

# Unit 13

Geospatial Data, Indexing and Search

Configure Geospatial Indexes

Load Geospatial Data

Build a Geospatial Search Query

## Exercise 1: Configure Geospatial Indexes

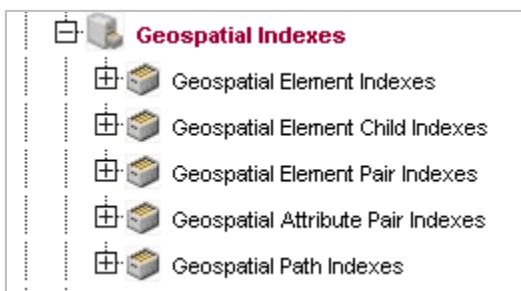
In this exercise we will activate the appropriate geospatial index for the Geophoto database. To activate the correct geospatial index, we need to determine how our geospatial data is structured in our database content.

1. When we load the data for the Geophoto app, we will have a document structure as follows:



```
{
  "originalFilename": "../data/photos/IMG_0661.jpg",
  "filename": "IMG_0661.jpg",
  "binary": "/binary/IMG_0661.jpg",
  "make": "Apple",
  "model": "iPhone 5",
  "created": 1382380516000,
  "location": {
    "type": "Point",
    "coordinates": [
      37.790342,
      -122.422158
    ]
  },
  "city": "San Francisco",
  "country": "United States"
}
```

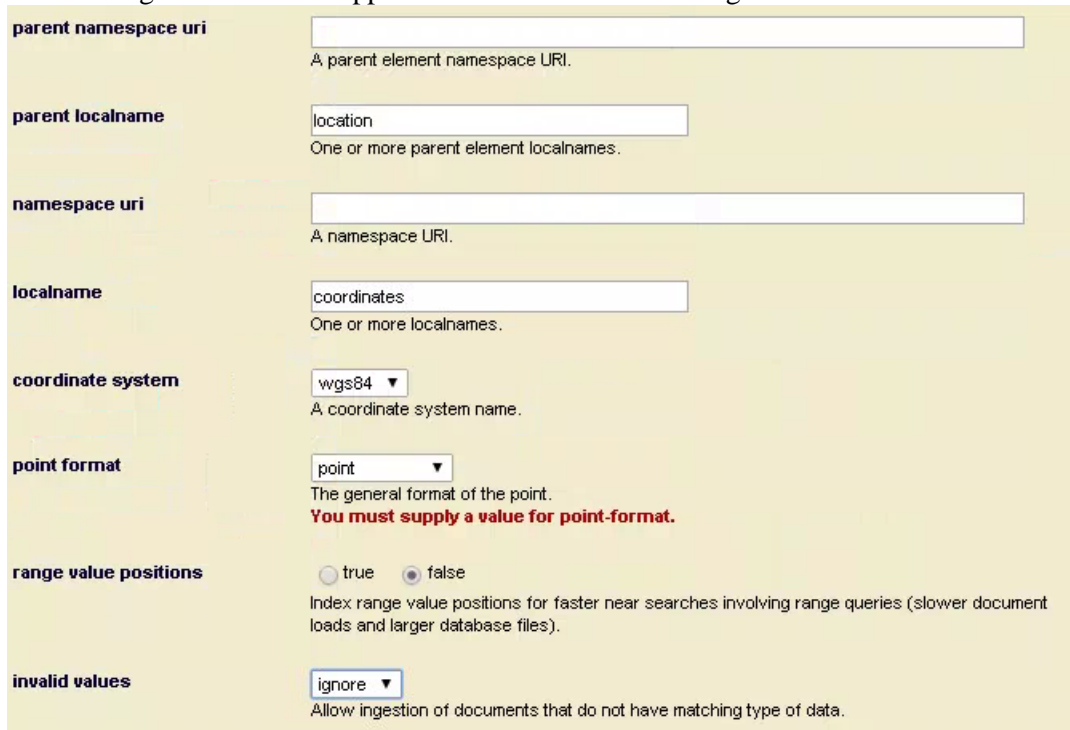
2. Notice the **location** property which contains a **coordinates** property.
3. In the browser, open the Admin interface on port 8001.
4. Select **Configure – Databases – geophoto-content – Geospatial Indexes**.



5. Select **Geospatial Element Child Indexes**. Our data is structured as **Element Child** because we have a defined parent property containing a defined child property with both the latitude and longitude data.
6. Click **Add**.
7. Fill out the fields and take care that the spelling is exactly as follows:

- Parent namespace uri: **leave this field blank**
- Parent localname: **location**
- Namespace uri: **leave this field blank**
- Localname: **coordinates**
- Coordinate system: **wgs84**
- Point format: **point**
- Range value positions: **false**
- Invalid values: **ignore**

8. Your configuration should appear as shown in the following screenshot:



The screenshot shows a configuration form with the following fields and values:

- parent namespace uri:** (empty text box)
- parent localname:** location
- namespace uri:** (empty text box)
- localname:** coordinates
- coordinate system:** wgs84
- point format:** point
- range value positions:** false
- invalid values:** ignore

Below each field is a descriptive text: "A parent element namespace URI.", "One or more parent element localnames.", "A namespace URI.", "One or more localnames.", "A coordinate system name.", "The general format of the point.", "Index range value positions for faster near searches involving range queries (slower document loads and larger database files).", and "Allow ingestion of documents that do not have matching type of data." respectively.

9. Click **OK** to create the index.
10. While we created this index manually, we could have also documented the desired configuration and deployed it using the Management REST API.
11. Navigate to **c:\mls-developer-node\Unit13\** and open **geophoto-database-config.json**.
12. Take a minute to study the configuration. Note that along with the Geospatial index, it also will create some element range indexes which the application will need:

```
[
  "triple-index": true,
  "uri-lexicon": true,
  "collection-lexicon": true,

  "geospatial-element-child-index": [
    {
      "parent-namespace-uri": "",
      "parent-localname": "location",
      "namespace-uri": "",
      "localname": "coordinates",
      "coordinate-system": "wgs84",
      "point-format": "point",
      "range-value-positions": false,
      "invalid-values": "ignore"
    }
  ],
  "range-element-index": [
    {
      "scalar-type": "string",
      "namespace-uri": "",
      "localname": "country",
      "collation": "http://marklogic.com/collation/",
      "range-value-positions": false,
      "invalid-values": "reject"
    },
    {
      "scalar-type": "string",
      "namespace-uri": "",
      "localname": "city",

```

13. Open a command prompt window and navigate to **c:\curl**.
14. Navigate to **c:\mls-developer-node\Unit13\** and open **geophoto-deploy-curl.txt**.
15. Study the cURL command. Copy / paste the cURL command into the command prompt to execute it.
16. Validate that you receive the 204 response indicating that the configuration was applied successfully:

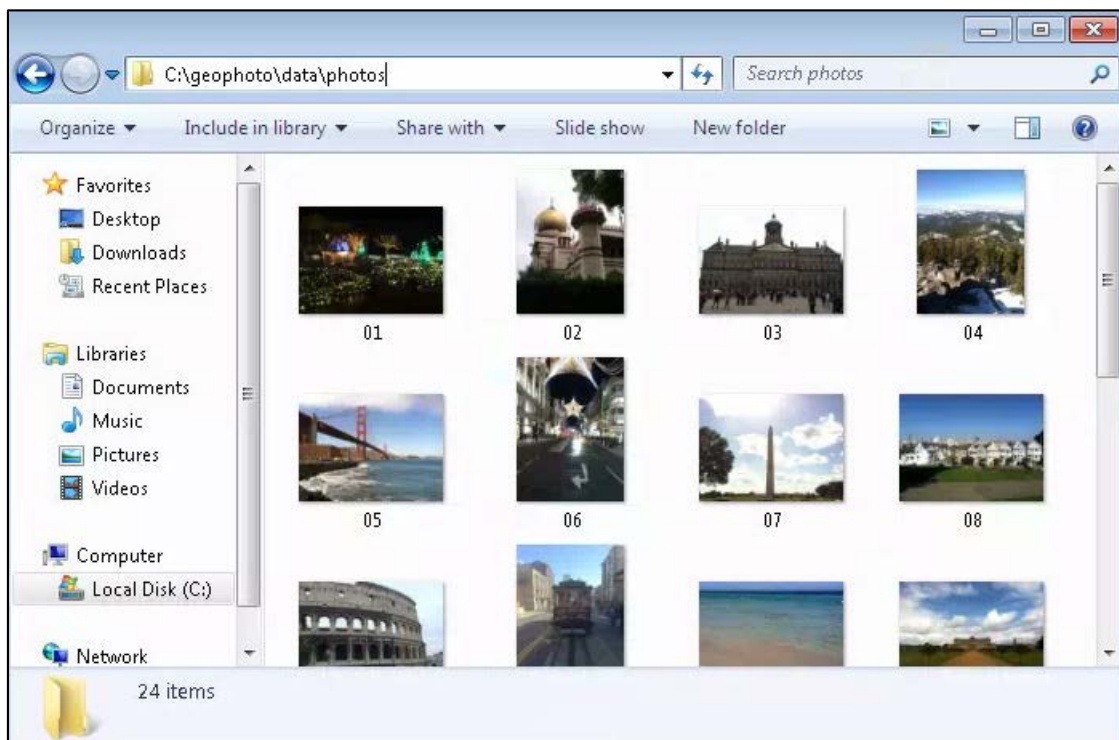
```
HTTP/1.1 204 No Content
Server: MarkLogic
Content-Length: 0
Connection: Keep-Alive
Keep-Alive: timeout=5
```

17. Validate the index configurations for the **geophoto-content** database using the Admin interface.

## Exercise 2: Load Geospatial Data

In this exercise you will load data for the Geophoto project.

1. In Windows Explorer, navigate to **c:\geophoto\data\photos**.
2. Notice this directory contains several image files. These images were taken with on a mobile phone with geo location enabled. So we actually have images and metadata about those images that was captured. That metadata includes geospatial information which we will extract from the images and store as JSON in the database.



3. Open a command prompt and navigate to **c:\geophoto\import**.
4. To load the photos, you need to launch the **import.js** module and pass it a parameter indicating the location of the photos you wish to load.
5. At the command line key **node import.js ../data/photos** and press enter:

```
c:\geophoto\import>node import.js ../data/photos
```

*Note:*

*The load may take a few minutes depending on connection speeds.*

*This load process is performing several transforms including extracting the EXIF metadata from the images, reverse geo-encoding lookups to determine the city and country information for the photo, and SPARQL queries to build triple data about the location of the photo.*

6. When the load is completed, you will get a report of all the load activity.
7. Note that a SJS extension was also loaded to the modules database. This is used to gather semantic data about the various countries where photos were taken:



8. Note that you have JPEG (binary) and JSON data:

```
JPEG file inserted /binary/01.JPG
JSON file inserted /image/02.JPG.json
JSON file inserted /image/IMG_20140801_102607.jpg.json
JSON file inserted /image/01.JPG.json
JSON file inserted /image/05.JPG.json
JPEG file inserted /binary/IMG_20140801_102607.jpg
JSON file inserted /image/08.JPG.json
JPEG file inserted /binary/IMG_3942.jpg
JSON file inserted /image/IMG_0132.JPG.json
JSON file inserted /image/IMG_0924.JPG.json
JSON file inserted /image/03.JPG.json
```

9. Note that you also have semantic data (more on that in the next Unit):

```
Calling semantic info for: Colombia
Calling semantic info for: Germany
Calling semantic info for: United_States
Calling semantic info for: Hungary
Calling semantic info for: Netherlands
Calling semantic info for: Belgium
Calling semantic info for: United_Kingdom
Calling semantic info for: Singapore
Calling semantic info for: Italy
RDF triple inserted /triplestore/54fd7c768906eea2.xml
RDF triple inserted /triplestore/8a70a5a39882af44.xml
RDF triple inserted /triplestore/c473a0102afe92fb.xml
RDF triple inserted /triplestore/34a6ee707d72470c.xml
RDF triple inserted /triplestore/7bcc066a1fac4d4.xml
RDF triple inserted /triplestore/d591f104f9a3b88b.xml
RDF triple inserted /triplestore/3f11c35ebc7172cf.xml
```

10. Next, let's test the app.
11. From the command line, navigate to **c:\geophoto** and enter **node app.js**
12. You will see the following response:

```
c:\geophoto>node app.js
Magic happens on port 4000
```

13. Open a browser and go to <http://localhost:4000>
14. View the app and experiment with Geospatial search.

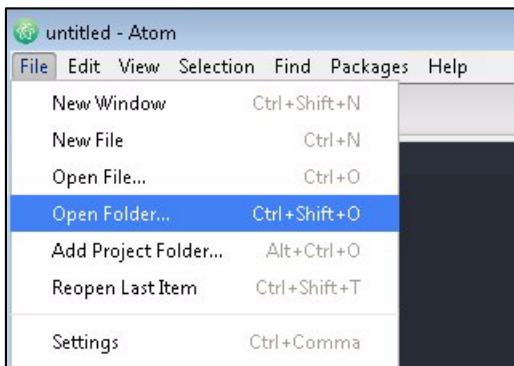
### Exercise 3: Build a Geospatial Search Query

In this exercise you will build searches with geospatial components.

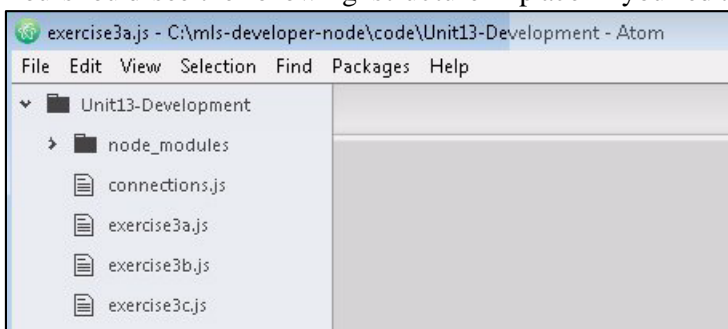
1. From the command line, navigate to the `c:\mls-developer-node\code\Unit13-Development` directory and run **npm install marklogic**.
2. Open the Atom editor from your Desktop:



3. In the Atom editor select **File→Open Folder...**:



4. Open the **Unit13-Development** folder from `c:\mls-developer-node\code\`
5. You should see the following structure in place in your editor:



6. Select **connections.js**



- Note that this is now pointing to the REST instance on port 5003, which is configured for the geophoto database:

```
restReader: {  
  host: 'localhost',  
  port: 5003,  
  user: 'rest-reader-user',  
  password: 'training'  
},
```

- Select **exercise3a.js**.
- Take a moment to study the comments and code.
- Note that a function is built to perform the circle search, and it accepts a radius, latitude and longitude. It then uses the `QueryBuilder` interface to create a structured query with geospatial components, and refers to the properties that contain the geospatial data and for which you have created the geospatial index:

```
function circle(radius, lat, lon) {  
  return dbRead.documents.query(  
    qb.where(  
      qb.geospatial(  
        qb.geoProperty(qb.property("location"), qb.property("coordinates")),  
        qb.circle(radius, lat, lon)  
      )  
    )  
  ).result();  
}
```

- Note the calling of the circle function and the output of the result data to the console:

```
var sanFranciscoLat = 37.7833;  
var sanFranciscoLon = -122.4167;  
  
circle(20, sanFranciscoLat, sanFranciscoLon).then(function(response) {  
  response.forEach(function(document) {  
    console.log(  
      document.uri + " matches your circle geo search (" +  
      document.content.location.city + ", " +  
      document.content.location.country + ")"  
    );  
  });  
})  
.catch(function(error) {  
  console.log("Error ", error);  
});
```

12. Now let's test the code.
13. At the command line, go to the **c:\mls-developer-node\Unit13-Development** directory.
14. Enter **node exercise3a.js** and press enter.
15. You should see the following response:

```
c:\mls-developer-node\code\Unit13-Development>node exercise3a.js
c:\mls-developer-node\code\Unit13-Development>node exercise3a.js
/image/08.JPG.json matches your circle geo search <San Francisco, United States>
/image/IMG_0661.jpg.json matches your circle geo search <San Francisco, United States>
/image/05.JPG.json matches your circle geo search <San Francisco, United States>
```

16. Perform the same review, analysis and testing for **exercise3b.js** and **exercise3c.js**.