GeoSPARQL User Guide

1/19/2012

Dave Kolas (dkolas@bbn.com)  
Robert Battle (rbattle@bbn.com)

Table of Contents

[Introduction 2](#_Toc314745961)

[GeoSPARQL at a Glance 2](#_Toc314745962)

[The GeoSPARQL Ontology 3](#_Toc314745963)

[GeoSPARQL Key Terms 3](#_Toc314745964)

[Linking an ontology to GeoSPARQL 5](#_Toc314745965)

[Example Ontology 5](#_Toc314745966)

[Example Data 5](#_Toc314745967)

[Converting W3C Geo Data 6](#_Toc314745968)

[Querying Your Data 7](#_Toc314745969)

[Retrieving Geometries 7](#_Toc314745970)

[Topological Relationships 7](#_Toc314745971)

[Topological Functions 7](#_Toc314745972)

[Topological Properties 8](#_Toc314745973)

[Other Query Functions 9](#_Toc314745974)

[Feature to Feature relations 9](#_Toc314745975)

[Coordinate Reference Systems 9](#_Toc314745976)

[Using GML instead of WKT 10](#_Toc314745977)

[Alternate spatial relation types 10](#_Toc314745978)

[Prefixes 11](#_Toc314745979)

# Introduction

GeoSPARQL is an emerging standard within the Open Geospatial Consortium (OGC). Its intent is to provide a standard way to express and query spatial elements in RDF, so that users can exchange data easily, and triple store implementors can have a standard format for indexing.

The purpose of this document is to provide an easy introduction to GeoSPARQL for Semantic Web users. This document assumes a working understanding of [RDF](http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/) and [RDFS](http://www.w3.org/TR/rdf-schema/), and the [Turtle](http://www.w3.org/TeamSubmission/turtle/) serialization of RDF. This document is not intended to be exhaustive, but more topics and deeper explanations may be added over time.

The full GeoSPARQL specification is available here <TODO>, and includes further example data, example queries, and their answers.

To keep things simple, this guide will focus on Well Known Text (WKT) and Simple Features spatial relationships for most of the examples. Using GML instead of WKT is discussed here, and other spatial relation types are discussed here.

GeoSPARQL includes a number of different conformance classes, allowing certain useful subsets of the specification to be implemented. We assume in this document, unless otherwise noted, that the triple store implements all of GeoSPARQL. Consult your triple store’s documentation to see features / conformance classes that were omitted.

To make the example data and queries shorter and clearer, all prefix definitions are pushed to the end of the document in the section Prefixes. Some of the examples in this document are adapted from the paper *Enabling the Geospatial Semantic Web with Parliament and  GeoSPARQL*.

# GeoSPARQL at a Glance

To quickly show the basics of GeoSPARQL, we will start with one simple piece of data and an associated bounding box query. First, we define the Washington Monument as a “feature,” and give it a point in space.

ex:WashingtonMonument a geo:Feature;

rdfs:label "Washington Monument";

geo:hasGeometry ex:WMPoint .

ex:WMPoint a sf:Point;

geo:asWKT "POINT(-77.03524 38.889468)"^^sf:WktLiteral.

Looking at this example line by line, the first line indicates that the resource ex:WashingtonMonument is a “feature,” or a thing in the real world that can have a location. The second line gives this new resource a label. The third line gives this feature a spatial location. The fourth line says that the spatial location is a Point, and the fifth line defines the point.

*Note: By default, points are represented in longitude latitude order, in the WGS84 datum. Representation of geometries in different Coordinate Reference Systems is covered in* Coordinate Reference Systems*.*

Now we can execute a query looking for features in a particular latitude and longitude range, a “bounding box” query.

SELECT ?f

WHERE {

?f geo:hasGeometry ?g .

?g geo:asWKT ?gWKT .

FILTER (geof:sfWithin(?gWKT,

"POLYGON ((-77.2 38.8, -77 38.8, -77 39, -77.2 39.9,

-77.2 38.8))"^^sf:wktLiteral))

}

This query looks for resources which have a geometry that is within a bounding box rectangle. The first line finds geometries associated with a resource. The second line gets the geometries’ representation. The filter compares the representation with a bounding box to see if the geometry is within the bounding box.

The bounding box polygon actually has five points in it instead of just the four corners; the repetition of the first point in the fifth point closes the polygon.

# The GeoSPARQL Ontology

In order to execute GeoSPARQL queries on a dataset, the data must have its spatial portion expressed in the GeoSPARQL ontology. This is a straightforward thing to do, and it is compatible with whatever specific domain ontology is needed. The ontology can be found here <TODO>.

## GeoSPARQL Key Terms

There are three key classes in the GeoSPARQL ontology. These are:

* geo:Feature – A thing that can have a spatial location; i.e., a park or a monument etc.;
* geo:Geometry – A representation of a spatial location; i.e., a set of coordinates;
* geo:SpatialObject – A superclass of both Features and Geometries.

Their basic relationship looks like this:

hasGeometry

The geo:hasGeometry property links Features (a thing) to their Geometry (their location). By separating the actual entities and their locations, GeoSPARQL allows multiple Geometries to be linked to a Feature for varying purposes.

The geometry resource then has an RDF literal representation, which is linked with a property named for the type of representation. In the first example, the geo:asWKT property links the Geometry resource to a wktLiteral. The literal holds the actual geometry information.

Because GeoSPARQL uses WKT, WKT geometry types are available in GeoSPARQL. These are explained fully in the Simple Features specification [ISO 19125], but four primary types of geometries are especially useful:

|  |  |  |  |
| --- | --- | --- | --- |
| **TYPE** | **SHAPE** | **Geometry Class** | **SYNTAX** |
| **POINT** |  | sf:Point | POINT(longitude latitude) |
| **LINESTRING** |  | sf:LineString | LINESTRING(long1 lat1, long2 lat2, …) |
| **POLYGON** |  | sf:Polygon | POLYGON((long1 lat1, long2 lat2, … , long1 lat1)) |
| **POLYGON (WITH HOLE)** |  | sf:Polygon | POLYGON((long1 lat1, long2 lat2, … , long1 lat1), (longA latA, longB latB, …, longA latA)) |

To make these WKT geometries into RDF literals, they are simply wrapped in quotes and given the geo:WktLiteral datatype: "POINT(-77.03524 38.889468)"^^geo-sf:wktLiteral.

To create a WKT geometry, a resource should be declared to be the appropriate type from the table above, and given an asWKT property with a literal of the appropriate form. For example, to declare a Polygon geometry in RDF, the following statements would be used:

ex:Point1 a sf:Polygon;

sf:asWKT "POLYGON ((-77.2 38.8, -77 38.8, -77 39, -77.2 39.9,

-77.2 38.8))"^^sf:wktLiteral.

## Linking an ontology to GeoSPARQL

The GeoSPARQL ontology is small, and is meant to be connected to the ontology of a particular domain. This is easily achieved by making a class in the domain ontology a subclass of geo:Feature, meaning that instances of the class can point to a geo:Geometry with the geo:hasGeometry property.

## Example Ontology

The following is an extremely simple Points of Interest ontology to show how an ontology can connect to GeoSPARQL. The last line in red is the key statement.

ex:Restaurant a owl:Class;

rdfs:subClassOf ex:Service .

ex:Park a owl:Class;

rdfs:subClassOf ex:Attraction .

ex:Museum a owl:Class;

rdfs:subClassOf ex:Attraction .

ex:Monument a owl:Class;

rdfs:subClassOf ex:Attraction .

ex:Service a owl:Class;

rdfs:subClassOf ex:PointOfInterest .

ex:Attraction a owl:Class;

rdfs:subClassOf ex:PointOfInterest .

ex:PointOfInterest a owl:Class;

rdfs:subClassOf geo:Feature .

## Example Data

The following data creates an example Monument and an example Park, with a Point and Polygon geometry, respectively.

ex:Monument1 a ex:Monument;

rdfs:label "Washington Monument";

geo:hasGeometry ex:Point1 .

ex:Point1 a geo:Point;

geo:asWKT "POINT(-77.03524 38.889468)"^^geo-sf:WktLiteral.

Ex:Park1 a ex:Park;

rdfs:label "Example Park";

geo:hasGeometry ex:Polygon1 .

ex:Polygon1 a geo:Polygon;

geo:asWKT "POLYGON((-77.05 38.87, -77.02 38.87, -77.02 38.9, -77.05 38.9, 77.05 38.87))"^^geo-sf:WktLiteral.

## Converting W3C Geo Data

The W3C Geo data representation is a widely used simple representation for point data. It is a straightforward process to convert data from this format into GeoSPARQL. It is simply a matter of concatenating the longitude and latitude into a WKT point. If your triple store supports SPARQL 1.1, the following SPARQL CONSTRUCT query can return a converted graph:

CONSTRUCT {

?feature a geo:Feature ;

geo:hasGeometry [

a sf:Point ;

geo:asWKT ?wkt

] .

}

WHERE {

?feature a gn:Feature ;

wgs84\_pos:lat ?lat ;

wgs84\_pos:long ?long .

BIND (STRDT(CONCAT("POINT(",?long, " ", ?lat, ")"),sf:WktLiteral) as ?wkt) .

}

If a SPARQL 1.1 processor is not available, this process can be executed programmatically instead.

# Querying Your Data

## Retrieving Geometries

The most basic type of GeoSPARQL querying is simply retrieving Geometry information back from the triple store. To do this, you simply need to form a SPARQL query in the same ontology as the data was inserted:

SELECT ?wkt

WHERE {

ex:Monument1 geo:hasGeometry ?g .

?g geo:askWKT ?wkt .

}

## Topological Relationships

With GeoSPARQL, you can also do topological comparisons between geometries. There are three ways to do this.

1. Use GeoSPARQL filter functions
2. Use geometry-to-geometry properties
3. Use feature-to-feature properties

### Topological Functions

With the filter functions, you can compare two geometries that are fetched from the triple store, or you can compare a geometry from the triple store and one that is explicitly stated in the query. The previous bounding box query was an example of the latter. That example is repeated here:

SELECT ?f

WHERE {

?f geo:hasGeometry ?g .

?g geo:asWKT ?gWKT .

FILTER (geof:sfWithin(?gWKT,

"POLYGON ((-77.2 38.8, -77 38.8, -77 39, -77.2 39.9,

-77.2 38.8))"^^sf:wktLiteral))

}

The filter function in this example is geof:sfWithin. All of the topological comparisons from Simple Features are available:

|  |
| --- |
| **Functions** |
| geof:sfEquals |
| geof:sfDisjoint |
| geof:sfIntersects |
| geof:sfTouches |
| geof:sfWithin |
| geof:sfContains |
| geof:sfOverlaps |
| geof:sfCrosses |

### Topological Properties

The above topological comparisons can also be used to compare two geometries in the triple store by a direct property between the geometries. For instance, to find Monuments that were within ex:Park1, we could issue the following query:

SELECT ?f

WHERE {

ex:Park1 geo:hasGeometry ?g1 .

?f a ex:Monument;

geo:hasGeometry ?g2 .

?g2 geo:sfWithin ?g1 .

}

Note that the prefix on the topological *properties* differs from that of the topological *functions.* However, the same set of relations is available:

|  |
| --- |
| **Properties** |
| geo:sfEquals |
| geo:sfDisjoint |
| geo:sfIntersects |
| geo:sfTouches |
| geo:sfWithin |
| geo:sfContains |
| geo:sfOverlaps |
| geo:sfCrosses |

There are also other sets of topological relations that can be applied, whose functionality overlaps that of the simple features. They are discussed in Alternate spatial relation types.

The third way to compare spatial topology, feature-to-feature properties, is discussed in the next section.

## Other Query Functions

There are other query functions available for calculating distance, buffering objects, etc. These are listed below. For further explanation of these functions, consult the specification.

|  |  |  |
| --- | --- | --- |
| **Properties** | **Parameters** | **Returns** |
| **geof:distance** | Geom1, Geom2, unitsURI | xsd:double |
| **geof:buffer** | Geom1, radius, unitsURI | Geometry literal |
| **geof:convexHull** | Geom1 | Geometry literal |
| **geof:intersection** | Geom1, Geom2 | Geometry literal |
| **geof:union** | Geom1, Geom2 | Geometry literal |
| **geof:difference** | Geom1, Geom2 | Geometry literal |
| **geof:symDifference** | Geom1, Geom2 | Geometry literal |
| **geof:envelope** | Geom1 | Geometry literal |
| **geof:boundary** | Geom1 | Geometry literal |
| **geof:getsrid** | Geom1 | SRID of literal |

# Feature to Feature relations

Feature to feature relations allow topological relationships to be queried between Features as opposed to Geometries for convenience. For instance, instead of representing the query “Which Monuments are inside ex:Park1 as it is expressed in the previous section, it could be expressed like this:

SELECT ?f

WHERE {

ex:Park1 geo:sfWithin ?g1 .

?f a ex:Monument .

}

To use this method of querying, the Features in question need to have a *default* geometry. This allows the triple store to find the default geometry for each feature and use it for the feature-to-feature comparison. Default geometries are expressed with the geo:hasDefaultGeometry property instead of the geo:hasGeometry property.

Feature to feature relations may not be implemented in a particular triple-store.

# Coordinate Reference Systems

The default coordinate reference system in WKT literals is <http://www.opengis.net/def/crs/OGC/1.3/CRS84>, meaning using WGS84 and a longitude latitude order. To use a different CRS in a WKT literal, simply prepend it in angle brackets at the beginning of the literal:

"<http://www.opengis.net/def/crs/EPSG/0/4326> POINT(38.889468 -77.03524)"^^geo-sf:WktLiteral

# Using GML instead of WKT

If you prefer to use GML representations for Geometries, rather than WKT, you need only change three things. The URL’s for the Geometry types will be different, as listed below. You will need to use the datatype geo:GMLLiteral for literals, and the geo:asGML property is used instead of geo:asWKT. Examples follow.

A GML Geometry subclass is just the GML type (i.e., Polygon) and the prefix http://www.opengis.net/def/gml/. Thus the RDF type for a GML polygon is:

http://www.opengis.net/def/gml/Polygon

The example monument from above, described in GML instead of WKT, looks like this:

ex:Monument1 a ex:Monument;

rdfs:label "Washington Monument";

geo:hasGeometry ex:Point1 .

ex:Point1 a gml:Point;

geo:asGML "<gml:Point

srsName=\"http://www.opengis.net/def/crs/OGC/1.3/CRS84\"

xmlns:gml=\"http://www.opengis.net/gml\">

<gml:pos>-83.38 33.95</gml:pos>

</gml:Point>"^^gml:gmlLiteral

# Alternate spatial relation types

Along with the Simple Features topological relations listed in the previous sections, GeoSPARQL also includes both the Egenhofer spatial relations from the 9-intersection model and the RCC8 relations as well. The URIs for these are listed below.

Egenhofer Topological Relations

|  |  |
| --- | --- |
| **Properties** | **Functions** |
| geo:ehEquals | geof:ehEquals |
| geo:ehDisjoint | geof:ehDisjoint |
| geo:ehMeet | geof:ehMeet |
| geo:ehOverlap | geof:ehOverlap |
| geo:ehCovers | geof:ehCovers |
| geo:ehCoveredBy | geof:ehCoveredBy |
| geo:ehInside | geof:ehInside |
| geo:ehContains | geof:ehContains |

RCC8 Topological Relations

|  |  |
| --- | --- |
| **Properties** | **Functions** |
| geo:rcc8eq | geof:rcc8eq |
| geo:rcc8dc | geof:rcc8dc |
| geo:rcc8ec | geof:rcc8ec |
| geo:rcc8po | geof:rcc8po |
| geo:rcc8tppi | geof:rcc8tppi |
| geo:rcc8tpp | geof:rcc8tpp |
| geo:rcc8ntpp | geof:rcc8ntpp |
| geo:rcc8ntppi | geof:rcc8ntppi |

# Prefixes

Here the prefixes used in this document are gathered for ease of use. They are repeated twice; once in Turtle form for input data, and once in SPARQL form for queries.

Turtle Prefixes

@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .  
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .  
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

@prefix geo: <http://www.opengis.net/def/geosparql/> .  
@prefix sf: <http://www.opengis.net/def/sf/> .  
@prefix gml: <http://www.opengis.net/def/gml/> .

@prefix ex: <http://example.org/exampleOntology/> .

SPARQL Prefixes

PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>   
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>   
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

PREFIX geo: <http://www.opengis.net/def/geosparql/>   
PREFIX geof: <http://www.opengis.net/def/geosparql/function/>   
PREFIX sf: <http://www.opengis.net/def/sf/>   
PREFIX gml: <http://www.opengis.net/def/gml/>

PREFIX ex: <http://example.org/exampleOntology/>