SchemaXml

The schema.xml file contains all of the details about which fields your documents can contain, and how those fields should be dealt with when adding documents to the index, or when querying those fields.

Analysis for Multiterm queries can be separately specified, see: Multiterm Query Analysis, which handles automatically lowercasing wildcard queries under most circumstances. ⚠️ Solr3.6 ⚠️ Solr4.0

A sample Solr schema.xml with detailed comments can be found in the Source Repository.

Data Types

The `<types>` section allows you to define a list of `<fieldtype>` declarations you wish to use in your schema, along with the underlying Solr class that should be used for that type, as well as the default options you want for fields that use that type.

Any subclass of FieldType may be used as a field type class, using either its full package name, or the "solr" alias if it is in the default Solr package. For common numeric types (integer, float, etc...) there are multiple implementations provided depending on your needs, please see SolrPlugins for information on how to ensure that your own custom Field Types can be loaded into Solr.

- Common options that field types can have are...
  - `sortMissingLast=true|false`
  - `sortMissingFirst=true|false`
  - `indexed=true|false`
  - `stored=true|false`
  - `multiValued=true|false`
  - `omitNorms=true|false`
  - `omitTermFreqAndPositions=true|false` ⚠️ Solr1.4
  - `omitPositions=true|false` ⚠️ Solr3.4
  - `positionIncrementGap=N`
  - `autoGeneratePhraseQueries=true|false` (in schema version 1.4 and later this now defaults to false)
  - `postingsFormat=<name of a postings format>` ⚠️ codec factory], only works if you use a [http://wiki.apache.org/solr/SolrConfigXml#codecFactory that is schema-aware such as SchemaCodecFactory. Please note that the postings formats used in a fieldType definition need to be in any of Solr lib directories. (For example, some useful (but unsupported) postings formats are available in the lucene-codecs JAR.). For detailed instructions on how to configure SimpleTextCodec, see: SimpleTextCodec Example

{{TextField}}s can also support Analyzers with highly configurable Tokenizers and Token Filters.

Field types that store text (`TextField`, `StrField`) support compression of stored contents:

- `compressed=true|false`
- `compressThreshold=<integer>`

⚠️ compression support was removed in 1.4.1. There are (untested) patches for 3.x in https://issues.apache.org/jira/browse/SOLR-752.

`compressThreshold` is the minimum length required for text compression to be invoked. This applies only if `compressed=true`; a common pattern is to set `compressThreshold` on the field type definition, and turn compression on and off in the individual field definitions.

Fields

The `<fields>` section is where you list the individual `<field>` declarations you wish to use in your documents. Each `<field>` has a `name` that you will use to reference it when adding documents or executing searches, and an associated `type` which identifies the name of the fieldtype you wish to use for this field. There are various field options that apply to a field. These can be set in the field type declarations, and can also be overridden at an individual field's declaration. Field names should consist of alphanumeric or underscore characters only and not start with a digit. This is not currently strictly enforced, but other field names will not have first class support from all components and back compatibility is not guaranteed. Names with both leading and trailing underscores (e.g. *version*) are reserved.

Recommended fields

While these fields aren't strictly mandatory (Solr will run if you remove them fully), Bad Things happen in some situations if they aren't defined. We recommend that you leave these fields alone. If you don't use them, there's no appreciable penalty.

- id - Almost all Solr installations have this field defined as the <uniqueKey> (see below).
- *version* ⚠️ SolrCloud] - This field is used for optimistic locking in [http://wiki.apache.org/solr/SolrCloud and it enables Real Time Get. If you remove it you must also remove the transaction logging from solrconfig.xml, see Real Time Get.

Common field options

Common options that fields can have are...

- `default`
  - The default value for this field if none is provided while adding documents
- `indexed=true|false`
  - True if this field should be "indexed". If (and only if) a field is indexed, then it is searchable, sortable, and facetable.
- `stored=true|false`
  - True if the value of the field should be retrievable during a search, or if you're using highlighting or MoreLikeThis.
- `compressed=true|false`
  - True if this field should be stored using gzip compression. (This will only apply if the field type is compressible; among the standard field types, only TextField and StrField are.)
- `compressThreshold=<integer>`
- `multiValued=true|false`
  - True if this field may contain multiple values per document, i.e. if it can appear multiple times in a document
- `omitNorms=true|false`
  - This is arguably an advanced option.
  - Set to true to omit the norms associated with this field (this disables length normalization and index-time boosting for the field, and saves some memory). Only full-text fields or fields that need an index-time boost need norms.
- `termVectors=false|true` <?> Solr 1.1
  - If set, include full term vector info.
  - If enabled, often also used with termPositions="true" and termOffsets="true".
  - To use interactively, requires TermVectorComponent
  - Corresponds to TV button in Luke, and V field attribute.
- `omitTermFreqAndPositions=true|false` ⚠️ Solr1.4
  - If set, omit term freq, positions and payloads from postings for this field. This can be a performance boost for fields that don't require that information and reduces storage space required for the index. Queries that rely on position that are issued on a field with this option fail with an exception. Prior to ⚠️ Solr4.0 the queries would silently fail to find documents.
- `omitPositions=true|false` ⚠️ Solr3.4
  - If set, omits positions, but keeps term frequencies

See also FieldOptionsByUseCase, which discusses how these options should be set in various circumstances. See SolrPerformanceFactors for how different options can affect Solr performance.

Dynamic fields

One of the powerful features of Lucene is that you don't have to pre-define every field when you first create your index. Even though Solr provides strong datatyping for fields, it still preserves that flexibility using "Dynamic Fields". Using `<dynamicField>` declarations, you can create field rules that Solr will use to understand what datatype should be used whenever it is given a field name that is not explicitly defined, but matches a prefix or suffix used in a dynamicField.

For example the following dynamic field declaration tells Solr that whenever it sees a field name ending in "_i" which is not an explicitly defined field, then it should dynamically create an integer field with that name...

```
<dynamicField name="*_i"  type="integer"  indexed="true"  stored="true"/>
```

The glob-like pattern in the name attribute must have a "*" only at the start or the end. Longer patterns will be matched first. if equal size patterns both match, the first appearing in the schema will be used.

Indexing same data in multiple fields

Note that, with textual data, it will often make sense to take what's logically speaking a single field (e.g. product name) and index it into several different Solr fields, each with different field options and/or analyzers.

As an example, if I had a field with a list of authors, such as:

- *Schildt, Herbert; Wolpert, Lewis; Davies, P.*

I might want to index the same data differently in three different fields (perhaps using the Solr copyField directive):

- For searching: Tokenized, case-folded, punctuation-stripped:
  - schildt / herbert / wolpert / lewis / davies / p
- For sorting: Untokenized, case-folded, punctuation-stripped:
  - schildt herbert wolpert lewis davies p
- For faceting: Primary author only, using a `solr.StringField`:

- Schildt, Herbert

(See also SolrFacetingOverview.)

Expert field options

The storage of Lucene term vectors can be triggered using the following field options:

- `termVectors=true|false`
- `termPositions=true|false`
- `termOffsets=true|false`

These options can be used to accelerate highlighting and other ancillary functionality, but impose a substantial cost in terms of index size. They are *not* necessary for typical uses of Solr (phrase queries, etc., do not require these settings to be present).

Miscellaneous Settings

In addition to the `<types>` and `<fields>` sections of the schema, there are several other declarations that can appear in your schema.

The Unique Key Field

The `<uniqueKey>` declaration can be used to inform Solr that there is a field in your index which should be unique for all documents. If a document is added that contains the same value for this field as an existing document, the old document will be deleted.

It is not mandatory for a schema to have a uniqueKey field, but an overwhelming majority of them do. It *shouldn't* matter whether you rename this to something else (and change the <uniqueKey> value), but occasionally it has in the past. We recommend that you just leave this definition alone.

Note that if you have enabled the QueryElevationComponent in solrconfig.xml it requires the schema to have a uniqueKey of type StrField. It cannot be, for example, an int field.

The Default Search Field

The `<defaultSearchField>` is used by Solr when parsing queries to identify which field name should be searched in queries where an explicit field name has not been used. It is preferable to not use or rely on this setting; instead the request handler or query LocalParams for a search should specify the default field(s) to search on. This setting here can be omitted and it is being considered for deprecation.

Default query parser operator

The default operator used by Solr's query parser (SolrQueryParser) can be configured with <solrQueryParser defaultOperator="AND|OR"/>. The default operator is "OR" if unspecified. It is preferable to not use or rely on this setting; instead the request handler or query LocalParams should specify the default operator. This setting here can be omitted and it is being considered for deprecation.

Copy Fields

See Copying Fields in the Apache Solr Reference Guide

Similarity

A (global) `<similarity>` declaration can be used to specify a custom Similarity implementation that you want Solr to use when dealing with your index. A Similarity can be specified either by referring directly to the name of a class with a no-arg constructor...

```
<similarity class="org.apache.lucene.search.similarities.DefaultSimilarity"/>
```

...or by referencing a `SimilarityFactory` implementation, which may take optional init params....

```
<similarity class="solr.DFRSimilarityFactory">
  <str name="basicModel">P</str>
  <str name="afterEffect">L</str>
  <str name="normalization">H2</str>
  <float name="c">7</float>
</similarity>
```

Begining with Solr4.0, Similarity factories such as `SchemaSimilarityFactory` can also support specifying specific Similarity implementations on individual field types...

```
<types>
  <fieldType name="text_dfr" class="solr.TextField">
    <analyzer class="org.apache.lucene.analysis.standard.StandardAnalyzer"/>
    <similarity class="solr.DFRSimilarityFactory">
      <str name="basicModel">I(F)</str>
      <str name="afterEffect">B</str>
      <str name="normalization">H2</str>
    </similarity>
  </fieldType>
  <fieldType name="text_ib" class="solr.TextField">
    <analyzer class="org.apache.lucene.analysis.standard.StandardAnalyzer"/>
    <similarity class="solr.IBSimilarityFactory">
      <str name="distribution">SPL</str>
      <str name="lambda">DF</str>
      <str name="normalization">H2</str>
    </similarity>
  </fieldType>
  ...
</types>
<similarity class="solr.SchemaSimilarityFactory"/>
```

If no (global) `<similarity>` is configured in the schema.xml file, an implicit instance of `DefaultSimilarityFactory` is used.

Poly Field Types

Some FieldTypes can be "poly" field types. A Poly FieldType is one that can potentially create multiple Fields per "declared" field. Some examples include the LatLonType and PointType, both which use multiple indexed fields internally to represent what the user sees as a single value (e.g. "35.9,-79.0"). Another example is CurrencyField which indexes the value and currency symbol separately (e.g. "10,USD").

See the example schema, SpatialSearch and CurrencyField for more info on using these field types.

Schema version attribute in the root node

For the up-to-date documentation, see example example schema shipped with Solr

```
<schema name="example" version="1.5">
  <!-- attribute "name" is the name of this schema and is only used for display purposes.
       version="x.y" is Solr's version number for the schema syntax and
       semantics.  It should not normally be changed by applications.

       1.0: multiValued attribute did not exist, all fields are multiValued
            by nature
       1.1: multiValued attribute introduced, false by default
       1.2: omitTermFreqAndPositions attribute introduced, true by default
            except for text fields.
       1.3: removed optional field compress feature
       1.4: autoGeneratePhraseQueries attribute introduced to drive QueryParser
            behavior when a single string produces multiple tokens.  Defaults
            to off for version >= 1.4
       1.5: omitNorms defaults to true for primitive field types
            (int, float, boolean, string...)
     -->
```

See also Upgrade / Migrate Solr 3.x to Solr 4

TODO

- Perhaps make a DTD for the schema.
- Talk about omitNorms and positionIncrementGap wrt text fields
- check whether defaultSearchField is also used by the DisMaxRequestHandler and not only by the StandardRequestHandler