.45:
/conf/geometry-extension-dggs/sa-functions
Target: /req/geometry-extension-dggs/sa-functions
Test-purpose: Check conformance with this requirement
Test-method: Verify that implementations 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.
Test-method-type: Capabilities
Reference: 8.10

A.4.2 DGGS Serialization

Requirement A.46:
/conf/geometry-extension-dggs/dggs-literal
Target: /req/geometry-extension-dggs/dggs-literal
Test-purpose: Check conformance with this requirement
Test-method: Verify that queries involving empty `dggsLiteral` values return the correct result for a test dataset.
Test-method-type: Capabilities
Reference: [datatype-dggsLiteral]

Requirement A.47:
/conf/geometry-extension-dggs/dggs-literal-empty
Target: /req/geometry-extension-dggs/dggs-literal-empty
Test-purpose: Check conformance with this requirement
Test-method: Verify that queries involving empty `dggsLiteral` values return the correct result for a test dataset.
Test-method-type: Capabilities
Reference: Requirement 40

Requirement A.48:
/conf/geometry-extension-dggs/geometry-as-dggs-literal
Target: /req/geometry-extension-dggs/geometry-as-dggs-literal
Test-purpose: Check conformance with this requirement
Test-method: Verify that queries involving the `asDGGS` property return the correct result for a test dataset.
Test-method-type: Capabilities
Reference: Requirement 41

Requirement A.49:
/conf/geometry-extension-dggs/asDGGS-function
Target: /req/geometry-extension-dggs/asDGGS-function
Test-purpose: Check conformance with this requirement
Test-method: Verify that a set of SPARQL queries involving the `asDGGS` function returns the correct result for a test dataset when using the specified serialization and version.
Test-method-type: Capabilities
Reference: Function/ geof

A.5 Conformance Class: Geometry Topology Extension

Requirement A.50:
/conf/geometry-topology-extension. Geometry Topology Extension
Target: /req/geometry-topology-extension
Abstract-test: /conf/geometry-topology-extension/relate-query-function
Abstract-test: /conf/geometry-topology-extension/sf-query-functions

Abstract-test: /conf/geometry-topology-extension/eh-query-functions
Abstract-test: /conf/geometry-topology-extension/rcc8-query-functions

A.5.1 Tests for all relation families

Requirement A.51:
/conf/geometry-topology-extension/relate-query-function
Target: /req/geometry-topology-extension/relate-query-function
Test-purpose: Check conformance with this requirement
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.
Test-method-type: Capabilities
Reference: `relate`

A.5.2 Simple Features Relation Family

Requirement A.52:
/conf/geometry-topology-extension/sf-query-functions
Target: /req/geometry-topology-extension/sf-query-functions
Test-purpose: Check conformance with this requirement
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`.
Test-method-type: Capabilities
Reference: Table 6

A.5.3 Egenhofer Relation Family

Requirement A.53:
/conf/geometry-topology-extension/eh-query-functions
Target: /req/geometry-topology-extension/eh-query-functions
Test-purpose: Check conformance with this requirement
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`.
Test-method-type: Capabilities
Reference: Table 7

A.5.4 RCC8 Relation Family

Requirement A.54:
/conf/geometry-topology-extension/rcc8-query-functions
Target: /req/geometry-topology-extension/rcc8-query-functions
Test-purpose: Check conformance with this requirement
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`.
Test-method-type: Capabilities
Reference: Table 8

A.6 Conformance Class: RDFS Entailment Extension

Requirement A.55:
/conf/rdfs-entailment-extension. RDFS Entailment Extension
Target: /req/rdfs-entailment-extension
Abstract-test: /conf/rdfs-entailment-extension/bgp-rdfs-ent
Abstract-test: /conf/rdfs-entailment-extension/wkt-geometry-types

Abstract-test: /conf/rdfs-entailment-extension/gml-geometry-types

A.6.1 Tests for all implementations

Requirement A.56:
/conf/rdfs-entailment-extension/bgp-rdfs-ent
Target: /req/rdfs-entailment-extension/bgp-rdfs-ent
Test-purpose: Check conformance with this requirement
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.
Test-method-type: Capabilities
Reference: [10.2]

A.6.2 WKT Serialization

Requirement A.57:
/conf/rdfs-entailment-extension/wkt-geometry-types
Target: /req/rdfs-entailment-extension/wkt-geometry-types
Test-purpose: Check conformance with this requirement
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.
Test-method-type: Capabilities
Reference: [10.3]

A.6.3 GML Serialization

Requirement A.58:
/conf/rdfs-entailment-extension/gml-geometry-types
Target: /req/rdfs-entailment-extension/gml-geometry-types
Test-purpose: Check conformance with this requirement
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.
Test-method-type: Capabilities
Reference: [10.4]

A.7 Conformance Class: Query Rewrite Extension

Requirement A.59:
/conf/query-rewrite-extension. Query Rewrite Extension
Target: /req/query-rewrite-extension
Abstract-test: /conf/query-rewrite-extension/sf-query-rewrite
Abstract-test: /conf/query-rewrite-extension/eh-query-rewrite
Abstract-test: /conf/query-rewrite-extension/rcc8-query-rewrite

A.7.1 Simple Features Relation Family

Requirement A.60:
/conf/query-rewrite-extension/sf-query-rewrite
Target: /req/query-rewrite-extension/sf-query-rewrite
Test-purpose: Check conformance with this requirement
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`.
Test-method-type: Capabilities
Reference: [11.2]

A.7.2 Egenhofer Relation Family

Requirement A.61:
/conf/query-rewrite-extension/eh-query-rewrite
Target: /req/query-rewrite-extension/eh-query-rewrite
Test-purpose: Check conformance with this requirement
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`.
Test-method-type: Capabilities
Reference: 11.3

A.7.3 RCC8 Relation Family

Requirement A.62:
/conf/query-rewrite-extension/rcc8-query-rewrite
Target: /req/query-rewrite-extension/rcc8-query-rewrite
Test-purpose: Check conformance with this requirement
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`.
Test-method-type: Capabilities
Reference: 11.4

Annex B
(normative)

Functions Summary

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 `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 [Geometry_Serializations] for the individual specification references.

B.1 Functions Summary Table

Table B.1 — GeoSPARQL Functions Summary

Simple Features Functions				
Function	**Input Datatypes**	**Input Subtypes**	**Output Datatype**	**Output Subtype**
sfcontains	2x ogc:geomLiteral	1x Polygon, 1x Geometry	xsd:boolean	
sfcrosses	2x ogc:geomLiteral	1x Point or LineString, 1 x LineString or Polygon	xsd:boolean	
sfdisjoint	2x ogc:geomLiteral	2x Geometry	xsd:boolean	
sfequals	2x ogc:geomLiteral	2x Geometry	xsd:boolean	
sfintersects	2x ogc:geomLiteral	2x Polygon	xsd:boolean	
sfoverlaps	2x ogc:geomLiteral	2x Point or 2x LineString or 2x Polygon	xsd:boolean	
sftouches	2x ogc:geomLiteral	2x Geometry but not Point	xsd:boolean	
sfwithin	2x ogc:geomLiteral	1x Geometry, 1x Polygon	xsd:boolean	
Egenhofer Functions				
Function	**Input Datatypes**	**Input Subtypes**	**Output Datatype**	**Output Subtype**
ehcontains	2x ogc:geomLiteral	1x Polygon, 1x Geometry	xsd:boolean	
ehcoveredby	2x ogc:geomLiteral	1x Polygon, 1x Geometry	xsd:boolean	
ehcovers	2x ogc:geomLiteral	1x Polygon, 1x Geometry	xsd:boolean	
ehdisjoint	2x ogc:geomLiteral	2x Geometry	xsd:boolean	
ehequals	2x ogc:geomLiteral	2x Geometry	xsd:boolean	

Table B.1 *(continued)*

ehmeet	2x ogc:geomLiteral	2x Geometry but not Point	xsd:boolean	
ehoverlap	2x ogc:geomLiteral	2x Geometry	xsd:boolean	
ehinside	2x ogc:geomLiteral	2x Geometry	xsd:boolean	
Region Connection Calculus Functions				
Function	**Input Datatypes**	**Input Subtypes**	**Output Datatype**	**Output Subtype**
rcc8dcc	2x ogc:geomLiteral	2x Polygon	xsd:boolean	
rcc8ecc	2x ogc:geomLiteral	2x Polygon	xsd:boolean	
rcc8eq	2x ogc:geomLiteral	2x Polygon	xsd:boolean	
rcc8ntpp	2x ogc:geomLiteral	2x Polygon	xsd:boolean	
rcc8ntppi	2x ogc:geomLiteral	2x Polygon	xsd:boolean	
rcc8po	2x ogc:geomLiteral	2x Polygon	xsd:boolean	
rcc8tpp	2x ogc:geomLiteral	2x Polygon	xsd:boolean	
rcc8tppi	2x ogc:geomLiteral	2x Polygon	xsd:boolean	
Spatial Aggregate Functions				
Function	**Input Datatypes**	**Input Subtypes**	**Output Datatype**	**Output Subtype**
aggboundingbox	1 or more ogc:geomLiteral		ogc:geomLiteral	square Polygon (not DGGS), CellList (DGGS)
aggboundingcircle	1 or more ogc:geomLiteral		ogc:geomLiteral	Polygon (not DGGS) CellList (DGGS)
aggcentroid	1 or more ogc:geomLiteral		ogc:geomLiteral	Point (not DGGS), Cell (DGGS)
aggconcavehull	1 or more ogc:geomLiteral		ogc:geomLiteral	Polygon (not DGGS), CellList (DGGS)
aggconvexhull	1 or more ogc:geomLiteral		ogc:geomLiteral	Polygon (not DGGS), CellList (DGGS)
aggunion	1 or more ogc:geomLiteral		ogc:geomLiteral	Polygon (not DGGS), CellList (DGGS)
Non-topological Query Functions				
Function	**Input Datatypes**	**Input Subtypes**	**Output Datatype**	**Output Subtype**
metricarea	1x ogc:geomLiteral	Polygon	xsd:double	
area	1x ogc:geomLiteral	Polygon	xsd:double	
boundary	1x ogc:geomLiteral	Geometry	ogc:geomLiteral	LineString (not DGGS), OrderedCellList (DGGS)
buffer	1x ogc:geomLiteral, 1x xsd:double, 1x xsd:anyURI	any	ogc:geomLiteral	(Multi)Polygon (not DGGS), CellList (DGGS)
convexhull	1x ogc:geomLiteral	Geometry	ogc:geomLiteral	LineString (not DGGS)
coordinatedimension	1x ogc:geomLiteral	Geometry	xsd:integer	
difference	2x ogc:geomLiteral	2x Geometry	ogc:geomLiteral	(Multi)Polygon (not DGGS), CellList (DGGS)
dimension	1x ogc:geomLiteral	Geometry	xsd:double	
metricdistance	2x ogc:geomLiteral, 1x xsd:anyURI	2x Geometry	xsd:double	
distance	2x ogc:geomLiteral, 1x xsd:anyURI	2x Geometry	rdfs:Resource	
envelope	1x ogc:geomLiteral, 1x xsd:anyURI	Geometry	ogc:geomLiteral	(Multi)Polygon (not DGGS), CellList (DGGS)

Table B.1 *(continued)*

geometryn	1x ogc:geomLiteral	GeometryCollection (not DGGS)	xsd:double	
geometrytype	1x ogc:geomLiteral	Geometry	xsd:anyURI	
getsrid	1x ogc:geomLiteral	Geometry	xsd:anyURI	
intersection	2x ogc:geomLiteral	2x Geometry	ogc:geomLiteral	Polygon (not DGGS), CellList (DGGS)
is3d	1x ogc:geomLiteral	Geometry	xsd:boolean	
isempty	1x ogc:geomLiteral	Geometry	xsd:boolean	
ismeasured	1x ogc:geomLiteral	Geometry	xsd:boolean	
issimple	1x ogc:geomLiteral	Geometry	xsd:boolean	
metriclength	1x ogc:geomLiteral	Geometry	xsd:double	
length	1x ogc:geomLiteral	Geometry	rdfs:Resource	
numgeometries	1x ogc:geomLiteral	Geometry (not DGGS)	xsd:double	
metricperimeter	1x ogc:geomLiteral	Geometry	xsd:double	
perimeter	1x ogc:geomLiteral	Geometry	rdfs:Resource	
hasspatialdimension	1x ogc:geomLiteral	Geometry	xsd:integer	
symdifference	2x ogc:geomLiteral	2x Geometry	ogc:geomLiteral	(Multi)Polygon (not DGGS), CellList DGGS)
transform	1x ogc:geomLiteral, 1x xsd:anyURI	Geometry	ogc:geomLiteral	Geometry
union	2x ogc:geomLiteral	2x Geometry	ogc:geomLiteral	Polygon (not DGGS), CellList (DGGS)
Serialization Functions				
Function	**Input Datatypes**	**Input Subtypes**	**Output Datatype**	**Output Subtype**
asdggs	1x ogc:geomLiteral	Geometry	geo:dggsLiteral	
asgeojson	1x ogc:geomLiteral	Geometry	geo:geoJSONLiteral	
asgml	1x ogc:geomLiteral, 1x xsd:string	Geometry	geo:gmlLiteral	
askml	1x ogc:geomLiteral	Geometry	geo:kmlLiteral	
aswkt	1x ogc:geomLiteral	Geometry	geo:wktLiteral	
Extent Functions				
Function	**Input Datatypes**	**Input Subtypes**	**Output Datatype**	**Output Subtype**
getsrid	1x ogc:geomLiteral	Geometry	xsd:anyURI	
maxx	1x ogc:geomLiteral	Geometry	xsd:double	
maxy	1x ogc:geomLiteral	Geometry	xsd:double	
maxz	1x ogc:geomLiteral	Geometry	xsd:double	
minx	1x ogc:geomLiteral	Geometry	xsd:double	
miny	1x ogc:geomLiteral	Geometry	xsd:double	
minz	1x ogc:geomLiteral	Geometry	xsd:double	
Other Functions				
Function	**Input Datatypes**	**Input Subtypes**	**Output Datatype**	**Output Subtype**
relate	2x ogc:geomLiteral		xsd:string	

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 B.2 — GeoSPARQL To SFA Mappings

GeoSPARQL Function	in 1.0	in 1.1	SFA
metricArea		x	Area
area		x	Area
			AsBinary
asWKT*	x	x	AsText
boundary	x	x	Boundary
buffer	x	x	Buffer
			Centroid
convexHull	x	x	ConvexHull
coordinateDimension		x	
difference	x	x	Difference
dimension		x	Dimension
metricDistance		x	Distance
distance	x	x	Distance
			EndPoint
envelope	x	x	Envelope
geometryN		x	GeometryN
geometryType		x	GeometryType
getSRID	x	x	SRID
			InteriorRingN
intersection	x	x	Intersection
is3D		x	
			IsClosed
isEmpty		x	IsEmpty
isMeasured		x	
			IsRing
isSimple		x	IsSimple
metricLength		x	Length
length		x	Length
maxX		x	
maxY		x	
maxZ		x	
minX		x	
minY		x	
minZ		x	
numGeometries		x	NumGeometries
			NumInteriorRing
			NumPoints
perimeterLength		x	
perimeter		x	
			PointN
			PointOnSurface
hasSpatialDimension		x	
			StartPoint
symDifference	x	x	SymDifference
transform		x	
union	x	x	Union
			X
			Y

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

Annex C
(informative)

Examples

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 `SpatialObject`.

C.1.1.1.1 Basic use

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

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

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

C.1.1.1.2 Size Properties

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

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

eg:moreton-island
    a geo:SpatialObject ;

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

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

```
.
```

Here a spatial object, Moreton Island, has the distance of its coastline given with two properties: `hasPerimeterLength` and `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[8)] and its `qudt:numericValue` and `qudt:unit` is just one of many possible ways to convey the value of `hasPerimeterLength` and any subproperty of `hasSize`.

C.1.1.2 `Feature`

The `Feature` class is defined in `Feature`.

C.1.1.2.1 Basic use

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

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

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

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

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

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

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

This example and the example below (B 1.1.2.4) show the same `Feature`, but with a different specification of its area. This example shows the recommended way to express size: by using a subproperty of `hasMetricSize` (in this case, `hasMetricArea`). These subproperties have fixed units based on meter (the unit of distance in the International System of Units).

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

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

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

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

Here a `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

```
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` woud not be easily able to differentiate them.

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

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

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

C.1.1.2.7 `Feature` with non-metric size

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

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

```
    ]
.
ex:Schoenus
  a qudt:Unit;
  skos:exactMatch dbp:Schoenus;
.
```

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

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

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

Here a `Feature` instance has two geometries, one indicated with the general property `hasGeometry` and a second indicated with the 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 `Geometry`.

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 C.1.1.2.2 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.

```
<https://example.com/geometry/y>
    a geo:Geometry ;
    skos:prefLabel "Geometry Y";
```

```
    geo:asWKT "MULTIPOLYGON (((149.06016 -35.23610, 149.060620 -35.236043, ... , 149.06016 -
35.23610)))"^^geo:wktLiteral ;
.
```

Here the `Geometry` instance has data in WKT form and, since no CRS is declared, WGS84 is the assumed, default, CRS.

C.1.1.3.2 A `Geometry` with multiple serializations

```
eg:x
    a geo:Feature ;
    skos:prefLabel "Feature X";
    geo:hasGeometry [
        geo:asWKT "<http://www.opengis.net/def/crs/EPSG/0/4326> MULTIPOLYGON (((-35.23610
149.06016, -35.236043 149.060620, ... , -35.23610 149.06016)))"^^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

```
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 `asWKT` and a scalar area indicated with the predicate `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 `hasMetricArea`, is possible since the domain of such properties is `SpatialObject`, the superclass of both `Feature` and `Geometry`.

C.1.1.4 `SpatialObjectCollection`

`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 `featurecollection` and `geometrycollection` classes.

C.1.1.5 `FeatureCollection`

This example shows a `FeatureCollection` instance containing 3 `Feature` instances.

```
ex:fc-x
    a geo:FeatureCollection ;
    dcterms:title "Feature Collection X" ;
    rdfs:member
        ex:feature-something ,
        ex:feature-other ,
        ex:feature-another ;
```

.

All of the GeoSPARQL collection classes are unordered since they are 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.

```
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 seralizations used in GeoSPARQL define the geometry type — point, polygon etc. *within* the literal, e.g. WKT can encode `POLYGON(())` or 'POINT()', however the *Simple Features Vocabulary* resource within GeoSPARQL 1.1 contains specialised Geometry RDF classes such as `sf:Polygon`, `sf:PolyhedralSurface` and others.

It may be appropriate to use these 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:

```
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, `hasCentroid` indicates is could be used with a `sf:Point` and `hasBoundingBox` indicates use with an `sf:Envelope`.

C.1.2 Properties

C.1.2.1 Spatial Object Properties

See the section C.1.1.1.2 above.

C.1.2.2 Feature Properties

This example shows a `Feature` instance with each of the properties defined in 6.4 used, except for the properties `hasMetricSize` and `hasSize`, that are intended to be used through their subproperties and `hasMetricPerimeterLength` and `hasPerimeterLength` which are examplified in C.1.1.1.2.

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

eg:x
    a geo:Feature ;
    skos:preferredLabel "Feature X" ;
    geo:hasGeometry [
        geo:asWKT "<http://www.opengis.net/def/crs/EPSG/0/4326> POLYGON ((-35.23610
149.06016, ... , -35.23610 149.06016)))"^^geo:wktLiteral ;
    ] ;
    geo:hasDefaultGeometry [
        geo:asWKT "<http://www.opengis.net/def/crs/EPSG/0/4326> POLYGON ((-35.2361 149.0601, .
.. , -35.2361 149.0601)))"^^geo:wktLiteral ;
    ] ;
    geo:hasMetricLength "355"^^xsd:double ;
    geo:hasLength [
        qudt:numericValue 355 ;
        qudt:unit <http://qudt.org/vocab/unit/M> ;  # meter
    ] ;
    geo:hasMetricArea "8.7E4"^^xsd:double ;
    geo:hasArea [
        qudt:numericValue 8.7 ;
        qudt:unit <http://qudt.org/vocab/unit/HA> ;  # hectare
    ] ;
    geo:hasMetricVolume "624432"^^xsd:double ;
    geo:hasVolume [
        qudt:numericValue 624432 ;
        qudt:unit <http://qudt.org/vocab/unit/M3> ;  # cubic meter
    ] ;
    geo:hasCentroid [
        geo:asWKT "POINT (149.06017 -35.23612)"^^geo:wktLiteral ;
    ] ;
    geo:hasBoundingBox [
        geo:asWKT "<http://www.opengis.net/def/crs/EPSG/0/4326> POLYGON ((-35.236 149.060, .
.. ,-35.236 149.060)))"^^geo:wktLiteral ;
    ] ;
    geo:hasMetricSpatialResolution "5"^^xsd:double ;
    geo:hasSpatialResolution [
        qudt:numericValue 5 ;
        qudt:unit <http://qudt.org/vocab/unit/M> ;  # meter
    ] ;
.
```

The properties defined for this example's `Feature` instance are vaguely aligned in that the values are not real but are not unrealistic either. It is outside the scope of GeoSPARQL to validate `Feature` instances' property values.

Note that this `Feature` has a 2D `Geometry` and yet a property indicating a scalar volume: `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 Standard Properties for geo/Geometry used.

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

```
eg:x
    a geo:Feature ;
    geo:hasGeometry [
        skos:prefLabel "Geometry Y" ;
        geo:dimension 2 ;
        geo:coordinateDimension 2 ;
        geo:hasSpatialDimension 2 ;
        geo:isEmpty false ;
        geo:isSimple true ;
        geo:hasSerialization "<http://www.opengis.net/def/crs/EPSG/0/4326> POLYGON ((-35.236
149.060, ... , -35.236 149.060)))"^^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 is empty 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 serlializations. The geometry values given are real geometry values and approximate Moreton Island in Queensland, Australia.

Note that the concrete DGGS serialization used is for example purposes only as it is not formally defined in GeoSPARQL.

```
eg:x
    a geo:Feature ;
    geo:hasGeometry [
        geo:asWKT """<http://www.opengis.net/def/crs/EPSG/0/4326>
            POLYGON ((
                -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
            ))"""^^geo:wktLiteral ;

        geo:asGML """<gml:Polygon
                srsName="http://www.opengis.net/def/crs/EPSG/0/4326">
                <gml:exterior>
                    <gml:LinearRing>
                        <gml:posList>
                            -27.0621757 153.3610112
                            -27.1990606 153.3658177
                            -27.3406573 153.421436
                            -27.3607835 153.4269292
                            -27.3315078 153.4434087
                            -27.2913403 153.4183848
                            -27.2039578 153.4189391
                            -27.0267166 153.4673476
                            -27.0621757 153.3610112
                        </gml:posList>
                    </gml:LinearRing>
                </gml:exterior>
            </gml:Polygon>"""^^go:gmlLiteral ;
```

```
geo:asKML """<Polygon>
        <outerBoundaryIs>
            <LinearRing>
                <coordinates>
                153.3610112,-27.0621757
                153.3658177,-27.1990606
                153.421436,-27.3406573
                153.4269292,-27.3607835
                153.4434087,-27.3315078
                153.4183848,-27.2913403
                153.4189391,-27.2039578
                153.4673476,-27.0267166
                153.3610112,-27.0621757
                </coordinates>
            </LinearRing>
        </outerBoundaryIs>
    </Polygon>"""^^go:kmlLiteral ;

geo:asGeoJSON """{
        "type": "Polygon",
        "coordinates": [[
            [153.3610112, -27.0621757],
            [153.3658177, -27.1990606],
            [153.421436, -27.3406573],
            [153.4269292, -27.3607835],
            [153.4434087, -27.3315078],
            [153.4183848, -27.2913403],
            [153.4189391, -27.2039578],
            [153.4673476, -27.0267166],
            [153.3610112, -27.0621757]
        ]]
    }"""^^geo:geoJSONLiteral ;

geo:asDGGS """<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
    R834603864 R834603866 R834603867 R834606021 R834606022 R834606024 R834606025
    R834606028 R834606052 R834606055 R834606160 R834606161 R834606162 R834606164
    R834606165 R834606167 R834606168 R834606200 R834606203 R834606206 R834606230
    R834606233 R834606236 R834606260 R834606263 R834606266 R834606401 R834606402
    R834606405 R834606408 R834606432 R834606471 R834606472 R834606474 R834606475
    R834606477 R834606478 R834606500 R834606503 R834606506 R834606530 R834606533
    R834606536 R834606560 R834606563 R834606566 R834606712 R834606715 R834606718
    R834606750 R834606751 R834606752 R834606753 R834606754 R834606755 R834606757
    R834606758 R834606781 R834606782 R834606784 R834606785 R834606788 R834606800
    R834606803 R834606806 R834606807 R834606830 R834606831 R834606833 R834606834
    R834606835 R834606836 R834606837 R834606838 R834606870 R834606873 R834606874
    R834606876 R834606877 R834630122 R834630125 R834630226 R834630230 R834630231
```

```
            R834630232 R834630234 R834630235 R834630237 R834630238 R834630240 R834630241
            R834630242 R834630243 R834630244 R834630245 R834630246 R834630247 R834630261
            R834630262 R834630264 R834630265 R834630268 R834630270 R834630271 R834630273
            R834630276 R834630502))"""^^geo:dggsLiteral ;
    ] ;
.
```

C.2 Example SPARQL Queries and 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 and the figure [img-illustration] for reference.

image::figures/03.png

```
@prefix geo: <http://www.opengis.net/ont/geosparql#> .
@prefix my: <http://example.org/ApplicationSchema#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix sf: <http://www.opengis.net/ont/sf#> .

my:PlaceOfInterest a rdfs:Class ;
    rdfs:subClassOf geo:Feature .

my:A a my:PlaceOfInterest ;
    my:hasExactGeometry my:AExactGeom ;
    my:hasPointGeometry my:APointGeom .

my:B a my:PlaceOfInterest ;
    my:hasExactGeometry my:BExactGeom ;
    my:hasPointGeometry my:BPointGeom .

my:C a my:PlaceOfInterest ;
    my:hasExactGeometry my:CExactGeom ;
    my:hasPointGeometry my:CPointGeom .

my:D a my:PlaceOfInterest ;
    my:hasExactGeometry my:DExactGeom ;
    my:hasPointGeometry my:DPointGeom .

my:E a my:PlaceOfInterest ;
    my:hasExactGeometry my:EExactGeom .

my:F a my:PlaceOfInterest ;
    my:hasExactGeometry my:FExactGeom .

my:hasExactGeometry a rdf:Property ;
    rdfs:subPropertyOf geo:hasDefaultGeometry,
        geo:hasGeometry .

my:hasPointGeometry a rdf:Property ;
    rdfs:subPropertyOf geo:hasGeometry .

my:AExactGeom a sf:Polygon ;
```

```
    geo:asWKT """<http://www.opengis.net/def/crs/OGC/1.3/CRS84>
                 Polygon((-83.6 34.1, -83.2 34.1, -83.2 34.5,
                 -83.6 34.5, -83.6 34.1))"""^^geo:wktLiteral.

my:APointGeom a sf:Point ;
    geo:asWKT """<http://www.opengis.net/def/crs/OGC/1.3/CRS84>
                 Point(-83.4 34.3)"""^^geo:wktLiteral.

my:BExactGeom a sf:Polygon ;
    geo:asWKT """<http://www.opengis.net/def/crs/OGC/1.3/CRS84>
                 Polygon((-83.6 34.1, -83.4 34.1, -83.4 34.3,
                 -83.6 34.3, -83.6 34.1))"""^^geo:wktLiteral.

my:BPointGeom a sf:Point ;
    geo:asWKT """<http://www.opengis.net/def/crs/OGC/1.3/CRS84>
                 Point(-83.5 34.2)"""^^geo:wktLiteral.

my:CExactGeom a sf:Polygon ;
    geo:asWKT """<http://www.opengis.net/def/crs/OGC/1.3/CRS84>
                 Polygon((-83.2 34.3, -83.0 34.3, -83.0 34.5,
                 -83.2 34.5, -83.2 34.3))"""^^geo:wktLiteral.

my:CPointGeom a sf:Point ;
    geo:asWKT """<http://www.opengis.net/def/crs/OGC/1.3/CRS84>
                 Point(-83.1 34.4)"""^^geo:wktLiteral.

my:DExactGeom a sf:Polygon ;
    geo:asWKT """<http://www.opengis.net/def/crs/OGC/1.3/CRS84>
                 Polygon((-83.3 34.0, -83.1 34.0, -83.1 34.2,
                 -83.3 34.2, -83.3 34.0))"""^^geo:wktLiteral.

my:DPointGeom a sf:Point ;
    geo:asWKT """<http://www.opengis.net/def/crs/OGC/1.3/CRS84>
                 Point(-83.2 34.1)"""^^geo:wktLiteral.

my:EExactGeom a sf:LineString ;
    geo:asWKT """<http://www.opengis.net/def/crs/OGC/1.3/CRS84>
                 LineString(-83.4 34.0, -83.3 34.3)"""^^geo:wktLiteral.

my:FExactGeom a sf:Point ;
    geo:asWKT """<http://www.opengis.net/def/crs/OGC/1.3/CRS84>
                 Point(-83.4 34.4)"""^^geo:wktLiteral.
```

C.2.2 Example Queries

This Section illustrates the use of GeoSPARQL functions through a series of example queries.

C.2.2.1 All features that a given feature contains

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 All features within bounding box

Find all features that are within a transient bounding box geometry, where spatial calculations are based on `my:hasPointGeometry`.

```
PREFIX my: <http://example.org/ApplicationSchema#>
PREFIX geo: <http://www.opengis.net/ont/geosparql#>
PREFIX geof: <http://www.opengis.net/def/function/geosparql/>

SELECT ?f
WHERE {
    ?f my:hasPointGeometry ?fGeom .
    ?fGeom geo:asWKT ?fWKT .
    FILTER (
        geof:sfWithin(
            ?fWKT,
            "<http://www.opengis.net/def/crs/OGC/1.3/CRS84>
            Polygon ((-83.4 34.0, -83.1 34.0,
                        -83.1 34.2, -83.4 34.2,
                        -83.4 34.0))"^^geo:wktLiteral
        )
    )
}
```

Result:

?f
my:D

C.2.2.3 All features touching the union of two features

Find all features that touch the union of feature `my:A` *and feature* `my:D`*, where computations are based on* `my:hasExactGeometry`.

```
PREFIX my: <http://example.org/ApplicationSchema#>
PREFIX geo: <http://www.opengis.net/ont/geosparql#>
PREFIX geof: <http://www.opengis.net/def/function/geosparql/>

SELECT ?f
WHERE {
    ?f my:hasExactGeometry ?fGeom .
    ?fGeom geo:asWKT ?fWKT .
    my:A my:hasExactGeometry ?aGeom .
    ?aGeom geo:asWKT ?aWKT .
    my:D my:hasExactGeometry ?dGeom .
    ?dGeom geo:asWKT ?dWKT .
    FILTER (
        geof:sfTouches(
            ?fWKT,
            geof:union(?aWKT, ?dWKT)
        )
    )
}
```

Result:

?f
my:C

C.2.2.4 Three closest features to a feature

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 Maximum and mimimum coordinates of a set of geometries

Find the maximum and minimum coordinates of a given set of geometries.

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

SELECT ?minX ?minY ?minZ ?maxX ?maxY ?maxZ
WHERE {
    BIND ("<http://www.opengis.net/def/crs/OGC/1.3/CRS84>
            Polygon Z((-83.4 34.0 0, -83.1 34.0 1,
                        -83.1 34.2 1, -83.4 34.2 1,
                        -83.4 34.0 0))"^^geo:wktLiteral) AS ?testgeom)
    BIND(geof:minX(?testgeom) AS ?minX)
    BIND(geof:maxX(?testgeom) AS ?maxX)
    BIND(geof:minY(?testgeom) AS ?minY)
    BIND(geof:maxY(?testgeom) AS ?maxY)
    BIND(geof:maxZ(?testgeom) AS ?maxZ)
    BIND(geof:minZ(?testgeom) AS ?minZ)
}
```

Result:

?minX	?minY	?minZ	?maxX	?maxY	?maxZ
-83.4	34.0	0	-83.1	34.2	1

C.2.3 Example Rule Application

This section illustrates the query transformation strategy for implementing GeoSPARQL rules.

C.2.3.1 All features or geometries overlapping with another feature

Find all features or geometries that overlap feature `my:A`.

Original Query:

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

SELECT ?f
WHERE { ?f geo:sfOverlaps my:A }
```

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

```
PREFIX my: <http://example.org/ApplicationSchema#>
PREFIX geo: <http://www.opengis.net/ont/geosparql#>
PREFIX geof: <http://www.opengis.net/def/function/geosparql/>

SELECT ?f
WHERE {
    { # check for asserted statement
        ?f geo:sfOverlaps my:A }
    UNION
    { # feature - feature
        ?f geo:hasDefaultGeometry ?fGeom .
        ?fGeom geo:asWKT ?fSerial .
        my:A geo:hasDefaultGeometry ?aGeom .
        ?aGeom geo:asWKT ?aSerial .
        FILTER (geof:sfOverlaps(?fSerial, ?aSerial))
    }
    UNION
    { # feature - geometry
        ?f geo:hasDefaultGeometry ?fGeom .
        ?fGeom geo:asWKT ?fSerial .
        my:A geo:asWKT ?aSerial .
        FILTER (geof:sfOverlaps(?fSerial, ?aSerial))
    }
    UNION
    { # geometry - feature
        ?f geo:asWKT ?fSerial .
        my:A geo:hasDefaultGeometry ?aGeom .
        ?aGeom geo:asWKT ?aSerial .
        FILTER (geof:sfOverlaps(?fSerial, ?aSerial))
    }
    UNION
    { # geometry - geometry
        ?f geo:asWKT ?fSerial .
        my:A geo:asWKT ?aSerial .
        FILTER (geof:sfOverlaps(?fSerial, ?aSerial))
    }
```

Result:

?f
my:D
my:DExactGeom
my:E
my:EExactGeom

C.2.4 Example Geometry Serialization Conversion Functions

For the geometry literal values in [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` and `geof:asDGGS`.

Annex D
(informative)

Usage of SHACL shapes

D.0. Overview

This Annex provides guidance on the usage of the SHACL shapes included with this version of GeoSPARQL.

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, 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 by using these Shapes and use the validation results to correct any mistakes.

D.1 Tools

The SHACL Shapes provided with GeoSPARQL, see [GeoSPARQL_Standard_structure], 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 D.1 — Alignment: GeoSPARQL SHACL Shapes

SHACL Shape ID	Severity	Test purpose	Requirements tested
Shape 1a	Violation	Each node with an incoming `hasGeometry`, or a specialization of it, should have at minimum one outgoing relation that is either `hasSerialization`, or a specialization of it.	8.7, Requirement 8
Shape 1b	Violation	Each node with an incoming `hasGeometry`, or a specialization of it, can have a maximum of one outgoing `asWKT` relation.	8.7 Requirement 21
Shape 1c	Violation	Each node with an incoming `hasGeometry`, or a specialization of it, can have a maximum of one outgoing `asGML` relation.	8.7 Requirement 26
Shape 1d	Violation	Each node with an incoming `hasGeometry`, or a specialization of it, can have a maximum of one outgoing `asGeoJSON` relation.	8.7 Requirement 31
Shape 1e	Violation	Each node with an incoming `hasGeometry`, or a specialization of it, can have a maximum of one outgoing `asKML` relation.	8.7 Requirement 36
Shape 2	Violation	Each node with one or more outgoing relations that are either `hasSerialization`, or a specialization of it, should have at least one incoming `hasGeometry` relation or a specialization of it.	8.7
Shape 3a-c	Violation	A node that has an incoming `hasGeometry` property, or specialization of it, cannot have an outgoing `hasGeometry` property, or a specialization of, it at the same time (a `Feature` cannot be a `Geometry` at the same time)	Requirement 8
Shape 4	Violation	The target of a `hasSerialization` property, or a specialization	8.7

Table D.1 *(continued)*

SHACL Shape ID	Severity	Test purpose	Requirements tested
		of, it should be an RDF literal	
Shape 5	Violation	The target of a `asWKT` property should be an RDF literal with datatype `wktLiteral`	Requirement 17
Shape 6	Violation	The target of a `asGML` property should be an RDF literal with datatype `gmlLiteral`	Requirement 23
Shape 7	Violation	The target of a `asGeoJSON` property should be an RDF literal with datatype `geoJSONLiteral`	Requirement 28
Shape 8	Violation	The target of a `asKML` property should be an RDF literal with datatype `kmlLiteral`	Requirement 33
Shape 9	Violation	A `Geometry` node should have a maximum of one outgoing `coordinateDimension` property	8.7
Shape 10	Violation	A `Geometry` node should have a maximum of one outgoing `dimension` property	8.7
Shape 11	Violation	A `Geometry` node should have a maximum of one outgoing `isEmpty` property	8.7
Shape 12	Violation	A `Geometry` node should have a maximum one outgoing `isSimple` property	8.7
Shape 13	Violation	A `Geometry` node should have maximum of one outgoing `hasSpatialDimension` property	8.7
Shape 14a	Violation	A `Geometry` node should have maximum of one outgoing `hasSpatialResolution` property	8.7
Shape 14b	Violation	A `Geometry` node should have maximum of one outgoing `hasSpatialAccuracy` property	8.7
Shape 14c	Violation	A `Geometry` node should have maximum of one outgoing `hasMetricSpatialAccuracy` property	8.7

Table D.1 *(continued)*

SHACL Shape ID	Severity	Test purpose	Requirements tested
Shape 14d	Violation	A Geometry node should have maximum of one outgoing hasMetricSpatialResolution property	8.7
Shape 15	Violation	The content of an RDF literal with an incoming asWKT relation must conform to a well-formed WKT string, as defined by its official specification (Simple Features Access)	Requirement 17
Shape 16	Violation	The content of an RDF literal with an incoming asWKT relation must conform to a well-formed WKT string, as defined by its official specification (Simple Features Access)	Requirement 23
Shape 17	Violation	The content of an RDF literal with an incoming asGeoJSON relation must conform to a well-formed GeoJSON geometry string, as defined by its official specification	Requirement 28
Shape 18	Violation	The content of an RDF literal with an incoming asKML relation must conform to a well-formed KML geometry XML string, as defined by its official specification	Requirement 33
Shape 20	Violation	If both geo:dimension and coordinateDimension properties are asserted, the value of dimension should be less than or equal to the value of geo:coordinateDimension	8.7
Shape 21a	Violation	An instance of FeatureCollection should have at least one outgoing rdfs:member relation	Requirement 6
Shape 21b	Violation	An instance of FeatureCollection should only have outgoing rdfs:member going to Feature instances	Requirement 6
Shape 22a	Violation	An instance of GeometryCollection should have at least one outgoing rdfs:member relation	Requirement 15

Table D.1 *(continued)*

SHACL Shape ID	Severity	Test purpose	Requirements tested
Shape 22b	Violation	An instance of `GeometryCollection` should only have outgoing rdfs:member relations to `Geometry` instances	Requirement 15
Shape 23a	Violation	An instance of `SpatialObjectCollection` should have at least one outgoing rdfs:member relation	Requirement 5
Shape 23b	Violation	An instance of `SpatialObjectCollection` should only have outgoing rdfs:member relations going to `SpatialObject` instances, or subclasses of them	Requirement 5

Annex E
(informative)

Alignments

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 E.1 — Alignment: Namespaces

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

E.1 ISA Programme Location Core Vocabulary (LOCN)

LOCN Source: https://www.w3.org/ns/locn

The LOCN specification provides notes on the use of GeoSPARQL literals (see https://www.w3.org/ns/locn#changes).

Table E.2 — Alignment: ISA Programme Location Core Vocabulary (LOCN)

From Element	Mapping relation	To Element	Notes
Feature	rdfs:subClassOf	dcterms:Location	LOCN states that dcterms:Location "represents any location, irrespective of size or other restriction". As such, it can be considered as a superclass of Feature.
locn:Address	rdfs:subClassOf	Feature	Although LOCN does not explicitly indicate spatial or geometry properties for locn:Address, this class can be considered as a specialized form of a Feature.
Geometry	rdfs:subClassOf	locn:Geometry	In LOCN, class locn:Geometry "[...] defines the notion of geometry at the conceptual level, and it shall be encoded by using different formats". More precisely, its instances can be either literals or individuals. The GeoSPARQL's class Geometry is more narrowly defined, as its instances can only be individuals, and not literals.
hasGeometry	rdfs:subPropertyOf	locn:geometry	In LOCN, the usage note to property locn:geometry states that "Depending on how a geometry is encoded, the range of this property may be one of the following: a literal [...], an instance of a geometry class [...], geocoded URIs [...]". The GeoSPARQL's property hasGeometry is more narrowly defined, as it can only be used with instances of Geometry, and not with literals.

E.2 WGS84 Geo Positioning: an RDF vocabulary (POS)

POS Source: http://www.w3.org/2003/01/geo/

Table E.3 — Alignment: WGS84 Geo Positioning Vocabulary (POS)

From Element	Mapping relation	To Element	Notes
SpatialObject	owl:equivalentClass	pos:SpatialThing	Both classes are unrestricted, essentially abstract classes
pos:Point	rdfs:subClassOf	Geometry	Via pos:Point rdfs:subClassOf pos:SpatialThing but since

Table E.3 *(continued)*

From Element	Mapping relation	To Element	Notes
			pos:Point usage notes indicates direct positioning, it is a form of geometry
pos:Point	owl:equivalentClass	sf:Point	
pos:lat_long	rdfs:subPropertyOf	hasSerialization	A special datatype is not indicated for use with this property by POS, unlike GeoSPARQL's hasSerialization object literals
pos:location	rdfs:subPropertyOf	hasGeometry	

E.3 W3C Activity Streams Vocabulary

AS Source: https://www.w3.org/TR/activitystreams-vocabulary/

Table E.4 — Alignment: W3C Activity Streams Vocabulary

From Element	Mapping relation	To Element	Notes
as:Place	owl:equivalentClass	Feature	AS places are only defined for point geometries
as:accuracy	rdfs:subPropertyOf	hasSpatialAccuracy	AS expresses the accuracy in percent
as:altitude			The altitude property can be expressed as a Z coordinate in GeoSPARQL-compatible literals
as:latitude	rdfs:subPropertyOf	hasSerialization	AS defines the range of this property as xsd:float
as:longitude	rdfs:subPropertyOf	hasSerialization	AS defines the range of this property as xsd:float

E.4 Geonames Ontology (GN)

Geonames source: http://www.geonames.org/ontology/documentation.html

Table E.5 — Alignment: Geonames Vocabulary (GN)

From Element	Mapping relation	To Element	Notes
gn:Feature	owl:equivalentClass	Feature	
gn:GeonamesFeature	rdfs:subClassOf	Feature	The GN class is defined as "A feature described in geonames database..."
Feature	rdfs:subClassOf	gn:Class	The GN class' definition reads "A class of features"
gn:locatedIn	owl:equivalentProperty	sfWithin	
gn:nearby	rdfs:subPropertyOf	sfDisjoint	A gn:nearby B means A is not within or touching B. The only close SF property is disjoint
gn:neighbour	owl:equivalentProperty	sfTouches	

E.5 NeoGeo Vocabulary

NeoGeo Source: http://geovocab.org/ / http://geovocab.org/doc/neogeo/

Table E.6 — Alignment: NeoGeo Vocabulary

From Element	Mapping relation	To Element	Notes
spatial:Feature	owl:equivalentClass	Feature	
spatial:C	rdfs:subPropertyOf	rcc8ec	Sub proerty not equivalent property since the NeoGeo property has more restrictive domain and range
spatial:DR	rdfs:subPropertyOf	rcc8dc	
spatial:EC	rdfs:subPropertyOf	rcc8ec	
spatial:EQ	rdfs:subPropertyOf	rcc8eq	
spatial:NTPP	rdfs:subPropertyOf	rcc8ntpp	
spatial:NTPPi	rdfs:subPropertyOf	rcc8ntppi	
spatial:O	rdfs:subPropertyOf	sfOverlaps	
spatial:P	rdfs:subPropertyOf	sfWithin	
spatial:PO	rdfs:subPropertyOf	rcc8po	
spatial:PP	rdfs:subPropertyOf	sfWithin	
spatial:PPi	rdfs:subPropertyOf	sfContains	
spatial:Pi	rdfs:subPropertyOf	sfContains	
spatial:TPP	rdfs:subPropertyOf	rcc8tpp	
spatial:TPPi	rdfs:subPropertyOf	rcc8tppi	
geom:Geometry	owl:equivalentClass	Geometry	
geom:BoundingBox	rdfs:subClassOf	Geometry	GeoSPARQL doesn't have a BoundingBox class but has a generic Geometry class that is the range of the hasBoundingBox property
geom: GeometryCollection	owl:equivalentClass	GeometryCollection	
geom:LineString	owl:equivalentClass	sf:LineString	
geom:LinearRing	owl:equivalentClass	sf:LinearRing	
geom:MultiLineString	owl:equivalentClass	sf:MultiLineString	
geom:MultiPoint	owl:equivalentClass	sf:MultiPoint	
geom:MultiPolygon	owl:equivalentClass	sf:MultiPolygon	
geom:Polygon	owl:equivalentClass	sf:Polygon	
geom:Point	owl:equivalentClass	sf:Point	
hasGeometry	rdfs:subPropertyOf	geom:geometry	hasGeometry has more restrictive domain

- The geom:bbox property relates a Geometry to another Geometry and is thus not equivalent to GeoSPARQL's Feature-to-Geometry hasBoundingBox.
  - An equivalent to bbox could be made using a Feature with a Geometry, indicated by hasGeometry and a second, specialised Bounding Box Geometry indicated with hasBoundingBox

E.6 Juso Ontology

Juso Source: http://rdfs.co/juso/

Juso contains mappings to GeoSPARQL but uses owl:sameAs which it should instead use owl:equivalentClass.

Table E.7 — Alignment: Juso Ontology

From Element	Mapping relation	To Element
juso:SpatialThing	owl:equivalentClass	SpatialObject
juso:Feature	owl:equivalentClass	Feature
juso:Geometry	owl:equivalentClass	Geometry
juso:Point	owl:equivalentClass	sf:Point
juso:geometry	owl:equivalentProperty	hasGeometry
juso:parent	rdfs:subPropertyOf	sfWithin
juso:political_division	rdfs:subPropertyOf	sfContains
juso:within	owl:equivalentProperty	sfWithin

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 `hasGeometry` to indicate that a `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 `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 `Feature` so that a spatio-temporal Feature may be indicated like this:

```
:flooded-area-x
    a geo:Feature ;
    geo:hasGeometry [
        a geo:Geometry ;
        geo:asWKT "POLYGON (((...)))"^^geo:wktLiteral ;
    ] ;
    time:hasTime [
        a time:ProperInterval ;
        time:hasBeginning [
            time:inXSDDate "..."^^xsd:date ;
        ] ;
        time:hasEnd [
            time:inXSDDate "..."^^xsd:date ;
        ] ;
    ] ;
.
```

In the above example, `:flooded-area-x` is a spatio-temporal Feature that has both a GeoSPARQL spatial projection — a `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 `Geometry` instances. This is allowed given `time:hasTime`'s open domain:

```
:flooded-area-x
    a geo:Feature ;
    geo:hasGeometry [
        a geo:Geometry ;
        geo:asWKT "POLYGON (((...)))"^^geo:wktLiteral ;
```

```
        time:hasTime [ ... ] ;
    ] ;
.
```

In contrast to the first example, `:flooded-area-x` is inferred to be a spatio-temporal Feature but since it is the Geometry of `:flooded-area-x` that has a temporality, it is possible to describe other Geometries of `:flooded-area-x` with other temporalities.

E.8 schema.org

schema.org Source: https://schema.org

Table E.8 — Alignment: schema.org

From Element	Mapping relation	To Element	Notes
Geometry	rdfs:subClassOf	sdo:GeoShape	A GeoShape can various literal geometry representation
sdo:GeospatialGeometry	owl:equivalentClass	SpatialObject	Since sdo: GeospatialGeometry is the domain of SimpleFeature-like properties and a superclass of GeoShape
sdo:GeoCoordinates	rdfs:subClassOf	Geometry	GoCoordinates uses direct lat, long, elevation etc properties to indicate position, not a while geometry serialization but it is nevertheless a form of a Geometry
sdo:geo	rdfs:subPropertyOf	hasGeometry	
sdo:geoCoveredBy	owl:equivalentProperty	ehCoveredBy	
sdo:geoCovers	owl:equivalentProperty	ehCovers	
sdo:geoCrosses	owl:equivalentProperty	sfCrosses	
sdo:geoDisjoint	owl:equivalentProperty	sfDisjoint	
sdo:geoEquals	owl:equivalentProperty	sfEquals	
sdo:geoIntersects	owl:equivalentProperty	sfIntersects	
sdo:geoOverlaps	owl:equivalentProperty	sfOverlaps	
sdo:geoTouches	owl:equivalentProperty	sfTouches	
sdo:geoWithin	owl:equivalentProperty	sfWithin	
sdo:geoMidpoint	owl:equivalentProperty	hasCentroid	
sdo:Landform	rdfs:subClassOf	Feature	

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 `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 `Feature`.

E.10 DCMI Metadata Terms (DCTERMS)

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

Table E.9 — Alignment: DCMI Metadata Terms (DCTERMS)

From Element	Mapping relation	To Element	Notes
`Feature`	`rdfs:subClassOf`	`dcterms:Location`	A Location is a "A spatial region or named place."
`hasGeometry`	`rdfs:subPropertyOf`	`dcterms:spatial`	`dcterms:spatial` indicates the "Spatial characteristics of the resource", thus it is a more general form of GeoSPARQL's `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 `Feature`, for which GeoSPARQL has no equivalent, but perhaps also for indicating a `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*[12) and the *DCMI Point Encoding Scheme*[13) could be indicated as GeoSPARQL geometry literals if a literal datatype were created for each. For example, the *DCMI Point Encoding Scheme* example of "The highest point in Australia" with the literal value `east=148.26218; north=-36.45746; elevation=2228; name=Mt. Kosciusko` might be encoded in GeoSPARQL like this:

```
:mt-kosciusko
    a geo:Feature ;
    geo:hasGeometry [
        a geo:Geometry ;
        geo:hasSerialization "east=148.26218; north=-36.45746; elevation=2228; name=Mt.
Kosciusko"^^ex:dcmiPoint ;
    ] ;
.
```

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` and `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`).

12) https://www.dublincore.org/specifications/dublin-core/dcmi-box/

13) https://www.dublincore.org/specifications/dublin-core/dcmi-point/

Table E.10 — Alignment: The Provenance Ontology (PROV)

From Element	Mapping relation	To Element	Notes
SpatialObjectCollection	rdfs:subClassOf	prov:Collection	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 (SpatialObject)
SpatialObject	rdfs:subClassOf	prov:Entity	All SpatialObjects fit within PROV's Entity's definition: "An entity is a physical, digital, conceptual, or other kind of thing with some fixed aspects; entities may be real or imaginary."
Feature	rdfs:subClassOf	prov:Location	A Location "...can be an identifiable geographic place (ISO 19112), but it can also be a non-geographic place such as a directory, row, or column" so seem to be wider in scope than GeoSPARQL's Feature although a Feature could indeed be something such as a "directory, row, or column"

— The PROV property prov:atLocation indicates prov:Location instances, which may be Feature instances, but GeoSPARQL has no property to indicate a 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:

```
:bounding-box-y prov:wasDerivedFrom :polygon-x .
```

E.12 WikiData

Table E.11 — Alignment: WikiData

From Element	Mapping relation	To Element	Notes
wdt:P625	owl:equivalentProperty	asWKT	The Wikidata description of this property labeled "coordinate location" note that "For Earth, please note that only WGS84 coordinating system is supported at the moment" but that is a system limit, not an ontological one
wdt:P3896	owl:propertyChainAxiom	(geo:hasGeometry geo:asGeoJSON)	This Wikidata property labeled "geoshape" indicated GeoJSON geometry literal content for a Feature, but it allows information other than just Geometry in the GeoJSON

Table E.11 *(continued)*

From Element	Mapping relation	To Element	Notes
			whereas GeoSPARQL does not.
wdt:P3096	owl:propertyChainAxiom	(geo:hasGeometry geo:asKML)	This Wikidata property labeled “KML File” links to a KML file which is related to the respective instance. This may not be the same representation as in GeoSPARQL, as GeoSPARQL KML literals only encode the geometry part of a KML.
wd:Q82794	rdfs:subClassOf	Feature	The Wikidata class is labeled “geographic region” and thus is a subclass of the more general Feature. There are likely many other classes in Wikidata that could be interpreted as subclasses of Feature
wd:Q618123	owl:equivalentClass	Feature	The Wikidata class is labeled “geographical feature” and thus corresponds to Feature.
wd:Q25404640	owl:equivalentClass	SpatialObject	The Wikidata class is labeled “spatial object” and thus corresponds to SpatialObject.
wdt:P150	rdfs:subPropertyOf	sfContains	The Wikidata property is labeled “contains administrative territorial entity” but also alternatively labeled “contains”, “has districts” and others. There are likely many other specialized forms of sfContains and sfWithin in Wikidata
sfWithin	rdfs:subPropertyOf	wdt:P361	The Wikidata property is labeled “part of” 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 “has part”
sfContains	rdfs:subPropertyOf	wd:Q65964571	The property labeled “has part” is the inverse of wdt:P361 (see above)
wdt:P131	rdfs:subPropertyOf	sfContains	The Wikidata property is labeled “located in the administrative territorial entity” and is essentially

Table E.11 *(continued)*

From Element	Mapping relation	To Element	Notes
			the inverse of wdt:P150 (described above)
wdt:P706	rdfs:subPropertyOf	sfWithin	The Wikidata property is labeled “located in/on physical feature” and is indicated for use with a “(geo)physical feature” and not to be used for administrative features where wdt:P131 (see above) should be
wdt:P4688	rdfs:subClassOf	Feature	The Wikidata class is labeled “geomorphological unit” and is one of many Wikidata feature classes that could be expressed as a subclass of Feature. More specialized geological unit examples are wd:Q5107 “continent” and wdt:P4552 “mountain range”.
wdt:P2046	owl:equivalentProperty	hasArea	The Wikidata property is labeled “area”. It indicates a microformat — NUMBER + SPACE + ALLOWED_UNIT_LABEL — with a fixed set of ALLOWED_UNIT_LABELs to present values and units of measure.

E.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 Feature and is linked to either a Geometry or a latitude longitude representation. Hence, every linked geodata class can be considered a Feature in the sense of GeoSPARQL.

Table E.12 — Alignment: LinkedGeoData

From Element	Mapping relation	To Element	Notes
Any LGD Class	rdfs:subClassOf	Feature	Any class defined in the LinkedGeoData ontology is a subclass of Feature

E.13.2 OpenStreetMap RDF (Sophox)

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

Table E.13 — Alignment: OpenStreetMap RDF (Sophox)

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

E.13.3 Routable Tiles Ontology

https://github.com/openplannerteam/routable-tiles-ontology

Table E.14 — Alignment: Routable Tiles Ontology

From Element	Mapping relation	To Element	Notes
osm:Element	owl:equivalentClass	Geometry	The class osm:Element is equivalent to a Geometry
osm:Node	owl:equivalentClass	sf:Point	The class osm:Node is equivalent to a sf:Point
osm:Way	owl:equivalentClass	sf:LineString	The class osm:Way is equivalent to a sf: LineString

Table E.14 *(continued)*

From Element	Mapping relation	To Element	Notes
osm:Relation	owl:equivalentClass	sf:GeometryCollection	The class osm:Relation is equivalent to a sf: GeometryCollection

E.14 Ordnance Survey UK Spatial Ontology

http://www.ordnancesurvey.co.uk/legacy/ontologies/spatialrelations.owl and 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 E.15 — Alignment: Ordnance Survey UK Spatial Ontology

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

— spatialuk:easting describes a latitude coordinate east of the national UK grid and GeoSPARQL does not contain modelling of individual 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

CRMGeo Source: https://www.cidoc-crm.org/crmgeo/sites/default/files/CRMgeo1_2.pdf

Table E.16 — Alignment: CIDOC CRM Geo

From Element	Mapping relation	To Element	Notes
cidoc:SP1_ PhenomenalSpaceTimeVolume	rdfs:subClassOf	Feature	The CIDOC CRMgeo class SP1_ PhenomenalSpaceTimeVolume is a subclass of geo:Feature

Table E.16 *(continued)*

From Element	Mapping relation	To Element	Notes
			as described in the CRMgeo 1.2 specification document.
cidoc:SP2_PhenomenalPlace	rdfs:subClassOf	Feature	The CIDOC CRMgeo class SP2_PhenomenalPlace is a subclass of Feature as described in the CRMgeo 1.2 specification document.
cidoc:SP5_GeometricPlaceExpression	rdfs:subClassOf	Geometry	The CIDOC CRMgeo class SP5_GeometricPlaceExpression is a subclass of Geometry as described in the CRMgeo 1.2 specification document.
cidoc:SP6_DeclarativePlace	rdfs:subClassOf	Geometry	The CIDOC CRMgeo class SP6_DeclarativePlace is a subclass of geo:Geometry as described in the CRMgeo 1.2 specification document.
cidoc:SP7_DelcarativePlace	rdfs:subClassOf	Geometry	The CIDOC CRMgeo class SP7_DeclarativePlace is a subclass of geo:Geometry as described in the CRMgeo 1.2 specification document.
cidoc:SP10_DeclarativeTimeSpan	rdfs:subClassOf	Geometry	The CIDOC CRMgeo class SP10_DeclarativeTimeSpan is a subclass of geo:Geometry as described in the CRMgeo 1.2 specification document.
cidoc:SP14_TimeExpression	rdfs:subClassOf	Geometry	The CIDOC CRMgeo class SP14_TimeExpression is a subclass of geo:Geometry as described in the CRMgeo 1.2 specification document.
cidoc:SP15_Geometry	rdfs:subClassOf	Geometry	The CIDOC CRMgeo class SP15_Geometry is a subclass of geo:Geometry as described in the CRMgeo 1.2 specification document.

E.16 Basic Formal Ontology (BFO)

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

Table E.17 — Alignment: Basic Formal Ontology (BFO)

From Element	Mapping relation	To Element	Notes
SpatialObject	rdfs:subClassOf	obo:BFO_0000004 “independent continuant”	BFO’s “independent continuant” is the superclass of “material entity” and “immaterial entity” which are mapped to Feature and Geometry respectively, so at least some independent continuants must be Spatial Objects
Geometry	rdfs:subClassOf	obo:BFO_0000006 “spatial region”	BFO’s “spatial region” class is described as a “spatial projection of a portion of spacetime” so Geometry appears

Table E.17 *(continued)*

From Element	Mapping relation	To Element	Notes
			to be a subclass of this as it's "A coherent set of direct positions in space"
Geometry	rdfs:subClassOf	obo:IAO_0000030 "information content entity"	BFO's "information content entity" class is described as "an entity that represents information about some other entity", so Geometry appears to be subclass of this as well as "spatial region" since in GeoSPARQL, Geometry gives the details of the spatial projection of a Feature.
obo:BFO_0000040 "material entity"	rdfs:subClassOf	Feature	A BFO "material entity" is something that "has some portion of matter as continuant part" and some Features are such, however Features may be imaginary too
obo:BFO_0000029 "site"	rdfs:subClassOf	Feature	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
hasGeometry	rdfs:subPropertyOf	obo:BFO_0000211 "occupies spatial region at all times"	The BFO property links a thing that is not a spatial region to a spatial region, so it can be used as hasGeometry is used when the thing is taken to be a Feature and the spatial region a Geometry. No GeoSPARQL temporality indicators mean mappings are eternal.
hasGeometry	rdfs:subPropertyOf	obo:BFO_0000210 "occupies spatial region at some time"	A transitive mapping from the mapping above. Temporal qualification can be used with GeoSPARQL, see the OWL TIME alignment.
sfWithin	rdfs:subPropertyOf	obo:BFO_0000082 "located in at all times"	The BFO property "located in at all times" is a super property of sfWithin when the thing located in the spatial region are defined to both be instances of Feature. Since GeoSPARQL natively supplies no temporal qualifiers, pure GeoSPARQL assertions are assumed to be eternal: "...at all times"
sfWithin	rdfs:subPropertyOf	obo:BFO_0000171 "located in at some time"	A transitive mapping from the mapping above. Temporal qualification can be used with GeoSPARQL, see the OWL TIME alignment.
obo:BFO_0000066 "occurs in"	rdfs:range	SpatialObject	The BFO property relates a temporal activity to a spatial region but since GeoSPARQL has no notion of events, no mapping to this property can be made. However, BFO indicates this property should be used with a BFO "spatial region" (Geometry) range value but from GeoSPARQL's point of view, it could also be used with a Feature where the "in" would be taken to be within the feature's geometry, so the superclass of feature and geometry is given as the range
obo:BFO_0000216 "spatially projects onto at some time"	rdfs:range	SpatialObject	The reasoning is the same as for "occurs in"

— BFO distinguishes between *continuants* and *occurrants*, which *spatial region* and *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 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 F
(informative)

CQL / GeoSPARQL Mapping

F.1 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.2 Accessing spatial Features in a SPARQL endpoint

Spatial *Features* accessed via SPARQL endpoints SPARQLPROT are, as defined in the GeoSPARQL standard, instances of the OWL class `Feature` or of subclasses of it. They may have one or more `hasGeometry` properties indicating `Geometry` instances and other properties related to the Feature. They may also be grouped into `FeatureCollection` instances where `FeatureCollection` is a new class in GeoSPARQL 1.1, specifically for the description of collections of `Feature` instances.

The following example SPARQL query retrieves all Features within the Feature Collection with the IRI `ex:x` within a given SPARQL endpoint.

```
SELECT ?fcollection ?item ?rel ?val ?geom
WHERE {
  ex:x rdfs:member ?item .
  ?item rdfs:subClassOf* geo:Feature .
}
```

GeoSPARQL's `FeatureCollection` definition requires that `Feature` instances are to be linked to the Collection by use of the `rdf:member` property. No inverse property is defined.

NOTE Some CQL-implementing systems, such as OGC API, have fixed notions of Feature Collections and require that Features be members of exactly one Feature Collection. There is no such restriction in GeoSPARQL: Features may be members of one or more Feature Collections.

An extension to the above can retrieve any Geometry serializations for the Features within Feature Collection `ex:x`:

```
SELECT ?fcollection ?item ?rel ?val ?geom
WHERE {
  ex:x rdfs:member ?item .
  ?item rdfs:subClassOf* geo:Feature .

  OPTIONAL {
    ?item geo:hasGeometry/geo:hasSerialization ?geom
  }
}
```

Some additional concerns for GeoSPARQL / CQL or OGC API Features Feature Collections mappings are:

— APIs may need more information about the `FeatureCollection` instance for correct handling, in particular, an identifier and perhaps a label. If the back-end data store also contains information for the `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.3 Mappings from CQL2 statements to GeoSPARQL queries

This section presents lists of equivalences between Common Query Language (CQL2) CQLDEF statements and GeoSPARQL statements.

F.3.1 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 F.1 — CQL To GeoSPARQL Mappings: Query Parameters

Query Parameter	Example	SPARQL Expression	Example	Comment
limit	limit=5	LIMIT	LIMIT 5	
offset	offset=10	OFFSET	OFFSET 10	
bbox	`bbox=160.6,-55.95,-170,-25.89`	`FILTER(geo:sfIntersects())`	`FILTER(geo:sfIntersects(?geom, "POLYGON( (160.6 -55.95, 160.6 -25.89, -170 -25.89, -170 -55.95, 160.6 -55.95))"^^geo:wktLiteral))`	WKT does not define a type boundingbox, therefore a bbox is converted to a Polygon
datetime	`datetime= 2018-02-12T23%3A20%3A52Z`	-	-	GeoSPARQL doesn't detail temporal aspects of data. Filtering data using RDF temporal properties may be achieved using basic SPARQL queries and also OWL TIME TIME

F.3.2 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 F.2 — CQL To GeoSPARQL Mappings: Literal Values

CQL2 literal	Examples	(Geo)SPARQL literal	Examples
String	"This is a string"	`xsd:string`	"This is a string"^^xsd:string
Number	-100 3.14159	`xsd:int`, `xsd:integer`, `xsd:double`	"-100"^^xsd:integer "3.14159"^^xsd:double
Boolean	true false	`xsd:boolean`	"true"^^xsd:boolean "false"^^xsd:boolean
Spatial Geometry (WKT)	POINT(1 1)	`wktLiteral`	"POINT(1 1)"^^geo:wktLiteral

Table F.2 *(continued)*

CQL2 literal	Examples	(Geo)SPARQL literal	Examples
Spatial Geometry (JSON)	{"type": "Point", "coordinates":[1,1]}	`geoJSONLiteral`	"{"type": "Point", "coordinates":[1,1]}"^^geo: geoJSONLiteral
Temporal Literal	1969-07-20 1969-07-20T20:17:40Z	`xsd:date`, `xsd:dateTime`, `xsd:dateTimeStamp`	"1969-07-20"^^xsd: date "1969-07-20T20:17:40Z"^^xsd: dateTime

F.3.3 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 F.3 — CQL To GeoSPARQL Mappings: Property references

Property Reference	Triple pattern
name="OGC"	`?item my:name "OGC"^^xsd:string`
number=5	`?item my:number "5"^^xsd:integer`
number>5	`?item my:number ?number . FILTER(?number>5)`

F.3.4 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 F.4 — CQL To GeoSPARQL Mappings: Comparison Predicates

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

There is no direct GeoSPARQL equivalent to a CRS-based CQL filter, however certain GeoSPARQL geometry literals have explicity CRS/SRS information that may be filtered using SPARQL `REGEX` operators.

F.3.5 Spatial Operators

GeoSPARQL includes equivalents of many CQL2 filter functions as can be seen in the table below.

Table F.5 — CQL To GeoSPARQL Mappings: Spatial Operators

CQL2 Filter Expression	GeoSPARQL Filter Function
CONTAINS(geometry1,geometry2)	FILTER(geof:sfContains(?geometry1,?geometry2))
CROSSES(geometry1,geometry2)	FILTER(geof:sfCrosses(?geometry1,?geometry2))
DISJOINT(geometry1,geometry2)	FILTER(geof:sfDisjoint(?geometry1,?geometry2))
EQUALS(geometry1,geometry2)	FILTER(geof:sfEquals(?geometry1,?geometry2))
INTERSECTS(geometry1,geometry2)	FILTER(geof:sfIntersects(?geometry1,?geometry2))
OVERLAPS(geometry1,geometry2)	FILTER(geof:sfOverlaps(?geometry1,?geometry2))
TOUCHES(geometry1,geometry2)	FILTER(geof:sfTouches(?geometry1,?geometry2))
WITHIN(geometry1,geometry2)	FILTER(geof:sfWithin(?geometry1,?geometry2))

F.3.6 Temporal Operators

Temporal operators are not part of the GeoSPARQL standard.

Table F.6 — CQL To GeoSPARQL Mappings: Temporal Operators

CQL2 Filter Expression	GeoSPARQL Filter Function
beginTime AFTER 1969-07-16T13:32:00Z	N/A
beginTime BEFORE 1969-07-16T13:32:00Z	N/A
beginTime BEGINS 1969-07-16T13:32:00Z	N/A
beginTime BEGUNBY 1969-07-16T13:32:00Z	N/A
beginTime DURING 1969-07-16T13:32:00Z	N/A
beginTime ENDEDBY 1969-07-16T13:32:00Z	N/A
beginTime ENDS 1969-07-16T13:32:00Z	N/A
beginTime MEETS 1969-07-16T13:32:00Z	N/A
beginTime METBY 1969-07-16T13:32:00Z	N/A
beginTime OVERLAPPEDBY 1969-07-16T13:32:00Z	N/A
beginTime TCONTAINS 1969-07-16T13:32:00Z	N/A
beginTime TEQUALS 1969-07-16T13:32:00Z	N/A
beginTime TOVERLAPS 1969-07-16T13:32:00Z	N/A

As noted above in Section F.3.1, 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.4 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 F.7 — CQL To GeoSPARQL Mappings: Simple Features for SQL

Simple Features for SQL	GeoSPARQL Equivalent	Since GeoSPARQL	Related Property Available	Since GeoSPARQL
2.1.1.1 Basic Methods on Geometry				
Dimension(): Double	`geof:dimension`	-	`geo:dimension`	1.0
GeometryType(): Integer	Class of geometry instance	1.0	N/A	-
SRID(): Integer	`geof:getSRID`	1.0	N/A	-
Envelope(): Geometry	`geof:envelope`	1.0	`geo:hasBoundingBox`	1.1

Table F.7 *(continued)*

Simple Features for SQL	GeoSPARQL Equivalent	Since GeoSPARQL	Related Property Available	Since GeoSPARQL
AsText(): String	`geof:asWKT`	1.1	`geo:asWKT`	1.0
AsBinary(): Binary	N/A	-	N/A	-
IsEmpty(): Integer	`geof:isEmpty`	-	`geo:isEmpty`	1.0
IsSimple(): Integer	`geof:isEmpty`	-	`geo:isSimple`	1.0
Boundary(): Geometry	`geof:boundary`	1.0	N/A	-
2.1.1.2 Spatial Relations				
Equals(anotherGeometry: Geometry): Integer	`geof:sfEquals`	1.0	`geo:sfEquals`	1.0
Disjoint(anotherGeometry: Geometry): Integer	`geof:sfDisjoint`	1.0	`geo:sfDisjoint`	1.0
Intersects(anotherGeometry: Geometry): Integer	`geof:sfIntersects`	1.0	`geo:sfIntersects`	1.0
Touches(anotherGeometry: Geometry): Integer	`geof:sfTouches`	1.0	`geo:sfTouches`	1.0
Crosses(anotherGeometry: Geometry): Integer	`geof:sfCrosses`	1.0	`geo:sfCrosses`	1.0
Within(anotherGeometry: Geometry): Integer	`geof:sfWithin`	1.0	`geo:sfWithin`	1.0
Contains(anotherGeometry: Geometry): Integer	`geof:sfContains`	1.0	`geo:sfContains`	1.0
Overlaps(anotherGeometry: Geometry): Integer	`geof:sfOverlaps`	1.0	`geo:sfOverlaps`	1.0
Relate(anotherGeometry: Geometry, IntersectionPatternMatrix: String): Integer	`geof:relate`	1.0	N/A	-
2.1.1.3 Spatial Analysis				
Buffer(distance: Double): Geometry	`geof:buffer`	1.0	N/A	-
ConvexHull(): Geometry	`geof:convexHull`	1.0	N/A	-
Intersection(anotherGeometry: Geometry): Geometry	`geof:intersection`	1.0	N/A	-
Union(anotherGeometry: Geometry): Geometry	`geof:union`	1.0	N/A	-
Difference(anotherGeometry: Geometry): Geometry	`geof:difference`	1.0	N/A	-
SymDifference(anotherGeometry: Geometry): Geometry	`geof: symDifference`	1.0	N/A	-
2.1.2.1 GeometryCollection				
NumGeometries(): Integer	`geof: numGeometries`	-	N/A	-
GeometryN(N: Integer): Geometry	`geof:geometryN`	-	N/A	-
2.1.3.1 Point				
X(): Double	N/A	-	N/A	-
Y(): Double	N/A	-	N/A	-
Z(): Double (not in the SQL spec, but a logical extension)	N/A	-	N/A	-

Table F.7 *(continued)*

Simple Features for SQL	GeoSPARQL Equivalent	Since GeoSPARQL	Related Property Available	Since GeoSPARQL
M(): Double (not in the SQL spec, but a logical extension)	N/A	-	N/A	-
2.1.5.1 Curve				
Length(): Double	`geof:length`	-	`geo:hasLength`	1.1
StartPoint(): Point	N/A	-	N/A	-
EndPoint(): Point	N/A	-	N/A	-
IsClosed(): Integer	N/A	-	N/A	-
IsRing(): Integer	N/A	-	N/A	-
2.1.6.1 LineString				
NumPoints(): Integer	N/A	-	N/A	-
PointN(N: Integer): Point	N/A	-	N/A	-
2.1.7.1 MultiCurve				
IsClosed(): Integer	N/A	-	N/A	-
Length(): Double	`geof:length`	-	`geo:hasLength`	1.1
2.1.9.1 Surface				
Area(): Double	`geof:area`	-	`geo:hasArea`	1.1
Centroid(): Point	`geof:centroid`	1.1	`geo:hasCentroid`	1.1
PointOnSurface(): Point	N/A	-	N/A	-
2.1.10.1 Polygon				
ExteriorRing(): LineString	N/A	-	N/A	-
NumInteriorRing(): Integer	N/A	-	N/A	-
InteriorRingN(N: Integer): LineString	N/A	-	N/A	-
2.1.11.1 MultiSurface				
Area(): Double	`geof:area`	-	`geo:hasArea`	1.1
Centroid(): Point	`geof:centroid`	1.1	`geo:hasCentroid`	1.1
PointOnSurface(): Point	N/A	-	N/A	-

Annex G
(informative)

Revision History

This revision history covers all versions of GeoSPARQL, from the OGC's 1.0, to 1.1 to this ISO 1.2.

Table G.1 — Revision History

Date	Release	Author	Paragraph modified	Description
27 Oct. 2009	Draft	Matthew Perry	Clause 6	Technical Draft
11 Nov. 2009	Draft	John R. Herring	All	Creation
06 Jan. 2010	Draft	John R. Herring	All	Comment responses
30 March 2010	Draft	Matthew Perry	All	Comment responses
26 Oct. 2010	Draft	Matthew Perry	All	Revision based on working group discussion
28 Jan. 2011	Draft	Matthew Perry	All	Revision based on working group discussion
18 April 2011	Draft	Matthew Perry	All	Restructure with multiple conformance classes
02 May 2011	Draft	Matthew Perry	Clause 6 and Clause 8	Move Geometry Class from core to geometryExtension
05 May 2011	Draft	Matthew Perry	All	Update URIs
13 Jan. 2012	Draft	Matthew Perry	All	Revision based on Public RFC
16 April 2012	Draft	Matthew Perry	All	Revision based on adoption vote comments
19 July 2012	1.0	Matthew Perry	All	Revision of URIs based on OGC Naming Authority recommendations
09 Oct. 2020	1.1 Draft	Joseph Abhayaratna	All	Establishment of the 1.1 Specification
10 Oct. 2020 to 02 June. 2022	1.1 Draft	GeoSPARQL 1.1 SWG	All	Addition of GeoSPARQL 1.1 elements
23 Oct. 2022	1.1 For Public Comment	Carl Reed, Joseph Abhayaratna	All	Final review prior to public comment
13 Sept. 2025	1.2 Draft	Nicholas Car	All	1.2 modifications to 1.1 for ISO styling — no normative content changes

Bibliography

[1] AUSPIX, *AusPIX: An Australian Government implementation of the rHEALPix DGGS in Python*. Geoscience Australia. 2020. Available from: https://github.com/GeoscienceAustralia/AusPIX_DGGS

[2] CATEG, EGENHOFER, M. and J. HERRING. Categorizing binary topological relations between regions, lines and points in geographic databases, the 9-intersection: Formalism and its Use for Naturallanguage Spatial Predicates. *Santa Barbara CA National Center for Geographic Information and Analysis Technical Report*. 1990, vol. 94, p. 1

[3] CHARTER, OPEN GEOSPATIAL CONSORTIUM. *OGC GeoSPARQL SWG Charter*. 2020. Available from: https://github.com/opengeospatial/ogc-geosparql/blob/master/charter/swg_charter.pdf

[4] CQLDEF, OPEN GEOSPATIAL CONSORTIUM. *OGC API — Features — Part 3: Filtering and the Common Query Language (CQL2)*. 2021a. Available from: https://docs.ogc.org/DRAFTS/19-079r1.html

[5] DOUGLASPEUCKER, DOUGLAS, D. H. and T. K. PEUCKER. ALGORITHMS FOR THE REDUCTION OF THE NUMBER OF POINTS REQUIRED TO REPRESENT A DIGITIZED LINE OR ITS CARICATURE. *Cartographica: The International Journal for Geographic Information and Geovisualization*. 1973, vol. 10, p. 112. Available from: https://onlinelibrary.wiley.com/doi/abs/10.1002/9780470669488.ch2

[6] GEOJSON, BUTLER, H., M. DALY, A. DOYLE, S. GILLIES, T. SCHAUB and S. HAGEN. *The GeoJSON Format*. RFC Editor. Available from: https://www.rfc-editor.org/info/rfc7946

[7] IETF5234, CROCKER, D. and P. OVERELL. *Augmented BNF for Syntax Specifications: ABNF*. RFC Editor. Available from: https://www.rfc-editor.org/info/rfc5234

[8] ISO13249, INTERNATIONAL ORGANIZATION FOR STANDARDIZATION. *Information technology — Database languages — SQL multimedia and application packages — Part 3: Spatial*. Geneva, CH: 2000a. vol. 2000. Available from: https://www.iso.org/standard/31369.html

[9] ISO19105, INTERNATIONAL ORGANIZATION FOR STANDARDIZATION. *Geographic information — Conformance and testing*. Geneva, CH: 2000b. vol. 2000. Available from: https://www.iso.org/standard/26010.html

[10] ISO19107, INTERNATIONAL ORGANIZATION FOR STANDARDIZATION. *Geographic information — Spatial schema*. Geneva, CH: 2018. vol. 2019. Available from: https://www.iso.org/standard/66175.html

[11] ISO19109, INTERNATIONAL ORGANIZATION FOR STANDARDIZATION. *Geographic information — Rules for application schemas*. Geneva, CH: 2005b. vol. 2000. Available from: https://www.iso.org/standard/39891.html

[12] JSON-LD, KELLOGG, G., D. LONGLEY and P. CHAMPIN. *JSON-LD 1.1*. 2020. Available from: https://www.w3.org/TR/2020/REC-json-ld11-20200716/

[13] OGCSFACA, OPEN GEOSPATIAL CONSORTIUM. *OpenGIS Implementation Specification for Geographic information — Simple feature access — Part 1: Common architecture*. 2011. Available from: https://portal.ogc.org/files/?artifact_id=25355

[14] PROF, CAR, N. *The Profiles Vocabulary*. 2019. Available from: https://www.w3.org/TR/2019/NOTE-dx-prof-20191218/

[15] QUAL, COHN, A. G., B. BRANDON, J. GOODAY and N. MARK GOTTS. A small set of formal topological relationships suitable for end-user interaction. Berlin, Heidelberg: GeoInformatica. 1997. vol. 1 p. 275. Available from: https://doi.org/10.1023/A:1009712514511

[16] RDFSEM, PATEL-SCHNEIDER, P. and P. HAYES. *RDF 1.1 Semantics*. 2014. Available from: https://www.w3.org/TR/2014/REC-rdf11-mt-20140225/

[17] RDFXML, GANDON, F. and G. SCHREIBER. *RDF 1.1 XML Syntax*. 2014. Available from: https://www.w3.org/TR/2014/REC-rdf-syntax-grammar-20140225/

[18] RIF, BOLEY, H. and M. KIFER. *RIF Overview (Second Edition)*. 2013. Available from: https://www.w3.org/TR/2013/NOTE-rif-overview-20130205/

[19] SHACL, KNUBLAUCH, H. and D. KONTOKOSTAS. *Shapes Constraint Language (SHACL)*. 2017. Available from: https://www.w3.org/TR/2017/REC-shacl-20170720/

[20] SKOS, MILES, A. and S. BECHHOFER. *SKOS Simple Knowledge Organization System Reference*. 2009. Available from: https://www.w3.org/TR/2009/REC-skos-reference-20090818/

[21] SPARQLSERVDESC, WILLIAMS, G. *SPARQL 1.1 Service Description*. 2013. Available from: https://www.w3.org/TR/2013/REC-sparql11-service-description-20130321/

[22] TIME, COX, S. and C. LITTLE. *Time Ontology in OWL*. 2022. Available from: https://www.w3.org/TR/2022/CRD-owl-time-20221115/

[23] TURTLE, CAROTHERS, G. and E. PRUD'HOMMEAUX. *RDF 1.1 Turtle*. 2014. Available from: https://www.w3.org/TR/2014/REC-turtle-20140225/

[24] WFS, INTERNATIONAL ORGANIZATION FOR STANDARDIZATION. *Geographic information — Spatial schema*. Geneva, CH: 2010. vol. 2010. Available from: https://www.iso.org/standard/42136.html

[25] XML, BRAY, T., F. YERGEAU, M. SPERBERG-MCQUEEN, J. PAOLI and E. MALER. *Extensible Markup Language (XML) 1.0 (Fifth Edition)*. 2008. Available from: https://www.w3.org/TR/2008/REC-xml-20081126/

[26] XMLNS, LAYMAN, A., H. THOMPSON, D. HOLLANDER, T. BRAY and R. TOBIN. *Namespaces in XML 1.0 (Third Edition)*. 2009. Available from: https://www.w3.org/TR/2009/REC-xml-names-20091208/

[27] XSD1, THOMPSON, H., D. BEECH, N. MENDELSOHN and M. MALONEY. *XML Schema Part 1: Structures Second Edition*. 2004. Available from: https://www.w3.org/TR/2004/REC-xmlschema-1-20041028/

[28] XSD2, MALHOTRA, A. and P. V. BIRON. *XML Schema Part 2: Datatypes Second Edition*. 2004. Available from: https://www.w3.org/TR/2004/REC-xmlschema-2-20041028/

[29] OGC12-007r2, OPEN GEOSPATIAL CONSORTIUM. *OGC KML 2.3*. 2013. Available from: http://www.opengis.net/doc/IS/kml/2.3

[30] ISO10303-21, INTERNATIONAL ORGANIZATION FOR STANDARDIZATION. *Industrial automation systems and integration — Product data representation and exchangePart 21: Implementation methods: Clear text encoding of the exchange structure*. Geneva, CH: 2000c. vol. 2016. Available from: https://www.iso.org/standard/33713.html

[31] OGC07-036, OPEN GEOSPATIAL CONSORTIUM. *OpenGIS Geography Markup Language (GML) Encoding Standard*. 2007b. Available from: https://portal.ogc.org/files/?artifact_id=20509

[32] OGC20-040r3, OPEN GEOSPATIAL CONSORTIUM. *Abstract Standard Topic 21 — Discrete Global Grid Systems — Part 1 Core Reference system and Operations and Equal Area Earth Reference System*. 2021b. Available from: https://docs.ogc.org/as/20-040r3/20-040r3.html

[33] OGC17-069r3, OPEN GEOSPATIAL CONSORTIUM. *OGC API — Features — Part 1: Core*. 2019. Available from: http://www.opengis.net/doc/IS/ogcapi-features-1/1.0

[34] EGENHOFER, M. J. A Formal Definition of Binary Topological Relationships. (FOFO '89). Berlin, Heidelberg: Springer-Verlag. 1989. p. 457–472. Available from: https://link.springer.com/chapter/10.1007/3-540-51295-0_148

[35] CLEMENTINI, E., P. DI FELICE and P. VAN OOSTEROM. A small set of formal topological relationships suitable for end-user interaction. Berlin, Heidelberg: Springer Berlin Heidelberg. 1993. p. 277. Available from: https://link.springer.com/chapter/10.1007/3-540-56869-7_16

[36] RANDELL, D. A., Z. CUI and A. G. COHN. A spatial logic based on regions and connection. (KR'92). San Francisco, CA, USA: Morgan Kaufmann Publishers Inc. 1992. p. 165–176

[37] CAR, N. J. and T. HOMBURG. GeoSPARQL 1.1: Motivations, Details and Applications of the Decadal Update to the Most Important Geospatial LOD Standard. *ISPRS International Journal of Geo-Information*. 2022, vol. 11. Available from: https://www.mdpi.com/2220-9964/11/2/117

Price based on 108 pages