

SolrAdaptersForLuceneSpatial4

{{{#!wiki red/solid ⚠️ 😞 The most up to date information about spatial search at the Solr Reference Guide instead: <https://cwiki.apache.org/confluence/display/solr/Spatial+Search>. Some details are still here, and will eventually be copied. This page will not be kept current. ⚠️ 😞}}}}

⚠️ Solr4.0

Lucene / Solr 4 Spatial

This document describes how to use the *new* spatial field types and related functionality in Lucene / Solr 4. The [existing spatial support](#) introduced in Solr 3 is still present and is still the default used in Solr's example schema – [LatLonType](#).

The bulk of the new spatial implementation lives in the new Lucene 4 spatial module. It replaces the former "Lucene spatial contrib" in v3. The Solr piece is small as it only needs to provide field types which are essentially adapters to the code in the Lucene spatial module. The shape implementations and other core spatial code that isn't related to Lucene is held in a new open-source project called [Spatial4j](#). Presently, polygon support requires an additional dependency – [JTS](#).

There is a basic [demo application](#) that exercises a variety of these features. It's not "live" so you'll have to download and build it first. It's a bit rough around the edges as it's mostly used by the Lucene spatial developers.

- [Lucene / Solr 4 Spatial](#)
 - [New features, over Solr 3 spatial](#)
- [How to Use](#)
 - [Configuration](#)
 - [Indexing](#)
 - [Search](#)
 - [Spatial Predicates](#)
 - [Sorting and Relevancy](#)
 - [Units, Conversion](#)
- [JTS / WKT / Polygon notes](#)
- [TODO](#)
- [Using spatial to search time ranges](#)

New features, over Solr 3 spatial

Note: "Solr 3 spatial" refers to the spatial support introduced in that version of Solr which still exists in v4. Except for a small utility class, Solr 3 spatial does *not* actually use Lucene 3's defunct spatial contrib module.

These features describe what developer-users of Lucene/Solr 4 will appreciate. Under the hood, it's a framework designed to be extended for different so-called "spatial strategies". I'll assume here the [RecursivePrefixTreeStrategy](#) as it should address most use-cases and it has the best tests.

- Polygon, [LineString](#) and other new shapes. All shapes are supported as indexed shapes and query shapes. Shapes other than point, rectangle and circle are supported via JTS – an otherwise optional dependency. See JTS caveats below for more information.
- Multi-valued indexed fields. This is critical for storing the results of automatic place extraction from text using natural language processing techniques with a gazetteer (a variant of "geocoding" / "geotagging"), since a variable number of locations will be found.
- Index non-point shapes as well as points. Non-point shapes are essentially pixelated (i.e. gridded) to a configured resolution per shape – an approximation. By default that resolution is defined by a percentage of the overall shape size, and it applies to query shapes too. Note: If extremely high precision of shape edges needs to be retained for accurate indexing, then this solution probably won't scale too well at indexing time (big indexes, slow indexing). On the other hand, *query* shapes generally scale well to the maximum configured precision regardless of shape size.
- Rectangles with user-specifiable corners. Oddly, Solr 3 spatial only supports the bounding box of a circle.
- Multi-value distance sort / score boost. ⚠️ Note: this is a preliminary unoptimized implementation that uses a fair amount of RAM, even when `multiValued=false`. An alternative should be provided in the future.
- Configurable precision which can vary per shape at query time (and sort of at index time). This enhances the performance.
- Fast filtering. The code was benchmarked once showing it outperforms Solr 3's "LatLonType" at its own game (single valued indexed points), and several 3rd parties anecdotally reported the same, especially for multi-million document indices. It is based on SOLR-2155 which was benchmarked in January 2010; so a new benchmark is a TODO item. Also, Solr 3 [LatLonType](#) sometimes requires all the points to be in memory, whereas the new spatial module here doesn't for filtering.
- [Well Known Text](#) (WKT) support via JTS. WKT is arguably the most widely supported textual format for shapes. However, standard WKT doesn't specify a format for circles.

Of course, the basics in Solr 3 not mentioned here are implemented in this framework. For example, lat-lon bounding boxes and circles.

How to Use

Configuration

First, you must register a spatial field type in the Solr schema.xml file. The instructions in this whole document imply the [RecursivePrefixTreeStrategy](#) based field type used in a geospatial context.

```
<fieldType name="location_rpt" class="solr.SpatialRecursivePrefixTreeFieldType"
  spatialContextFactory="com.spatial4j.core.context.jts.JtsSpatialContextFactory"
  distErrPct="0.025"
  maxDistErr="0.000009"
  units="degrees"
/>
```

And finally, specify a field that uses this field type:

```
<field name="geo" type="location_rpt" indexed="true" stored="true" multiValued="true" />
```

A key feature of the new spatial module is multi-value support but you certainly aren't required to declare the field multiValued if it isn't.

The following configuration attributes are common to all new spatial field types based on Lucene 4 spatial:

- **spatialContextFactory:** (Spatial4j) If polygons or other WKT formatted shape support is needed, then use the JTS based class as shown above, otherwise this can be omitted. The JTS jar file must be on Solr's classpath as well. Due to a combination of things, JTS can't simply be referenced by a "<lib>" entry in solrconfig.xml; it needs to be in WEB-INF/lib in Solr's war file, basically.
- **units="degrees":** This parameter is mandatory, and currently the only value supported is "degrees". This affects the interpretation of the maxDistErr attribute, circle radius distances, and other absolute distances. There are approximately 111.2 kilometers in a degree, based on the average earth radius.
- **geo="true":** Whether the spatial fields' coordinates are latitude / longitude WGS84 based (if true) or whether they are pure Euclidean / Cartesian based. It defaults to true. When set to false, you should indicate worldBounds and probably maxDistErr as well.
- **worldBounds="minX minY maxX maxY":** Set the valid numerical ranges for x & y. If geo="true" then this is assumed "-180 -90 180 90". When geo="false" this is the limits of a Java double however those values have been shown to not work (yet), so definitely choose your boundaries for non-geospatial uses.
- **distCalculator="haversine":** Set the distance calculation algorithm. If geo="true" then haversine is the default, otherwise cartesian is. The possible values are: haversine, lawOfCosines (warning: faulty), vincentySphere, cartesian, and cartesian^2.

A [PrefixTree](#) based field sees the world as a grid. Each grid cell is further decomposed as another set of grid cells at the next level. The first and largest cells have "level 1", the next detailed is "level 2" and so on. Here are the attributes specific to [PrefixTree](#) based fields:

- **prefixTree="geohash":** Choose the spatial grid implementation. "geohash" uses the Geohash algorithm which has 32 children at each level, and its limited to use when geo="true". The other implementation is "quad" which has 4 children a each level.
- **maxLevels="10":** Set the maximum level (aka grid depth) for indexed data. It's easier to think in terms of a real distance and use maxDistErr instead.
- **maxDistErr="0.000009":** The highest level of detail required for indexed data. If you specify nothing then it is a meter – which is just a hair less than 0.000009 degrees. The units of this attribute are as indicated in the "units" attribute. On initialization, the prefix tree will determine what maxLevels should be to satisfy the desired distance precision. Unless you pick a maxDistErr at an exact threshold, the actual distance error will be even more precise. maxLevels is logged at startup.
- **distErrPct="0.025":** Specifies the default precision of non-point shapes, as a fraction between 0.0 (fully precise up to maxLevels) and 0.5. Shapes are basically pixelated on an indexed grid. This number is approximated as the fraction of the distance between the center of a shape and the farthest corner of its bounding box. The closer this number is to zero, the more accurate the shape will be, but an indexed shape will use more disk space and it will take longer to index. The default is 2.5%. It applies to both index and query shapes, but it is overridable for query shapes.

A couple more obscure attributes are defaultFieldValuesArrayLen (affects memory use in distance sorting) and prefixGridScanLevel (tunes heuristics for filter performance).

Indexing

Points are indexed just as they are in Solr 3 spatial:

```
{{ <field name="geo">43.17614,-90.57341</field>}}
```

If a comma is omitted, then it is in x-y (lon-lat) order:

```
{{ <field name="geo">-90.57341 43.17614</field>}}
```

A lat-lon rectangle can be indexed with 4 numbers in minX minY maxX maxY order:

```
{{ <field name="geo">-74.093 41.042 -69.347 44.558</field>}}
```

A circle is specified like so:

```
{{ <field name="geo">Circle(4.56,1.23 d=0.0710)</field>}}
```


The first part of it is the center point, in either "lon lat" or "lat,lon" format, then *the "d" distance radius is in degrees* (see further below on units).

For polygons, use the WKT standard (Well Known Text) like so:

```
{{ <field name="geo">POLYGON((-10 30, -40 40, -10 -20, 40 20, 0 0, -10 30))</field>}}
```

(That requires JTS, remember). In WKT, coordinates are in "x y" (lon lat) order, and the coordinates are each separated by commas. (*The double parenthesis is not a typo; see the WKT spec.*)

Search

As of  Solr 4.2 it became possible to use the `{!geofilt}` and `{!bbox}` query parsers with the new spatial field type. Quick example:

```
fq={!geofilt pt=45.15,-93.85 sfield=geo d=5}
```

'd' has the units of kilometers when used in geofilt or bbox no matter what the field type may say. That may change in the future. Although it's nice to use Solr's existing established spatial syntax, that syntax has limitations – namely you can't choose any rectangle as a filter, it's limited to the bounding box of a circle. To break free of that limitation, you can use Solr's range query syntax for spatial like so:

```
fq=geo:[45,-94 TO 46,-93]
```

It's limited to the latitude comma longitude syntax without spaces. The left side has the lower left corner, and the right side has the upper right corner. And the nice thing here is that it will correctly work across the international dateline, unlike [LatLonType](#).

A new syntax that the new spatial field type supported right from version 4.0 is a field-query style approach; it isn't a conventional Solr query parser. You currently need to use this syntax to use a spatial predicate other than Intersects, or to use WKT formatted shapes (for e.g. a polygon), or to add some precision tuning options. Here are a couple examples, first with a bounding box (not in WKT) then a WKT example with a tuning option.


```
{{ fq=geo:"Intersects(-74.093 41.042 -69.347 44.558)"}}
```

```
{{ fq=geo:"IsWithin(POLYGON((-10 30, -40 40, -10 -20, 40 20, 0 0, -10 30))) distErrPct=0"}}
```

Notice that the unusual trick here is that the spatial part of the query is placed into what looks like a Lucene phrase query. The query starts with the spatial predicate (AKA [SpatialOperation](#), see below) and then the contents of the parenthesis is either WKT or the other simple shape formats seen above in the indexing examples.

A query shape will by default have the `distErrPct` precision specified by the field type definition, which defaults to 0.025 (2.5%). Interpretation of this figure was described earlier. The above example sets it to 0 making the query shape as accurate as the grid is (`maxDistErr`, by default 1 meter). You can also specify `distErr=...` to explicitly set the precision for the shape (as measured in degrees) for when you know what the accuracy should be in exact terms instead of in relative terms.

Spatial Predicates

Initially only "Intersects" was supported;  Solr 4.3 introduced "IsWithin", "Contains", and "IsDisjointTo".

If you only have indexed points, then don't use [IsWithin](#) or [Contains](#); keep it to [Intersects](#) (or [IsDisjointTo](#) if you need that).

If you have indexed non-point shapes, then you might want to search according to the "IsWithin" and "Contains" predicates relative to your query shape. To clarify, use "IsWithin" if you want to search for indexed shapes that are WITHIN the query shape. Vice-versa for [Contains](#).

The semantics of [IsDisjointTo](#) is defined as the inverse of [Intersects](#), but with the constraint that the field must have spatial data. No spatial predicate will match a document that has no spatial data.

Sorting and Relevancy

A common spatial requirement is to sort the search results by distance from a point such as the center of a map window. **This works quite differently than Solr 3 spatial; you can't use `geodist()`.** Instead, the spatial queries seen earlier are capable of returning a distance based score, which can then be sorted, used in relevancy boosting, and even returned in search results.


Here, we show parameters that do a spatial filter & sort & returns the distance (as the score) simultaneously:

```
&fl=*,score&sort=score asc&q={!geofilt score=distance sfield=geo pt=54.729696,-98.525391 d=10}
```

Adding a user keyword search in this case would be added as an 'fq' param, most likely with the leading `{!edismax}`. *Notice the `score=distance` local-param here.* Without this (or if set to "none"), the query would yield a constant 1.0 for all documents. With "distance", it is the distance in degrees from the center of the query shape to the indexed point(s). You'll probably want to sort these values ascending, as shown. Another option is "recipDistance" which will use the reciprocal function such that distance 0 yields a score of 1, and a distance at the edge of the query shape yields ~0.1, trailing down closer to 0 beyond that. The "recipDistance" option is intended for use in boosting relevancy, such as using it in `dismax`'s `boost` parameter.

If you want to sort and to have the distance in the results like in the last example, but don't want the spatial filter, you can do this too:

```
&fl=*,score&sort=score asc&q={!geofilt score=distance filter=false sfield=geo pt=54.729696,-98.525391 d=10}
```

The only change was adding `filter=false` (added in  Solr 4.2). The 'd' is then effectively irrelevant (but is still required for the shape definition, a circle here) as only the center point of the shape is pertinent for sorting. If a document has no point in the spatial field, the distance used is 0.

If you only need to return the distance but *don't need to sort*, then the most performant approach is to calculate it on the client based on the lat & lon from the search results. Google for the haversine algorithm and your language of choice and you'll find a code snippet. If you ask Solr to do it then it'll put all the points in memory needlessly, but it'll certainly work. This shortcoming may be addressed in the future.

Notes:

- If you index non-point data (e.g. polygons), then the [PrefixTree](#) based strategy won't work for sorting. You should index points you want to sort on separately, ideally with [LatLonType](#) if you can live with the 1 point per document limit.
- If you supply multiple points or other shapes, then the distance to the closest one is used. If you need different behavior then file an issue in JIRA and explain your use-case.
- The [PrefixTree](#) based field type has a sub-par implementation for caching the indexed points in memory, currently. Even if `multiValue=false`, it's going to use the same big array of List of Point objects in memory. It's wasteful and the implementation is not friendly to real-time search requirements. Until a better implementation arrives, if you have single-valued point fields then use [LatLonType](#) for sorting instead. [LatLonType](#) also allows the choice of a float based coordinate field which halves memory compared to doubles, yet getting less than 3 meters of precision.

Sorting in Solr, whether it be a number/date or one of these spatial fields, requires some memory for each document and spatial sorting can involve some non-trivial math performed numerous times. Consequently, don't apply sorting without an actual need / requirement, versus a "hey, why not?" choice. The first time you sort on a field (spatial or not) it will load some data into memory then. This "first time" is the first time since the last commit, to be precise. You probably want to do put the sort query into `firstSearcher` & `newSearcher` so that an end user's search won't get hit with that penalty.

Units, Conversion

Degrees to desired units: `Math.toRadians(degrees) * earthRadiusInTheUnitsYouWant`



Degrees to kilometers: `degrees * 111.1951`

Degrees to miles: `degrees * 69.09341`

Just divide instead of multiply to go the other way.

JTS / WKT / Polygon notes

Shapes other than point, circle, or rectangle require [JTS](#), an otherwise optional dependency. If you want to use [Well Known Text](#) (WKT) but only need the basic shapes, you still need JTS – a restriction likely to be addressed in the near future.

- Due to a combination of things, JTS can't simply be referenced by a "`<lib>`" entry in `solrconfig.xml`; it needs to be in `WEB-INF/lib` in Solr's war file, basically.
- JTS views the world as a flat plane; the latitude and longitude are mapped to this plane directly. It uses Euclidean math operations, not Geodetic ones. This effectively warps shapes slightly, although it can be a bit much if the vertices are particularly far apart longitudinally.
- Dateline crossing **is** supported. `Spatial4j` adapts shapes that cross the dateline to be compatible with JTS, and so you shouldn't notice a problem (notwithstanding unknown bugs).
- Pole wrapping **is not** supported. Consequently if you want to index or query by an Antarctica polygon for example, you are out of luck for now. The only shape that can encompass a pole is a Circle. Technically a longitude-wrapping (-180 to +180) lat-lon box that touches a pole will too.
- Only Polygon, and [MultiPolygon](#) WKT types have been tested. [GeometryCollection](#) will not work but the others like [LineString](#) should in theory. Holes in polygons haven't been tested but there is code in place to support them.
- WKT shapes must have each vertex less than 180 degrees in longitude difference than the vertex before it, or else it will be confused as going the wrong way around the globe. The only exception to this is a Polygon representing a rectangle.
- All WKT coordinates are normalized into the standard geospatial lat-lon boundaries. So, -184 longitude becomes +176, for example. Both +180 and -180 are kept distinct – true for all of `Spatial4j`, not just JTS.
-  Common error:  The standard way to specify a rectangle in WKT is a Polygon – WKT doesn't have a rectangle shape. If you want to specify a Rectangle via WKT (instead of the simple non-WKT syntax seen earlier), you should take care to specify the coordinates in counter-clockwise order, the WKT standard. If this is done wrong then the rectangle will go the opposite direction longitudinally, even if it means one that spans nearly the entire globe (>180 degrees width). [OpenLayers](#) seems to not honor the WKT standard here, and depending on the corner you drag the rectangle from, might use a clockwise order. Some systems like PostGIS don't care what the ordering is, but the problem there is that there is then no way to specify a rectangle that has ≥ 180 width because there would be ambiguity. `Spatial4j` follows the WKT spec.

TODO

- ability to pass `d` parameter for km or miles for small distances (helper?)

Using spatial to search time ranges

The new spatial support here can actually be used for searching and indexing time durations or other numeric ranges. See [SpatialForTimeDurations](#).