# Spatial Functions

To use these functions, the Spatial Feature is required.

The spatial functions are based upon the JTS Topology Suite.

Notice, that JTS only considers x and y coordinate and ignores the z coordinate (although z can be definied)!



## AsCartesianCoordinates(SpatialPolarCoordinate[])

Transform the list of spatial polar coordinates into a list of Cartesian coordinates

## AsGeometry(Geometry)

## AsGeometryCollection(Geometry)

## AsLineString(Geometry)

## AsMultiLineString(Geometry)

## AsMultiPoint(Geometry)

## AsMultiPolygon(Geometry)

## AsPoint(Geometry)

## AsPolarCoordinates()

## FromWKT(String)

Transforms a String in WKT to a Geometry in Odysseus. If you have x and y (i.e. Latitude and Longitude) in your data stream, you can use this function as follows to create a Geometry.

```
#PARSER PQL
#QNAME VoyageStream
#RUNQUERY

proj = PROJECT({ATTRIBUTES = ['x', 'y', 'VoyageID']}, System.vesselVoyages)

/// Use x and y to create a geo-object
createWKTstring = MAP({EXPRESSIONS = [['"POINT (" + toString(x) + " " +
toString(y) + ") "', 'wktString']]}, proj)
createSpatialObject = MAP({EXPRESSIONS = [['FromWKT(wktString)',
'SpatialPoint']]}, createWKTstring)
```

We support the SRID field of EWKT as well as a custom extension. With "id=1234" you can set the id of the feature. This is used in serialization into GeoJSON, where an id field for a feature is necessary.

```
/// Create a geoObject from lat/lng
output = MAP({
        expressions = [
                ['FromWKT ("SRID=4326;ID=" + toString(ID_ATTRIBUTE) + ";
POINT (" + toString(latitude) + " " + toString(longitude) + ") ")',
'location'],
        ],
        name =
'geo_object'

        },
        input
)
```

## GetCentroid(Geometry)

Returns the centroid of the given geometry

## GetXFromSpatial / GetYFromSpatial

Extracts the X or Y value of a geometry. In Odysseus, x is typically latitude and y is longitude.

```
/// Get latitude and longitude from geo-object
extractLatLong = MAP({
                      expressions = [
                        ['GetXFromSpatial(location)','latitude'],
                        ['GetYFromSpatial(location)','longitude']
                      ],
                      keepinput =
true
                    },
                    input
                  )
```

## OrthodromicDistance(Geometry, Geometry)

Distance between the Geometries in meters on the earths surface. Internally, the first coordinate of the respective geometry is used. Hence, for points, this should be accurate. For other geometries, accuracy can vary. (see https://en.wikipedia.org/wiki/Great-circle_distance)

## SpatialBuffer(Geometry, Double)

Creates a buffer of the given size around the given geometry

## SpatialContains(Geometry, Geometry)

Checks whether the first geometry contains the second geometry.

## SpatialConvexHull(Geometry)

Computes the smallest convex spatial `polygon` that contains all the points in the `geometry`

## SpatialCoveredBy(Geometry, Geometry)

## SpatialCovers(Geometry, Geometry)

## SpatialCrosses(Geometry, Geometry)

## SpatialDisjoint(Geometry, Geometry)

## SpatialDistance(Geometry, Geometry)

Calculates the distance between the two points. The distance is given in degrees, as the function does not know anything about the used geodetic datum. Hence, it can be used without an SRID. Internally, the distance-function from the JTS is used.

**Example SpatialDistance**

```
distance = MAP({
                expressions = ['ship', 'neighbors',
'neighbor_coordinates', ['spatialDistance(ship,
neighbor_coordinates)','distance']]
              },
              tupleAsList
            )
```

## SpatialEquals(Geometry, Geometry)

### SpatialIntersection(Geometry, Geometry)

### SpatialIsLine(Geometry)

### SpatialIsPolygon(Geometry)

### SpatialIsWithinDistance(Geometry, Double)

### SpatialTouches(Geometry, Geometry)

### SpatialUnion(Geometry, Geometry)

### SpatialUnionBuffer(Geometry, Geometry, Geometry)

### SpatialWithin(Geometry, Geometry)

Checks if the first Geometry is within the second Geometry.

### ST_SetSRID(Geometry, Integer)

**Example ST_SetSRID**

```
/// Use x and y to create a geo-object
createWKTstring = MAP({EXPRESSIONS = [['"POINT (" + toString(X) + " " +
toString(Y) + ") "', 'wktString']]}, region1)
createSpatialObject = MAP({EXPRESSIONS = [['FromWKT(wktString)',
'SpatialPoint']]}, createWKTstring)
/// Set SRID for WGS84
setSRID = MAP({EXPRESSIONS = [['ST_SetSRID(SpatialPoint, 4326)',
'withSRID']]}, createSpatialObject)
```

### ST_Transform(Geometry, Integer)

### ToCartesianCoordinate(SpatialPolarCoordinate)

Transforms the given spatial polar coordinate into a spatial Cartesian coordinate.

### ToPolarCoordinate(SpatialCoordinate)

Transforms the given spatial Cartesian coordinate into a polar coordinate