TOVE Digital City Programming Manual  
Part III: Spatial Representation and Reasoning

Mark S. Fox, [msf@eil.utoronto.ca](mailto:msf@eil.utoronto.ca)

21 March 2021

# Introduction

Part III of the TOVE Digital City (TDC) Programming Manual focuses on spatial reasoning. Based on the spatial location pattern of 5087-1 found at http://ontology.eil.utoronto.ca/5087/spatialloc.owl, it makes the following assumptions:

1. The core spatial objects and relations are based on GeoSparql. The core objects used are geo:Point and geo:Polygon.
2. The specification of a Point and Polgon within the ontology is a WKTLiteral. For example:
   * Point is specified as 'POINT(-77.03524 38.889468) ' and
   * Polygon as 'POLYGON((-77.050125 38.892086, -77.039482 38.892036, -77.039482 38.895393, …))'
3. The core relations are geo:sfEquals, geo:sfTouches, geo:sfWithin, geo:sfContains, geo:sfDisjoint, geo:sfCrosses, geo:sfIntersects., sfOverlaps

Shape

Description automatically generated

1. The association of geometric information with city places is based on GeoSparql’s properties:

Diagram

Description automatically generated

where a geoSparql geometry is linked to a feature by the geo:hasGeometry property.

1. ISO/IEC 5087 is the core ontology for representing city data. The geometry of an area of a city is specified as follows:

|  |  |  |
| --- | --- | --- |
| Instance | Property | Value |
| Toronto | rdf:type | 5087-2:City |
| 5087-1:hasLocation | torontoBorder |
| torontoBorder | rdf:type | 5087-1:Feature |
| geo:hasGeometry | toPoly |
| toPoly | rdf:type | geo:Polygon |
| asWKT | 'POLYGON((…))' |

In the remainder of this report, we use the following ontology prefix’s:

|  |  |
| --- | --- |
| **Prefix** | **IRI** |
| geo | http://www.opengis.net/ont/geosparql# |

# GeoSpatial Functions (http://ontology.eil.utoronto.ca/dt/code/spatialloc.py)

TDC adopts the ISO/IEC 5087-1 Location Pattern which in turn is based on the GeoSparql standard. The following are basic functions for manipulating spatial objects.

|  |  |
| --- | --- |
| **latLongAddress(pointWKT, locator=Nominatim(user\_agent='myGeocoder')):** | |
| *Returns the address for a latitude longitude specified as a geoSPARQL point.* | |
| **pointWKT** | the latitude longitude pair specified as a WKT point.  E.g., 'POINT(-77.03524 38.889468) ' |
| **locator** | Specifies the method used to find the address for the pointWKT Defaults to Nominatim(user\_agent='myGeocoder') |
| **Returns** | String – the address |
|  | |
| **addressLatLong(address, locator=Nominatim(user\_agent='myGeocoder')):** | |
| *Returns the latitude/longitude of an address as a WKT Point* | |
| **address** | String specification of the address,  e.g., “14 Richland Lane, Pittsburgh PA USA” |
| **locator** | Specifies the method used to find the lat/long for the address |
| **Returns** | WKT Point containing the lat/long for the address |
|  | |
| **locationInside(pointWKT, polygonWKT):** | |
| *Determines with the lat/long point is within the polygon. The polygon is intended to represent the spatial boundary of a city, neighbourhood, etc.* | |
| **pointWKT** | String in WKT format representing the lat/long point |
| **polygonWKT** | String in WKT format representing the spatial boundary as a Polygon  e.g., 'POLYGON(((-77.050125 38.892086, -77.039482 38.892036, -77.039482 38.895393, …))' |
| **Returns** | Boolean: True if pointWKT is within polygonWKT |
|  | |
| **spatialRelation(geoWKT1, geoWKT2):** | |
| *Determines the spatial relation between two geometries.* | |
| **geoWKT1** | String: Any geo object represented in WKT format. |
| **geoWKT2** | String: Any geo object represented in WKT format. |
| **Returns** | The spatial relation that exists between geoWKT1 and geoWKT2. One of:  {geo:sfEquals, geo:sfTouches, geo:sfWithin, geo:sfContains, geo:sfDisjoint, geo:sfCrosses, geo:sfIntersects}. |
|  | |
| **distance(geoWKT1, geoWKT2):** | |
| *Determines the distance between two geolocations.* | |
| **geoWKT1** | String: Any geo object represented in WKT format, e.g., “POINT(0 0)” |
| **geoWKT2** | String: Any geo object represented in WKT format. |
| **Returns** | The distance between the two geo locations. |
|  |  |

# References

Cox, S., and Little, C. (2020), “Time Ontology in OWL”, OGC Document Number OGC 16-071r3, <https://www.w3.org/TR/owl-time/>. Downloaded 3 February 2021.

Hobbs, J. R., & Pan, F. (2006). Time ontology in OWL. *W3C working draft*, *27*, 133.

Lamy JB. Owlready: Ontology-oriented programming in Python with automatic classification and high level constructs for biomedical ontologies. Artificial Intelligence In Medicine 2017;80:11-28