Apache Solr Reference Guide

About Filters

Like tokenizers, filters consume input and produce a stream of tokens. Filters also derive from `org.apache.lucene.analysis.TokenStream`. Unlike tokenizers, a filter's input is another TokenStream. The job of a filter is usually easier than that of a tokenizer since in most cases a filter looks at each token in the stream sequentially and decides whether to pass it along, replace it or discard it.

A filter may also do more complex analysis by looking ahead to consider multiple tokens at once, although this is less common. One hypothetical use for such a filter might be to normalize state names that would be tokenized as two words. For example, the single token "california" would be replaced with "CA", while the token pair "rhode" followed by "island" would become the single token "RI".

Because filters consume one `TokenStream` and produce a new `TokenStream`, they can be chained one after another indefinitely. Each filter in the chain in turn processes the tokens produced by its predecessor. The order in which you specify the filters is therefore significant. Typically, the most general filtering is done first, and later filtering stages are more specialized.

```
<fieldType name="text" class="solr.TextField">
  <analyzer>
    <tokenizer class="solr.StandardTokenizerFactory"/>
    <filter class="solr.StandardFilterFactory"/>
    <filter class="solr.LowerCaseFilterFactory"/>
    <filter class="solr.EnglishPorterFilterFactory"/>
  </analyzer>
</fieldType>
```

This example starts with Solr's standard tokenizer, which breaks the field's text into tokens. Those tokens then pass through Solr's standard filter, which removes dots from acronyms, and performs a few other common operations. All the tokens are then set to lowercase, which will facilitate case-insensitive matching at query time.

The last filter in the above example is a stemmer filter that uses the Porter stemming algorithm. A stemmer is basically a set of mapping rules that maps the various forms of a word back to the base, or *stem*, word from which they derive. For example, in English the words "hugs", "hugging" and "hugged" are all forms of the stem word "hug". The stemmer will replace all of these terms with "hug", which is what will be indexed. This means that a query for "hug" will match the term "hugged", but not "huge".

Conversely, applying a stemmer to your query terms will allow queries containing non stem terms, like "hugging", to match documents with different variations of the same stem word, such as "hugged". This works because both the indexer and the query will map to the same stem ("hug").

Word stemming is, obviously, very language specific. Solr includes several language-specific stemmers created by the Snowball generator that are based on the Porter stemming algorithm. The generic Snowball Porter Stemmer Filter can be used to configure any of these language stemmers. Solr also includes a convenience wrapper for the English Snowball stemmer. There are also several purpose-

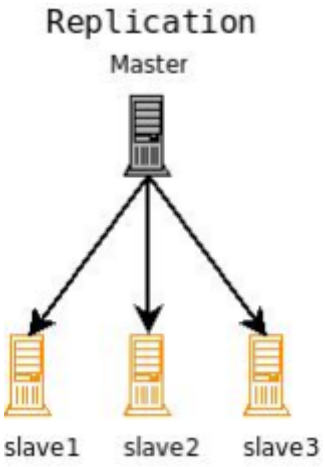
built stemmers for non-English languages. These stemmers are described in Language Analysis.

1. Apache Solr Reference Guide
2. Apache Solr Reference Guide
3. Legacy Scaling and Distribution

Index Replication

Index Replication distributes complete copies of a master index to one or more slave servers. The master server continues to manage updates to the index. All querying is handled by the slaves. This division of labor enables Solr to scale to provide adequate responsiveness to queries against large search volumes.

The figure below shows a Solr configuration using index replication. The master server's index is replicated on the slaves.



A Solr index can be replicated across multiple slave servers, which then processes requests.

Topics covered in this section:

- Index Replication in Solr
- Replication Terminology
- Configuring the ReplicationHandler
- Setting Up a Repeater with the ReplicationHandler
- Commit and Optimize Operations
- Slave Replication
- HTTP API Commands for the ReplicationHandler
- Distribution and Optimization

Index Replication in Solr

Solr includes a Java implementation of index replication that works over HTTP:

- The configuration affecting replication is controlled by a single file, `solrconfig.xml`
- Supports the replication of configuration files as well as index files
- Works across platforms with same configuration
- No reliance on OS-dependent file system features (eg: hard links)
- Tightly integrated with Solr; an admin page offers fine-grained control of each aspect of replication
- The Java-based replication feature is implemented as a request handler. Configuring replication is therefore similar to any normal request handler.

Replication In SolrCloud

Although there is no explicit concept of "master/slave" nodes in a SolrCloud cluster, the `ReplicationHandler` discussed on this page is still used by SolrCloud as needed to support "shard recovery" – but this is done in a peer to peer manner. When using SolrCloud, the `ReplicationHandler` must be available via the `/replication` path. Solr does this implicitly unless overridden explicitly in your `solrconfig.xml`, but If you wish to override the default behavior, make certain that you do not explicitly set any of the "master" or "slave" configuration options mentioned below, or they will interfere with normal SolrCloud operation.

Replication Terminology

The table below defines the key terms associated with Solr replication.

Term	Definition
Index	A Lucene index is a directory of files. These files make up the searchable and returnable data of a Solr Core.
Distribution	The copying of an index from the master server to all slaves. The distribution process takes advantage of Lucene's index file structure.
Inserts and Deletes	As inserts and deletes occur in the index, the directory remains unchanged. Documents are always inserted into newly created files. Documents that are deleted are not removed from the files. They are flagged in the file, deletable, and are not removed from the files until the index is optimized.
Master and Slave	A Solr replication master is a single node which receives all updates initially and keeps everything organized. Solr replication slave nodes receive no updates directly, instead all changes (such as inserts, updates, deletes, etc.) are made against the single master node. Changes made on the master are distributed to all the slave nodes which service all query requests from the clients.
Update	An update is a single change request against a single Solr instance. It may be a request to delete a document, add a new document, change a document, delete all documents matching a query, etc. Updates are handled synchronously within an individual Solr instance.
Optimization	A process that compacts the index and merges segments in order to improve query performance. Optimization should only be run on the master nodes. An optimized index may give query performance gains compared to an index that has become fragmented over a period of time with many updates. Distributing an optimized index requires a much longer time than the distribution of new segments to an un-optimized index.

Term	Definition
Segments	A self contained subset of an index consisting of some documents and data structures related to the inverted index of terms in those documents.
mergeFactor	A parameter that controls the number of segments in an index. For example, when mergeFactor is set to 3, Solr will fill one segment with documents until the limit maxBufferedDocs is met, then it will start a new segment. When the number of segments specified by mergeFactor is reached (in this example, 3) then Solr will merge all the segments into a single index file, then begin writing new documents to a new segment.
Snapshot	A directory containing hard links to the data files of an index. Snapshots are distributed from the master nodes when the slaves pull them, "smart copying" any segments the slave node does not have in snapshot directory that contains the hard links to the most recent index data files.

Configuring the ReplicationHandler

In addition to `ReplicationHandler` configuration options specific to the master/slave roles, there are a few special configuration options that are generally supported (even when using SolrCloud).

- `maxNumberOfBackups` an integer value dictating the maximum number of backups this node will keep on disk as it receives `backup` commands.
- Similar to most other request handlers in Solr you may configure a set of "defaults, invariants, and/or appends" parameters corresponding with any request parameters supported by the `ReplicationHandler` when processing commands.

Configuring the Replication RequestHandler on a Master Server

Before running a replication, you should set the following parameters on initialization of the handler:

Name	Description
replicateAfter	String specifying action after which replication should occur. Valid values are commit, optimize, or startup. There can be multiple values for this parameter. If you use "startup", you need to have a "commit" and/or "optimize" entry also if you want to trigger replication on future commits or optimizes.
backupAfter	String specifying action after which a backup should occur. Valid values are commit, optimize, or startup. There can be multiple values for this parameter. It is not required for replication, it just makes a backup.
maxNumberOfBackups	Integer specifying how many backups to keep. This can be used to delete all but the most recent N backups.
confFiles	The configuration files to replicate, separated by a comma.
commitReserveDuration	If your commits are very frequent and your network is slow, you can tweak this parameter to increase the amount of time taken to download 5Mb from the master to a slave. The default is 10 seconds.

The example below shows a possible 'master' configuration for the `ReplicationHandler`, including a fixed number of backups and an invariant setting for the `maxWriteMBPerSec` request parameter to

prevent slaves from saturating it's network interface

```
<requestHandler name="/replication" class="solr.ReplicationHandler">
  <lst name="master">
    <str name="replicateAfter">optimize</str>
    <str name="backupAfter">optimize</str>
    <str name="confFiles">schema.xml,stopwords.txt,elevate.xml</str>
    <str name="commitReserveDuration">00:00:10</str>
  </lst>
  <int name="maxNumberOfBackups">2</int>
  <lst name="invariants">
    <str name="maxWriteMBPerSec">16</str>
  </lst>
</requestHandler>
```

Replicating `solrconfig.xml`

In the configuration file on the master server, include a line like the following:

```
<str name="confFiles">solrconfig_slave.xml:solrconfig.xml,x.xml,y.xml</str>
```

This ensures that the local configuration `solrconfig_slave.xml` will be saved as `solrconfig.xml` on the slave. All other files will be saved with their original names.

On the master server, the file name of the slave configuration file can be anything, as long as the name is correctly identified in the `confFiles` string; then it will be saved as whatever file name appears after the colon, ":".

Configuring the Replication RequestHandler on a Slave Server

The code below shows how to configure a ReplicationHandler on a slave.

```
<requestHandler name="/replication" class="solr.ReplicationHandler">
  <lst name="slave">

    <!-- fully qualified url for the replication handler of master. It is
         possible to pass on this as a request param for the fetchindex command -->
    <str name="masterUrl">http://remote_host:port/solr/core_name/replication</str>

    <!-- Interval in which the slave should poll master.  Format is HH:mm:ss .
         If this is absent slave does not poll automatically.

         But a fetchindex can be triggered from the admin or the http API -->

    <str name="pollInterval">00:00:20</str>

    <!-- THE FOLLOWING PARAMETERS ARE USUALLY NOT REQUIRED-->

    <!-- To use compression while transferring the index files. The possible
         values are internal|external.  If the value is 'external' make sure
         that your master Solr has the settings to honor the accept-encoding header.
         See here for details: http://wiki.apache.org/solr/SolrHttpCompression
         If it is 'internal' everything will be taken care of automatically.
         USE THIS ONLY IF YOUR BANDWIDTH IS LOW.
         THIS CAN ACTUALLY SLOWDOWN REPLICATION IN A LAN -->
    <str name="compression">internal</str>

    <!-- The following values are used when the slave connects to the master to
         download the index files.  Default values implicitly set as 5000ms and
         10000ms respectively. The user DOES NOT need to specify these unless the
         bandwidth is extremely low or if there is an extremely high latency -->

    <str name="httpConnTimeout">5000</str>
    <str name="httpReadTimeout">10000</str>

    <!-- If HTTP Basic authentication is enabled on the master, then the slave
         can be configured with the following -->

    <str name="httpBasicAuthUser">username</str>
    <str name="httpBasicAuthPassword">password</str>
  </lst>
</requestHandler>
```

Setting Up a Repeater with the ReplicationHandler

A master may be able to serve only so many slaves without affecting performance. Some organizations have deployed slave servers across multiple data centers. If each slave downloads the index from a remote data center, the resulting download may consume too much network bandwidth. To avoid performance degradation in cases like this, you can configure one or more slaves as repeaters. A repeater is simply a node that acts as both a master and a slave.

- To configure a server as a repeater, the definition of the Replication `requestHandler` in the `solrconfig.xml` file must include file lists of use for both masters and slaves.
- Be sure to set the `replicateAfter` parameter to commit, even if `replicateAfter` is set to optimize on the main master. This is because on a repeater (or any slave), a commit is called only after the index is downloaded. The optimize command is never called on slaves.
- Optionally, one can configure the repeater to fetch compressed files from the master through the compression parameter to reduce the index download time.

Here is an example of a ReplicationHandler configuration for a repeater:

```
<requestHandler name="/replication" class="solr.ReplicationHandler">
  <lst name="master">
    <str name="replicateAfter">commit</str>
    <str name="confFiles">schema.xml,stopwords.txt,synonyms.txt</str>
  </lst>
  <lst name="slave">
    <str name="masterUrl">
http://master.solr.company.com:8983/solr/core_name/replication</str>
    <str name="pollInterval">00:00:60</str>
  </lst>
</requestHandler>
```

Commit and Optimize Operations

When a commit or optimize operation is performed on the master, the RequestHandler reads the list of file names which are associated with each commit point. This relies on the `replicateAfter` parameter in the configuration to decide which types of events should trigger replication.

Setting on the Master	Description
commit	Triggers replication whenever a commit is performed on the master index.
optimize	Triggers replication whenever the master index is optimized.
startup	Triggers replication whenever the master index starts up.

The replicateAfter parameter can accept multiple arguments. For example:

```
<str name="replicateAfter">startup</str>
<str name="replicateAfter">commit</str>
<str name="replicateAfter">optimize</str>
```

Slave Replication

The master is totally unaware of the slaves. The slave continuously keeps polling the master (depending on the `pollInterval` parameter) to check the current index version of the master. If the slave finds out that the master has a newer version of the index it initiates a replication process. The steps are as follows:

- The slave issues a `filelist` command to get the list of the files. This command returns the names of the files as well as some metadata (for example, size, a lastmodified timestamp, an alias if any).
- The slave checks with its own index if it has any of those files in the local index. It then runs the filecontent command to download the missing files. This uses a custom format (akin to the HTTP chunked encoding) to download the full content or a part of each file. If the connection breaks in between, the download resumes from the point it failed. At any point, the slave tries 5 times before giving up a replication altogether.
- The files are downloaded into a temp directory, so that if either the slave or the master crashes during the download process, no files will be corrupted. Instead, the current replication will simply abort.
- After the download completes, all the new files are moved to the live index directory and the file's timestamp is same as its counterpart on the master.
- A commit command is issued on the slave by the Slave's ReplicationHandler and the new index is loaded.

Replicating Configuration Files

To replicate configuration files, list them using using the `confFiles` parameter. Only files found in the `conf` directory of the master's Solr instance will be replicated.

Solr replicates configuration files only when the index itself is replicated. That means even if a configuration file is changed on the master, that file will be replicated only after there is a new commit/optimize on master's index.

Unlike the index files, where the timestamp is good enough to figure out if they are identical, configuration files are compared against their checksum. The `schema.xml` files (on master and slave) are judged to be identical if their checksums are identical.

As a precaution when replicating configuration files, Solr copies configuration files to a temporary directory before moving them into their ultimate location in the conf directory. The old configuration files are then renamed and kept in the same `conf/` directory. The ReplicationHandler does not automatically clean up these old files.

If a replication involved downloading of at least one configuration file, the ReplicationHandler issues a core-reload command instead of a commit command.

Resolving Corruption Issues on Slave Servers

If documents are added to the slave, then the slave is no longer in sync with its master. However, the slave will not undertake any action to put itself in sync, until the master has new index data. When a commit operation takes place on the master, the index version of the master becomes different from that of the slave. The slave then fetches the list of files and finds that some of the files present on the master are also present in the local index but with different sizes and timestamps. This means that the master and slave have incompatible indexes. To correct this problem, the slave then copies all the index files from master to a new index directory and asks the core to load the fresh index from the new directory.

HTTP API Commands for the ReplicationHandler

You can use the HTTP commands below to control the ReplicationHandler's operations.

Command	Description
http://*master_host:port*/solr/*core_name*/replication?command=enablereplication	Enables replication on the master for all its slaves.
http://*master_host:port*/solr/*core_name*/replication?command=disablereplication	Disables replication on the master for all its slaves.
http://*host:port*/solr/*core_name*/replication?command=indexversion	Returns the version of the latest replicatable index on the specified master or slave.
http://*slave_host:port*/solr/*core_name*/replication?command=fetchindex	Forces the specified slave to fetch a copy of the index from its master. If you like, you can pass an extra attribute such as masterUrl or compression (or any other parameter which is specified in the `<lst name="slave">` tag) to do a one time replication from a master. This obviates the need for hard-coding the master in the slave.
http://*slave_host:port*/solr/*core_name*/replication?command=abortfetch	Aborts copying an index from a master to the specified slave.
http://*slave_host:port*/solr/*core_name*/replication?command=enablepoll	Enables the specified slave to poll for changes on the master.
http://*slave_host:port*/solr/*core_name*/replication?command=disablepoll	Disables the specified slave from polling for changes on the master.
http://*slave_host:port*/solr/*core_name*/replication?command=details	Retrieves configuration details and current status.
http://*host:port*/solr/*core_name*/replication?command=filelist&indexversion=<*index-version-number*>	Retrieves a list of Lucene files present in the specified host's index. You can discover the version number of the index by running the `indexversion` command.

Command	Description
http://*master_host:port*/solr/*core_name*/replication ?command=backup	Creates a backup on master if there are committed index data in the server; otherwise, does nothing. This command is useful for making periodic backups. supported request parameters: * `numberToKeep:` request parameter can be used with the backup command unless the `maxNumberOfBackups` initialization parameter has been specified on the handler – in which case `maxNumberOfBackups` is always used and attempts to use the `numberToKeep` request parameter will cause an error. * `name` : (optional) Backup name . The snapshot will be created in a directory called snapshot.<name> within the data directory of the core . By default the name is generated using date in `yyyyMMddHHmmssSSS` format. If `location` parameter is passed , that would be used instead of the data directory * ` location ` : Backup location
http://*master_host:port* /solr/ *core_name*/replication?command=deletebackup	Delete any backup created using the `backup` command . request parameters: * name: The name of the snapshot . A snapshot with the name snapshot.<name> must exist .If not, an error is thrown * location: Location where the snapshot is created

Distribution and Optimization

Optimizing an index is not something most users should generally worry about - but in particular users should be aware of the impacts of optimizing an index when using the `ReplicationHandler`.

The time required to optimize a master index can vary dramatically. A small index may be optimized in minutes. A very large index may take hours. The variables include the size of the index and the speed of the hardware.

Distributing a newly optimized index may take only a few minutes or up to an hour or more, again depending on the size of the index and the performance capabilities of network connections and disks. During optimization the machine is under load and does not process queries very well. Given a schedule of updates being driven a few times an hour to the slaves, we cannot run an optimize with every committed snapshot.

Copying an optimized index means that the **entire** index will need to be transferred during the next snappull. This is a large expense, but not nearly as huge as running the optimize everywhere. Consider this example: on a three-slave one-master configuration, distributing a newly-optimized index takes approximately 80 seconds *total*. Rolling the change across a tier would require approximately ten minutes per machine (or machine group). If this optimize were rolled across the query tier, and if each slave node being optimized were disabled and not receiving queries, a rollout would take at least twenty minutes and potentially as long as an hour and a half. Additionally, the files would need to be synchronized so that the *following* the optimize, snappull would not think that the independently optimized files were different in any way. This would also leave the door open to independent corruption of indexes instead of each being a perfect copy of the master.

Optimizing on the master allows for a straight-forward optimization operation. No query slaves need to be taken out of service. The optimized index can be distributed in the background as queries are being normally serviced. The optimization can occur at any time convenient to the application providing index updates.

While optimizing may have some benefits in some situations, a rapidly changing index will not retain those benefits for long, and since optimization is an intensive process, it may be better to consider other options, such as lowering the merge factor (discussed in the section on Index Configuration).

1. Apache Solr Reference Guide
2. Apache Solr Reference Guide
3. Understanding Analyzers, Tokenizers, and Filters

Language Analysis

This section contains information about tokenizers and filters related to character set conversion or for use with specific languages. For the European languages, tokenization is fairly straightforward. Tokens are delimited by white space and/or a relatively small set of punctuation characters. In other

languages the tokenization rules are often not so simple. Some European languages may require special tokenization rules as well, such as rules for decompounding German words.

For information about language detection at index time, see Detecting Languages During Indexing.

Topics discussed in this section:

- KeywordMarkerFilterFactory
- StemmerOverrideFilterFactory
- Dictionary Compound Word Token Filter
- Unicode Collation
- ASCII Folding Filter
- Language-Specific Factories
- Related Topics

KeywordMarkerFilterFactory

Protects words from being modified by stemmers. A customized protected word list may be specified with the "protected" attribute in the schema. Any words in the protected word list will not be modified by any stemmer in Solr.

A sample Solr `protwords.txt` with comments can be found in the `sample_techproducts_configs` config set directory:

```
<fieldtype name="myfieldtype" class="solr.TextField">
  <analyzer>
    <tokenizer class="solr.WhitespaceTokenizerFactory"/>
    <filter class="solr.KeywordMarkerFilterFactory" protected="protwords.txt" />
    <filter class="solr.PorterStemFilterFactory" />
  </analyzer>
</fieldtype>
```

StemmerOverrideFilterFactory

Overrides stemming algorithms by applying a custom mapping, then protecting these terms from being modified by stemmers.

A customized mapping of words to stems, in a tab-separated file, can be specified to the "dictionary" attribute in the schema. Words in this mapping will be stemmed to the stems from the file, and will not be further changed by any stemmer.

A sample stemdict.txt with comments can be found in the Source Repository.

```
<fieldtype name="myfieldtype" class="solr.TextField">
  <analyzer>
    <tokenizer class="solr.WhitespaceTokenizerFactory"/>
    <filter class="solr.StemmerOverrideFilterFactory" dictionary="stemdict.txt" />
    <filter class="solr.PorterStemFilterFactory" />
  </analyzer>
</fieldtype>
```

Dictionary Compound Word Token Filter

This filter splits, or *decompounds*, compound words into individual words using a dictionary of the component words. Each input token is passed through unchanged. If it can also be decompounded into subwords, each subword is also added to the stream at the same logical position.

Compound words are most commonly found in Germanic languages.

Factory class

`solr.DictionaryCompoundWordTokenFilterFactory`

Arguments

`dictionary`

(required) The path of a file that contains a list of simple words, one per line. Blank lines and lines that begin with "#" are ignored. This path may be an absolute path, or path relative to the Solr config directory.

`minWordSize`

(integer, default 5) Any token shorter than this is not decompounded.

`minSubwordSize`

(integer, default 2) Subwords shorter than this are not emitted as tokens.

`maxSubwordSize`

(integer, default 15) Subwords longer than this are not emitted as tokens.

`onlyLongestMatch`

(true/false) If true (the default), only the longest matching subwords will generate new tokens.

Example

Assume that `germanwords.txt` contains at least the following words: `dumm kopf donau dampf schiff`

```
<analyzer>
  <tokenizer class="solr.StandardTokenizerFactory"/>
  <filter class="solr.DictionaryCompoundWordTokenFilterFactory" dictionary=
"germanwords.txt"/>
</analyzer>
```

In

"Donaudampfschiff dummkopf"

Tokenizer to Filter

"Donaudampfschiff"(1), "dummkopf"(2),

Out

"Donaudampfschiff"(1), "Donau"(1), "dampf"(1), "schiff"(1), "dummkopf"(2), "dumm"(2), "kopf"(2)

Unicode Collation

Unicode Collation is a language-sensitive method of sorting text that can also be used for advanced search purposes.

Unicode Collation in Solr is fast, because all the work is done at index time.

Rather than specifying an analyzer within `<fieldtype ... class="solr.TextField">`, the `solr.CollationField` and `solr.ICUCollationField` field type classes provide this functionality. `solr.ICUCollationField`, which is backed by the ICU4J library, provides more flexible configuration, has more locales, is significantly faster, and requires less memory and less index space, since its keys are smaller than those produced by the JDK implementation that backs `solr.CollationField`.

`solr.ICUCollationField` is included in the Solr `analysis-extras` contrib - see `solr/contrib/analysis-extras/README.txt` for instructions on which jars you need to add to your `SOLR_HOME/lib` in order to use it.

`solr.ICUCollationField` and `solr.CollationField` fields can be created in two ways:

- Based upon a system collator associated with a Locale.
- Based upon a tailored `RuleBasedCollator` ruleset.

ICUCollationField Arguments

Arguments for `solr.ICUCollationField` are specified as attributes within the `<fieldtype>` element.

Using a System collator:

`locale`
: (required) RFC 3066 locale ID. See the ICU locale explorer for a list of supported locales.

`strength`
: Valid values are `primary`, `secondary`, `tertiary`, `quaternary`, or `identical`. See Comparison Levels in ICU Collation Concepts for more information.

`decomposition`
: Valid values are `no` or `canonical`. See Normalization in ICU Collation Concepts for more information.

Using a Tailored ruleset:

`custom`
: (required) Path to a UTF-8 text file containing rules supported by the ICU ` RuleBasedCollator `

`strength`
: Valid values are `primary`, `secondary`, `tertiary`, `quaternary`, or `identical`. See Comparison Levels in ICU Collation Concepts for more information.

`decomposition`

Valid values are `no` or `canonical`. See Normalization in ICU Collation Concepts for more information.

Expert options:

`alternate`

Valid values are `shifted` or `non-ignorable`. Can be used to ignore punctuation/whitespace.

`caseLevel`

(true/false) If true, in combination with `strength="primary"`, accents are ignored but case is taken into account. The default is false. See CaseLevel in ICU Collation Concepts for more information.

`caseFirst`

Valid values are `lower` or `upper`. Useful to control which is sorted first when case is not ignored.

`numeric`

(true/false) If true, digits are sorted according to numeric value, e.g. foobar-9 sorts before foobar-10. The default is false.

`variableTop`

Single character or contraction. Controls what is variable for `alternate`

Sorting Text for a Specific Language

In this example, text is sorted according to the default German rules provided by ICU4J.

Locales are typically defined as a combination of language and country, but you can specify just the language if you want. For example, if you specify "de" as the language, you will get sorting that works well for the German language. If you specify "de" as the language and "CH" as the country, you will get German sorting specifically tailored for Switzerland.

```
<!-- Define a field type for German collation -->
<fieldType name="collatedGERMAN" class="solr.ICUCollationField"
           locale="de"
           strength="primary" />
...
<!-- Define a field to store the German collated manufacturer names. -->
<field name="manuGERMAN" type="collatedGERMAN" indexed="false" stored="false" docValues=
"true"/>
...
<!-- Copy the text to this field. We could create French, English, Spanish versions too,
     and sort differently for different users! -->
<copyField source="manu" dest="manuGERMAN"/>
```

In the example above, we defined the strength as "primary". The strength of the collation determines how strict the sort order will be, but it also depends upon the language. For example, in English, "primary" strength ignores differences in case and accents.

Another example:

```
<fieldType name="polishCaseInsensitive" class="solr.ICUCollationField"
           locale="pl_PL"
           strength="secondary" />
...
<field name="city" type="text_general" indexed="true" stored="true"/>
...
<field name="city_sort" type="polishCaseInsensitive" indexed="true" stored="false"/>
...
<copyField source="city" dest="city_sort"/>
```

The type will be used for the fields where the data contains Polish text. The "secondary" strength will ignore case differences, but, unlike "primary" strength, a letter with diacritic(s) will be sorted differently from the same base letter without diacritics.

An example using the "city_sort" field to sort:

```
q=*:*&fl=city&sort=city_sort+asc
```

Sorting Text for Multiple Languages

There are two approaches to supporting multiple languages: if there is a small list of languages you wish to support, consider defining collated fields for each language and using `copyField`. However, adding a large number of sort fields can increase disk and indexing costs. An alternative approach is to use the Unicode `default` collator.

The Unicode `default` or `ROOT` locale has rules that are designed to work well for most languages. To use the `default` locale, simply define the locale as the empty string. This Unicode default sort is still significantly more advanced than the standard Solr sort.

```
<fieldType name="collatedROOT" class="solr.ICUCollationField"
           locale=""
           strength="primary" />
```

Sorting Text with Custom Rules

You can define your own set of sorting rules. It's easiest to take existing rules that are close to what you want and customize them.

In the example below, we create a custom rule set for German called DIN 5007-2. This rule set treats umlauts in German differently: it treats ö as equivalent to oe, ä as equivalent to ae, and ü as equivalent to ue. For more information, see the ICU RuleBasedCollator javadocs.

This example shows how to create a custom rule set for `solr.ICUCollationField` and dump it to a file:

```
// get the default rules for Germany
// these are called DIN 5007-1 sorting
RuleBasedCollator baseCollator = (RuleBasedCollator) Collator.getInstance(new ULocale("
de", "DE"));

// define some tailorings, to make it DIN 5007-2 sorting.
// For example, this makes ö equivalent to oe
String DIN5007_2_tailorings =
    "& ae , a\u0308 & AE , A\u0308"+
    "& oe , o\u0308 & OE , O\u0308"+
    "& ue , u\u0308 & UE , u\u0308";

// concatenate the default rules to the tailorings, and dump it to a String
RuleBasedCollator tailoredCollator = new RuleBasedCollator(baseCollator.getRules() +
DIN5007_2_tailorings);
String tailoredRules = tailoredCollator.getRules();

// write these to a file, be sure to use UTF-8 encoding!!!
FileOutputStream os = new FileOutputStream(new File("/solr_home/conf/customRules.dat"));
IOUtils.write(tailoredRules, os, "UTF-8");
```

This rule set can now be used for custom collation in Solr:

```
---
<fieldType name="collatedCUSTOM" class="solr.ICUCollationField"
           custom="customRules.dat"
           strength="primary" />
---
```

JDK Collation

As mentioned above, ICU Unicode Collation is better in several ways than JDK Collation, but if you cannot use ICU4J for some reason, you can use `solr.CollationField`.

The principles of JDK Collation are the same as those of ICU Collation; you just specify `language`, `country` and `variant` arguments instead of the combined `locale` argument.

CollationField Arguments

Arguments for `solr.CollationField` are specified as attributes within the `<fieldtype>` element.

Using a System collator (see Oracle's list of locales supported in Java 7):

`language`
: (required) ISO-639 language code

`country`
: ISO-3166 country code

`variant`
: Vendor or browser-specific code

`strength`
: Valid values are `primary`, `secondary`, `tertiary` or `identical`. See Oracle Java 7 Collator javadocs for more information.

`decomposition`
: Valid values are `no`, `canonical`, or `full`. See Oracle Java 7 Collator javadocs for more information.

Using a Tailored ruleset:

`custom`
: (required) Path to a UTF-8 text file containing rules supported by the ` JDK RuleBasedCollator `

`strength`
: Valid values are `primary`, `secondary`, `tertiary` or `identical`. See Oracle Java 7 Collator javadocs for more information.

`decomposition`
: Valid values are `no`, `canonical`, or `full`. See Oracle Java 7 Collator javadocs for more information.

Example solr.CollationField

```
<fieldType name="collatedGERMAN" class="solr.CollationField"
           language="de"
           country="DE"
           strength="primary" /> <!-- ignore Umlauts and letter case when sorting -->
...
<field name="manuGERMAN" type="collatedGERMAN" indexed="false" stored="false" docValues=
"true" />
...
<copyField source="manu" dest="manuGERMAN"/>
```

ASCII Folding Filter

This filter converts alphabetic, numeric, and symbolic Unicode characters which are not in the first 127 ASCII characters (the "Basic Latin" Unicode block) into their ASCII equivalents, if one exists. Only those characters with reasonable ASCII alternatives are converted:

This can increase recall by causing more matches. On the other hand, it can reduce precision because language-specific character differences may be lost.

Factory class

`solr.ASCIIFoldingFilterFactory`

Arguments

None

Example

```
<analyzer>
  <tokenizer class="solr.StandardTokenizerFactory"/>
  <filter class="solr.ASCIIFoldingFilterFactory"/>
</analyzer>
```

In

"Björn Ångström"

Tokenizer to Filter

"Björn", "Ångström"

Out

"Bjorn", "Angstrom"

Language-Specific Factories

These factories are each designed to work with specific languages.

Arabic

Solr provides support for the Light-10 (PDF) stemming algorithm, and Lucene includes an example stopword list.

This algorithm defines both character normalization and stemming, so these are split into two filters to provide more flexibility.

Factory classes

`solr.ArabicStemFilterFactory`, `solr.ArabicNormalizationFilterFactory`

Arguments

None

Example

```
<analyzer>
  <tokenizer class="solr.StandardTokenizerFactory"/>
  <filter class="solr.ArabicNormalizationFilterFactory"/>
  <filter class="solr.ArabicStemFilterFactory"/>
</analyzer>
```

Brazilian Portuguese

This is a Java filter written specifically for stemming the Brazilian dialect of the Portuguese language. It uses the Lucene class `org.apache.lucene.analysis.br.BrazilianStemmer`. Although that stemmer can be configured to use a list of protected words (which should not be stemmed), this factory does not accept any arguments to specify such a list.

Factory class

`solr.BrazilianStemFilterFactory`

Arguments

None

Example

```
<analyzer type="index">
  <tokenizer class="solr.StandardTokenizerFactory"/>
  <filter class="solr.BrazilianStemFilterFactory"/>
</analyzer>
```

In

"praia praias"

Tokenizer to Filter

"praia", "praias"

Out

"pra", "pra"

Bulgarian

Solr includes a light stemmer for Bulgarian, following this algorithm (PDF), and Lucene includes an example stopword list.

Factory class

`solr.BulgarianStemFilterFactory`

Arguments

None

Example

```
<analyzer>
  <tokenizer class="solr.StandardTokenizerFactory"/>
  <filter class="solr.LowerCaseFilterFactory"/>
  <filter class="solr.BulgarianStemFilterFactory"/>
</analyzer>
```

Catalan

Solr can stem Catalan using the Snowball Porter Stemmer with an argument of `language="Catalan"`. Solr includes a set of contractions for Catalan, which can be stripped using `solr.ElisionFilterFactory`.

Factory class

`solr.SnowballPorterFilterFactory`

Arguments

`language`

(required) stemmer language, "Catalan" in this case

Example

```
<analyzer>
  <tokenizer class="solr.StandardTokenizerFactory"/>
  <filter class="solr.LowerCaseFilterFactory"/>
  <filter class="solr.ElisionFilterFactory"
          articles="lang/contractions_ca.txt"/>
  <filter class="solr.SnowballPorterFilterFactory" language="Catalan" />
</analyzer>
```

In

"llengües llengua"

Tokenizer to Filter

"llengües"(1) "llengua"(2),

Out

"llengu"(1), "llengu"(2)

Chinese

Chinese Tokenizer

The Chinese Tokenizer is deprecated as of Solr 3.4. Use the `solr.StandardTokenizerFactory` instead.

Factory class

`solr.ChineseTokenizerFactory`

Arguments

None

Example

```
<analyzer type="index">
  <tokenizer class="solr.ChineseTokenizerFactory"/>
</analyzer>
```

Chinese Filter Factory

The Chinese Filter Factory is deprecated as of Solr 3.4. Use the `solr.StopFilterFactory` instead.

Factory class

`solr.ChineseFilterFactory`

Arguments

None

Example

```
<analyzer type="index">
  <tokenizer class="solr.StandardTokenizerFactory"/>
  <filter class="solr.ChineseFilterFactory"/>
</analyzer>
```

Simplified Chinese

For Simplified Chinese, Solr provides support for Chinese sentence and word segmentation with the `solr.HMMChineseTokenizerFactory` in the `analysis-extras` contrib module. This component includes a large dictionary and segments Chinese text into words with the Hidden Markov Model. To use this filter, see `solr/contrib/analysis-extras/README.txt` for instructions on which jars you need to add to your `solr_home/lib`.

Factory class

`solr.HMMChineseTokenizerFactory`

Arguments

None

Examples

To use the default setup with fallback to English Porter stemmer for English words, use:

```
<analyzer class="org.apache.lucene.analysis.cn.smart.SmartChineseAnalyzer"/>
```

Or to configure your own analysis setup, use the `solr.HMMChineseTokenizerFactory` along with your custom filter setup.

```
<analyzer>
  <tokenizer class="solr.HMMChineseTokenizerFactory"/>
  <filter class="solr.StopFilterFactory
          words="org/apache/lucene/analysis/cn/smart/stopwords.txt"/>
  <filter class="solr.PorterStemFilterFactory"/>
</analyzer>
```

CJK

This tokenizer breaks Chinese, Japanese and Korean language text into tokens. These are not whitespace delimited languages. The tokens generated by this tokenizer are "doubles", overlapping pairs of CJK characters found in the field text.

Factory class

`solr.CJKTokenizerFactory`

Arguments

None

Example

```
<analyzer type="index">
  <tokenizer class="solr.CJKTokenizerFactory"/>
</analyzer>
```

Czech

Solr includes a light stemmer for Czech, following this algorithm, and Lucene includes an example stopword list.

Factory class

`solr.CzechStemFilterFactory`

Arguments

None

Example

```
<analyzer>
  <tokenizer class="solr.StandardTokenizerFactory"/>
  <filter class="solr.LowerCaseFilterFactory"/>
  <filter class="solr.CzechStemFilterFactory"/>
<analyzer>
```

In

"prezidenští, prezidenta, prezidentského"

Tokenizer to Filter

"prezidenští", "prezidenta", "prezidentského"

Out

"preziden", "preziden", "preziden"

Danish

Solr can stem Danish using the Snowball Porter Stemmer with an argument of `language="Danish"`.

Also relevant are the Scandinavian normalization filters.

Factory class

`solr.SnowballPorterFilterFactory`

Arguments

`language`

(required) stemmer language, "Danish" in this case

Example

```
<analyzer>
  <tokenizer class="solr.StandardTokenizerFactory"/>
  <filter class="solr.LowerCaseFilterFactory"/>
  <filter class="solr.SnowballPorterFilterFactory" language="Danish" />
</analyzer>
```

In

"undersøg undersøgelse"

Tokenizer to Filter

"undersøg"(1) "undersøgelse"(2),

Out

"undersøg"(1), "undersøg"(2)

Dutch

Solr can stem Dutch using the Snowball Porter Stemmer with an argument of `language="Dutch"`.

Factory class

`solr.SnowballPorterFilterFactory`

Arguments

`language`

(required) stemmer language, "Dutch" in this case

Example

```
<analyzer type="index">
  <tokenizer class="solr.StandardTokenizerFactory"/>
  <filter class="solr.LowerCaseFilterFactory"/>
  <filter class="solr.SnowballPorterFilterFactory" language="Dutch"/>
</analyzer>
```

In

"kanaal kanalen"

Tokenizer to Filter

"kanaal", "kanalen"

Out

"kanal", "kanal"

Finnish

Solr includes support for stemming Finnish, and Lucene includes an example stopword list.

Factory class

`solr.FinnishLightStemFilterFactory`

Arguments

None

Example

```
<analyzer type="index">
  <tokenizer class="solr.StandardTokenizerFactory"/>
  <filter class="solr.FinnishLightStemFilterFactory"/>
</analyzer>
```

In

"kala kalat"

Tokenizer to Filter

"kala", "kalat"

Out

"kala", "kala"

French

Elision Filter

Removes article elisions from a token stream. This filter can be useful for languages such as French, Catalan, Italian, and Irish.

Factory class

`solr.ElisionFilterFactory`

Arguments

`articles`

The pathname of a file that contains a list of articles, one per line, to be stripped. Articles are words such as "le", which are commonly abbreviated, such as in *l'avion* (the plane). This file should include the abbreviated form, which precedes the apostrophe. In this case, simply "*l*". If no `articles` attribute is specified, a default set of French articles is used.

`ignoreCase`

(boolean) If true, the filter ignores the case of words when comparing them to the common word file. Defaults to `false`.

Example

```
<analyzer>
  <tokenizer class="solr.StandardTokenizerFactory"/>
  <filter class="solr.ElisionFilterFactory"
          ignoreCase="true"
          articles="lang/contractions_fr.txt"/>
</analyzer>
```

In

"L'histoire d'art"

Tokenizer to Filter

"L'histoire", "d'art"

Out

"histoire", "art"

French Light Stem Filter

Solr includes three stemmers for French: one in the `solr.SnowballPorterFilterFactory`, a lighter stemmer called `solr.FrenchLightStemFilterFactory`, and an even less aggressive stemmer called `solr.FrenchMinimalStemFilterFactory`. Lucene includes an example stopword list.

Factory classes

`solr.FrenchLightStemFilterFactory`, `solr.FrenchMinimalStemFilterFactory`

Arguments

None

Examples

```
<analyzer>
  <tokenizer class="solr.StandardTokenizerFactory"/>
  <filter class="solr.LowerCaseFilterFactory"/>
  <filter class="solr.ElisionFilterFactory"
          articles="lang/contractions_fr.txt"/>
  <filter class="solr.FrenchLightStemFilterFactory"/>
</analyzer>
```

```
<analyzer>
  <tokenizer class="solr.StandardTokenizerFactory"/>
  <filter class="solr.LowerCaseFilterFactory"/>
  <filter class="solr.ElisionFilterFactory"
          articles="lang/contractions_fr.txt"/>
  <filter class="solr.FrenchMinimalStemFilterFactory"/>
</analyzer>
```

In

"le chat, les chats"

Tokenizer to Filter

"le", "chat", "les", "chats"

Out

"le", "chat", "le", "chat"

Galician

Solr includes a stemmer for Galician following this algorithm, and Lucene includes an example stopword list.

Factory class

`solr.GalicianStemFilterFactory`

Arguments

None

Example

```
<analyzer>
  <tokenizer class="solr.StandardTokenizerFactory"/>
  <filter class="solr.LowerCaseFilterFactory"/>
  <filter class="solr.GalicianStemFilterFactory"/>
</analyzer>
```

In

"felizmente Luzes"

Tokenizer to Filter

"felizmente", "luzes"

Out

"feliz", "luz"

German

Solr includes four stemmers for German:

- one in the `solr.SnowballPorterFilterFactory language="German"`,
- a stemmer called `solr.GermanStemFilterFactory`,
- a lighter stemmer called `solr.GermanLightStemFilterFactory`,
- and an even less aggressive stemmer called `solr.GermanMinimalStemFilterFactory`.

Lucene includes an example stopword list.

Factory classes::: `solr.GermanStemFilterFactory`, `solr.LightGermanStemFilterFactory`, `solr.MinimalGermanStemFilterFactory`

Arguments

None

Examples

```
<analyzer type="index">
  <tokenizer class="solr.StandardTokenizerFactory "/>
  <filter class="solr.GermanStemFilterFactory"/>
</analyzer>
```

```
<analyzer type="index">
  <tokenizer class="solr.StandardTokenizerFactory"/>
  <filter class="solr.GermanLightStemFilterFactory"/>
</analyzer>
```

```
<analyzer type="index">
  <tokenizer class="solr.StandardTokenizerFactory "/>
  <filter class="solr.GermanMinimalStemFilterFactory"/>
</analyzer>
```

In

"haus häuser"

Tokenizer to Filter

"haus", "häuser"

Out

"haus", "haus"

Greek

This filter converts uppercase letters in the Greek character set to the equivalent lowercase character.

Factory class

`solr.GreekLowerCaseFilterFactory`

Arguments

None

Use of custom charsets is not supported as of Solr 3.1. If you need to index text in these encodings, please use Java's character set conversion facilities (`InputStreamReader`, and so on.) during I/O, so that Lucene can analyze this text as Unicode instead.

Example

```
<analyzer type="index">
  <tokenizer class="solr.StandardTokenizerFactory"/>
  <filter class="solr.GreekLowerCaseFilterFactory"/>
</analyzer>
```

Hindi

Solr includes support for stemming Hindi following this algorithm (PDF), support for common spelling differences through the `solr.HindiNormalizationFilterFactory`, support for encoding differences through the `solr.IndicNormalizationFilterFactory` following this algorithm, and Lucene includes an example stopword list.

Factory classes

`solr.IndicNormalizationFilterFactory`, `solr.HindiNormalizationFilterFactory`, `solr.HindiStemFilterFactory`

Arguments

None

Example

```
<analyzer type="index">
  <tokenizer class="solr.StandardTokenizerFactory"/>
  <filter class="solr.IndicNormalizationFilterFactory"/>
  <filter class="solr.HindiNormalizationFilterFactory"/>
  <filter class="solr.HindiStemFilterFactory"/>
</analyzer>
```

Indonesian

Solr includes support for stemming Indonesian (Bahasa Indonesia) following this algorithm (PDF), and Lucene includes an example stopword list.

Factory class

`solr.IndonesianStemFilterFactory`

Arguments

None

Example

```
<analyzer>
  <tokenizer class="solr.StandardTokenizerFactory"/>
  <filter class="solr.LowerCaseFilterFactory"/>
  <filter class="solr.IndonesianStemFilterFactory" stemDerivational="true" />
</analyzer>
```

In

"sebagai sebagainya"

Tokenizer to Filter

"sebagai", "sebagainya"

Out

"bagai", "bagai"

Italian

Solr includes two stemmers for Italian: one in the `solr.SnowballPorterFilterFactory language="Italian"`, and a lighter stemmer called `solr.ItalianLightStemFilterFactory`. Lucene includes an example stopword list.

Factory class

`solr.ItalianStemFilterFactory`

Arguments

None

Example

```
<analyzer>
  <tokenizer class="solr.StandardTokenizerFactory"/>
  <filter class="solr.LowerCaseFilterFactory"/>
  <filter class="solr.ElisionFilterFactory"
          articles="lang/contractions_it.txt"/>
  <filter class="solr.ItalianLightStemFilterFactory"/>
</analyzer>
```

In

"propaga propagare propagamento"

Tokenizer to Filter

"propaga", "propagare", "propagamento"

Out

"propag", "propag", "propag"

Irish

Solr can stem Irish using the Snowball Porter Stemmer with an argument of `language="Irish"`. Solr includes `solr.IrishLowerCaseFilterFactory`, which can handle Irish-specific constructs. Solr also includes a set of contractions for Irish which can be stripped using `solr.ElisionFilterFactory`.

Factory class

`solr.SnowballPorterFilterFactory`

Arguments

`language`

(required) stemmer language, "Irish" in this case

Example

```
<analyzer>
  <tokenizer class="solr.StandardTokenizerFactory"/>
  <filter class="solr.ElisionFilterFactory"
          articles="lang/contractions_ga.txt"/>
  <filter class="solr.IrishLowerCaseFilterFactory"/>
  <filter class="solr.SnowballPorterFilterFactory" language="Irish" />
</analyzer>
```

In

"siopadóireacht síceapatacha b’fhearr m’athair"

Tokenizer to Filter

"siopadóireacht", "síceapatacha", "b’fhearr", "m’athair"

Out

"siopadóir", "síceapaite", "fearr", "athair"

Japanese

Solr includes support for analyzing Japanese, via the Lucene Kuromoji morphological analyzer, which includes several analysis components - more details on each below:

- `JapaneseIterationMarkCharFilter` normalizes Japanese horizontal iteration marks (odoriji) to their expanded form.
- `JapaneseTokenizer` tokenizes Japanese using morphological analysis, and annotates each term with part-of-speech, base form (a.k.a. lemma), reading and pronunciation.
- `JapaneseBaseFormFilter` replaces original terms with their base forms (a.k.a. lemmas).
- `JapanesePartOfSpeechStopFilter` removes terms that have one of the configured parts-of-speech.
- `JapaneseKatakanaStemFilter` normalizes common katakana spelling variations ending in a long sound character (U+30FC) by removing the long sound character.

Also useful for Japanese analysis, from lucene-analyzers-common:

- `CJKWidthFilter` folds fullwidth ASCII variants into the equivalent Basic Latin forms, and folds halfwidth Katakana variants into their equivalent fullwidth forms.

Japanese Iteration Mark CharFilter

Normalizes horizontal Japanese iteration marks (odoriji) to their expanded form. Vertical iteration marks are not supported.

Factory class

`JapaneseIterationMarkCharFilterFactory`

Arguments

`normalizeKanji`: set to `false` to not normalize kanji iteration marks (default is `true`)

` normalizeKana `: set to `false` to not normalize kana iteration marks (default is `true`)

Japanese Tokenizer

Tokenizer for Japanese that uses morphological analysis, and annotates each term with part-of-speech, base form (a.k.a. lemma), reading and pronunciation.

`JapaneseTokenizer` has a `search` mode (the default) that does segmentation useful for search: a heuristic is used to segment compound terms into their constituent parts while also keeping the original compound terms as synonyms.

Factory class

`solr.JapaneseTokenizerFactory`

Arguments

`mode`

Use `search` mode to get a noun-decompounding effect useful for search. `search` mode improves segmentation for search at the expense of part-of-speech accuracy. Valid values for `mode` are:

- `normal`: default segmentation
- `search`: segmentation useful for search (extra compound splitting)
- `extended`: search mode plus unigramming of unknown words (experimental)

For some applications it might be good to use `search` mode for indexing and `normal` mode for queries to increase precision and prevent parts of compounds from being matched and highlighted.

`userDictionary`

filename for a user dictionary, which allows overriding the statistical model with your own entries for segmentation, part-of-speech tags and readings without a need to specify weights. See `lang/userdict_ja.txt` for a sample user dictionary file.

`userDictionaryEncoding`

user dictionary encoding (default is UTF-8)

`discardPunctuation`

set to `false` to keep punctuation, `true` to discard (the default)

Japanese Base Form Filter

Replaces original terms' text with the corresponding base form (lemma). (`JapaneseTokenizer` annotates each term with its base form.)

Factory class

`JapaneseBaseFormFilterFactory`

Arguments

None

Japanese Part Of Speech Stop Filter

Removes terms with one of the configured parts-of-speech. `JapaneseTokenizer` annotates terms with parts-of-speech.

Factory class

`JapanesePartOfSpeechStopFilterFactory`

Arguments

`tags`
: filename for a list of parts-of-speech for which to remove terms; see `conf/lang/stoptags_ja.txt` in the `sample_techproducts_config` config set for an example.

`enablePositionIncrements`
: if `luceneMatchVersion` is `4.3` or earlier and `enablePositionIncrements="false"`, no position holes will be left by this filter when it removes tokens. **This argument is invalid if `luceneMatchVersion` is `5.0` or later.**

Japanese Katakana Stem Filter

Normalizes common katakana spelling variations ending in a long sound character (U+30FC) by removing the long sound character.

`CJKWidthFilterFactory` should be specified prior to this filter to normalize half-width katakana to full-width.

Factory class
: `JapaneseKatakanaStemFilterFactory`

Arguments
`minimumLength`
: terms below this length will not be stemmed. Default is 4, value must be 2 or more.

CJK Width Filter

Folds fullwidth ASCII variants into the equivalent Basic Latin forms, and folds halfwidth Katakana variants into their equivalent fullwidth forms.

Factory class
: `CJKWidthFilterFactory`

Arguments
: None

Example

```
<fieldType name="text_ja" positionIncrementGap="100" autoGeneratePhraseQueries="false">
  <analyzer>
    <!-- Uncomment if you need to handle iteration marks: -->
    <!-- <charFilter class="solr.JapaneseIterationMarkCharFilterFactory" /> -->
    <tokenizer class="solr.JapaneseTokenizerFactory" mode="search" userDictionary=
"lang/userdict_ja.txt"/>
    <filter class="solr.JapaneseBaseFormFilterFactory"/>
    <filter class="solr.JapanesePartOfSpeechStopFilterFactory" tags=
"lang/stoptags_ja.txt"/>
    <filter class="solr.CJKWidthFilterFactory"/>
    <filter class="solr.StopFilterFactory" ignoreCase="true" words=
"lang/stopwords_ja.txt"/>
    <filter class="solr.JapaneseKatakanaStemFilterFactory" minimumLength="4"/>
    <filter class="solr.LowerCaseFilterFactory"/>
  </analyzer>
</fieldType>
```

Hebrew, Lao, Myanmar, Khmer

Lucene provides support, in addition to UAX#29 word break rules, for Hebrew's use of the double and single quote characters, and for segmenting Lao, Myanmar, and Khmer into syllables with the `solr.ICUTokenizerFactory` in the `analysis-extras` contrib module. To use this tokenizer, see `solr/contrib/analysis-extras/README.txt` for instructions on which jars you need to add to your `solr_home/lib`.

See the ICUTokenizer for more information.

Latvian

Solr includes support for stemming Latvian, and Lucene includes an example stopword list.

Factory class

`solr.LatvianStemFilterFactory`

Arguments

None

Example

```
<fieldType name="text_lvstem" class="solr.TextField" positionIncrementGap="100">
  <analyzer>
    <tokenizer class="solr.StandardTokenizerFactory"/>
    <filter class="solr.LowerCaseFilterFactory"/>
    <filter class="solr.LatvianStemFilterFactory"/>
  </analyzer>
</fieldType>
```

In

"tirgiem tirgus"

Tokenizer to Filter

"tirgiem", "tirgus"

Out

"tirg", "tirg"

Norwegian

Solr includes two classes for stemming Norwegian, `NorwegianLightStemFilterFactory` and `NorwegianMinimalStemFilterFactory`. Lucene includes an example stopword list.

Another option is to use the Snowball Porter Stemmer with an argument of language="Norwegian".

Also relevant are the Scandinavian normalization filters.

Norwegian Light Stemmer

The `NorwegianLightStemFilterFactory` requires a "two-pass" sort for the -dom and -het endings. This means that in the first pass the word "kristendom" is stemmed to "kristen", and then all the general rules apply so it will be further stemmed to "krist". The effect of this is that "kristen," "kristendom," "kristendommen," and "kristendommens" will all be stemmed to "krist."

The second pass is to pick up -dom and -het endings. Consider this example:

One pass		Two passes	
Before	**After**	**Before**	**After**
forlegen	forleg	forlegen	forleg
forlegenhet	forlegen	forlegenhet	forleg
forlegenheten	forlegen	forlegenheten	forleg
forlegenhetens	forlegen	forlegenhetens	forleg
firkantet	firkant	firkantet	firkant

One pass		Two passes	
firkantethet	firkantet	firkantethet	firkant
firkantetheten	firkantet	firkantetheten	firkant

Factory class

`solr.NorwegianLightStemFilterFactory`

Arguments

`variant`

Choose the Norwegian language variant to use. Valid values are:

- `nb:` Bokmål (default)
- `nn:` Nynorsk
- `no:` both

Example

```
<fieldType name="text_no" class="solr.TextField" positionIncrementGap="100">
  <analyzer>
    <tokenizer class="solr.StandardTokenizerFactory"/>
    <filter class="solr.LowerCaseFilterFactory"/>
    <filter class="solr.StopFilterFactory" ignoreCase="true" words=
"lang/stopwords_no.txt" format="snowball"/>
    <filter class="solr.NorwegianLightStemFilterFactory"/>
  </analyzer>
</fieldType>
```

In

"Forelskelsen"

Tokenizer to Filter

"forelskelsen"

Out

"forelske"

Norwegian Minimal Stemmer

The `NorwegianMinimalStemFilterFactory` stems plural forms of Norwegian nouns only.

Factory class

`solr.NorwegianMinimalStemFilterFactory`

Arguments

`variant`

Choose the Norwegian language variant to use. Valid values are:

- `nb:` Bokmål (default)
- `nn:` Nynorsk
- `no:` both

Example

```
<fieldType name="text_no" class="solr.TextField" positionIncrementGap="100">
  <analyzer>
    <tokenizer class="solr.StandardTokenizerFactory"/>
    <filter class="solr.LowerCaseFilterFactory"/>
    <filter class="solr.StopFilterFactory" ignoreCase="true" words=
"lang/stopwords_no.txt" format="snowball"/>
    <filter class="solr.NorwegianMinimalStemFilterFactory"/>
  </analyzer>
</fieldType>
```

In

"Bilens"

Tokenizer to Filter

"bilens"

Out

"bil"

Persian

Persian Filter Factories

Solr includes support for normalizing Persian, and Lucene includes an example stopword list.

Factory class

`solr.PersianNormalizationFilterFactory`

Arguments

None

Example

```
<analyzer>
  <tokenizer class="solr.StandardTokenizerFactory"/>
  <filter class="solr.ArabicNormalizationFilterFactory"/>
  <filter class="solr.PersianNormalizationFilterFactory">
</analyzer>
```

Polish

Solr provides support for Polish stemming with the `solr.StempelPolishStemFilterFactory`, and `solr.MorphologikFilterFactory` for lemmatization, in the `contrib/analysis-extras` module. The `solr.StempelPolishStemFilterFactory` component includes an algorithmic stemmer with tables for Polish. To use either of these filters, see `solr/contrib/analysis-extras/README.txt` for instructions on which jars you need to add to your `solr_home/lib`.

Factory class

`solr.StempelPolishStemFilterFactory` and `solr.MorfologikFilterFactory`

Arguments

None

Example

```
<analyzer>
  <tokenizer class="solr.StandardTokenizerFactory"/>
  <filter class="solr.LowerCaseFilterFactory"/>
  <filter class="solr.StempelPolishStemFilterFactory"/>
</analyzer>
```

```
<analyzer>
  <tokenizer class="solr.StandardTokenizerFactory"/>
  <filter class="solr.LowerCaseFilterFactory"/>
  <filter class="solr.MorfologikFilterFactory" dictionary-resource="pl"/>
</analyzer>
```

In

""studenta studenci"

Tokenizer to Filter

"studenta", "studenci"

Out

"student", "student"

More information about the Stempel stemmer is available in the Lucene javadocs.

The Morfologik `dictionary-resource` param value is a constant specifying which dictionary to choose. The dictionary resource must be named `morfologik/dictionaries/{dictionaryResource}.dict` and have an associated `.info` metadata file. See the Morfologik project for details.

Portuguese

Solr includes four stemmers for Portuguese: one in the `solr.SnowballPorterFilterFactory`, an alternative stemmer called `solr.PortugueseStemFilterFactory`, a lighter stemmer called `solr.PortugueseLightStemFilterFactory`, and an even less aggressive stemmer called `solr.PortugueseMinimalStemFilterFactory`. Lucene includes an example stopword list.

Factory classes

`solr.PortugueseStemFilterFactory`, `solr.PortugueseLightStemFilterFactory`, `solr.PortugueseMinimalStemFilterFactory`

Arguments

None

Examples

```
<analyzer>
  <tokenizer class="solr.StandardTokenizerFactory"/>
  <filter class="solr.LowerCaseFilterFactory"/>
  <filter class="solr.PortugueseStemFilterFactory"/>
</analyzer>
```

```
<analyzer>
  <tokenizer class="solr.StandardTokenizerFactory"/>
  <filter class="solr.LowerCaseFilterFactory"/>
  <filter class="solr.PortugueseLightStemFilterFactory"/>
</analyzer>
```

```
<analyzer>
  <tokenizer class="solr.StandardTokenizerFactory"/>
  <filter class="solr.LowerCaseFilterFactory"/>
  <filter class="solr.PortugueseMinimalStemFilterFactory"/>
</analyzer>
```

In

"praia praias"

Tokenizer to Filter

"praia", "praias"

Out

"pra", "pra"

Romanian

Solr can stem Romanian using the Snowball Porter Stemmer with an argument of `language="Romanian"`.

Factory class

`solr.SnowballPorterFilterFactory`

Arguments

`language`

(required) stemmer language, "Romanian" in this case

Example

```
<analyzer>
  <tokenizer class="solr.StandardTokenizerFactory"/>
  <filter class="solr.LowerCaseFilterFactory"/>
  <filter class="solr.SnowballPorterFilterFactory" language="Romanian" />
</analyzer>
```

Russian

Russian Stem Filter

Solr includes two stemmers for Russian: one in the `solr.SnowballPorterFilterFactory language="Russian"`, and a lighter stemmer called `solr.RussianLightStemFilterFactory`. Lucene includes an example stopword list.

Factory class

`solr.RussianLightStemFilterFactory`

Arguments

None

Use of custom charsets is no longer supported as of Solr 3.4. If you need to index text in these encodings, please use Java's character set conversion facilities (`InputStreamReader`, and so on.) during I/O, so that Lucene can analyze this text as Unicode instead.

Example

```
<analyzer type="index">
  <tokenizer class="solr.StandardTokenizerFactory"/>
  <filter class="solr.LowerCaseFilterFactory"/>
  <filter class="solr.RussianLightStemFilterFactory"/>
</analyzer>
```

Scandinavian

Scandinavian is a language group spanning three languages Norwegian, Swedish and Danish which are very similar.

Swedish å,ä,ö are in fact the same letters as Norwegian and Danish å,æ,ø and thus interchangeable when used between these languages. They are however folded differently when people type them on a keyboard lacking these characters.

In that situation almost all Swedish people use a, a, o instead of å, ä, ö. Norwegians and Danes on the other hand usually type aa, ae and oe instead of å, æ and ø. Some do however use a, a, o, oo, ao and sometimes permutations of everything above.

There are two filters for helping with normalization between Scandinavian languages: one is `solr.ScandinavianNormalizationFilterFactory` trying to preserve the special characters (æäöå) and another `solr.ScandinavianFoldingFilterFactory` which folds these to the more broad ø/ö→o etc.

See also each language section for other relevant filters.

Scandinavian Normalization Filter

This filter normalize use of the interchangeable Scandinavian characters æÆäÄöÖøØ and folded variants (aa, ao, ae, oe and oo) by transforming them to åÅæÆøØ.

It's a semantically less destructive solution than `ScandinavianFoldingFilter`, most useful when a person with a Norwegian or Danish keyboard queries a Swedish index and vice versa. This filter does **not** perform the common Swedish folds of å and ä to a nor ö to o.

Factory class

`solr.ScandinavianNormalizationFilterFactory`

Arguments

None

Example

```
<analyzer>
  <tokenizer class="solr.StandardTokenizerFactory"/>
  <filter class="solr.LowerCaseFilterFactory"/>
  <filter class="solr.ScandinavianNormalizationFilterFactory"/>
</analyzer>
```

In

"blåbærsyltetøj blåbärsyltetöj blaabaarsyltetoej blabarsyltetoj"

Tokenizer to Filter

"blåbærsyltetøj", "blåbärsyltetöj", "blaabaersyltetoej", "blabarsyltetoj"

Out

"blåbærsyltetøj", "blåbærsyltetøj", "blåbærsyltetøj", "blabarsyltetoj"

Scandinavian Folding Filter

This filter folds Scandinavian characters åÅäæÄÆ→a and öÖøØ→o. It also discriminate against use of double vowels aa, ae, ao, oe and oo, leaving just the first one.

It's is a semantically more destructive solution than `ScandinavianNormalizationFilter`, but can in addition help with matching raksmorgas as räksmörgås.

Factory class

`solr.ScandinavianFoldingFilterFactory`

Arguments

None

Example

```
<analyzer>
  <tokenizer class="solr.StandardTokenizerFactory"/>
  <filter class="solr.LowerCaseFilterFactory"/>
  <filter class="solr.ScandinavianFoldingFilterFactory"/>
</analyzer>
```

In

"blåbærsyltetøj blåbärsyltetöj blaabaarsyltetoej blabarsyltetoj"

Tokenizer to Filter

"blåbærsyltetøj", "blåbärsyltetöj", "blaabaersyltetoej", "blabarsyltetoj"

Out

"blabarsyltetoj", "blabarsyltetoj", "blabarsyltetoj", "blabarsyltetoj"

Serbian

Serbian Normalization Filter

Solr includes a filter that normalizes Serbian Cyrillic and Latin characters to "bald" Latin. Cyrillic characters are first converted to Latin; then, Latin characters have their diacritics removed, with the exception of “đ” which is converted to “dj”. Note that this filter expects lowercased input.

Factory class

`solr.SerbianNormalizationFilterFactory`

Arguments

None

Example

```
<analyzer>
  <tokenizer class="solr.StandardTokenizerFactory"/>
  <filter class="solr.LowerCaseFilterFactory"/>
  <filter class="solr.SerbianNormalizationFilterFactory"/>
</analyzer>
```

In

"Đura (Ђура) српски"

Tokenizer to Filter

"đura" "ђура" "српски"

Out

"djura", "djura", "srpski"

Spanish

Solr includes two stemmers for Spanish: one in the `solr.SnowballPorterFilterFactory language="Spanish"`, and a lighter stemmer called `solr.SpanishLightStemFilterFactory`. Lucene includes an example stopword list.

Factory class

`solr.SpanishStemFilterFactory`

Arguments

None

Example

```
<analyzer>
  <tokenizer class="solr.StandardTokenizerFactory"/>
  <filter class="solr.LowerCaseFilterFactory"/>
  <filter class="solr.SpanishLightStemFilterFactory"/>
</analyzer>
```

In

"torear toreara torearlo"

Tokenizer to Filter

"torear", "toreara", "torearlo"

Out

"tor", "tor", "tor"

Swedish

Swedish Stem Filter

Solr includes two stemmers for Swedish: one in the `solr.SnowballPorterFilterFactory language="Swedish"`, and a lighter stemmer called `solr.SwedishLightStemFilterFactory`. Lucene includes an example stopword list.

Also relevant are the Scandinavian normalization filters.

Factory class

`solr.SwedishStemFilterFactory`

Arguments

None

Example

```
<analyzer>
  <tokenizer class="solr.StandardTokenizerFactory"/>
  <filter class="solr.LowerCaseFilterFactory"/>
  <filter class="solr.SwedishLightStemFilterFactory"/>
</analyzer>
```

In

"kloke klokhet klokheten"

Tokenizer to Filter

"kloke", "klokhet", "klokheten"

Out

"klok", "klok", "klok"

Thai

This filter converts sequences of Thai characters into individual Thai words. Unlike European languages, Thai does not use whitespace to delimit words.

Factory class

`solr.ThaiTokenizerFactory`

Arguments

None

Example

```
<analyzer type="index">
  <tokenizer class="solr.ThaiTokenizerFactory"/>
  <filter class="solr.LowerCaseFilterFactory"/>
</analyzer>
```

Turkish

Solr includes support for stemming Turkish through the `solr.SnowballPorterFilterFactory`; support for case-insensitive search through the `solr.TurkishLowerCaseFilterFactory`; support for stripping apostrophes and following suffixes through `solr.ApostropheFilterFactory` (see Role of Apostrophes in Turkish Information Retrieval); support for a form of stemming that truncating tokens at a configurable maximum length through the solr.TruncateTokenFilterFactory (see Information Retrieval on Turkish Texts); and Lucene includes an example stopword list.

Factory class

`solr.TurkishLowerCaseFilterFactory`

Arguments

None

Example

```
<analyzer>
  <tokenizer class="solr.StandardTokenizerFactory"/>
  <filter class="solr.ApostropheFilterFactory"/>
  <filter class="solr.TurkishLowerCaseFilterFactory"/>
  <filter class="solr.SnowballPorterFilterFactory" language="Turkish"/>
</analyzer>
```

Another example, illustrating diacritics-insensitive search:

```
<analyzer>
  <tokenizer class="solr.StandardTokenizerFactory"/>
  <filter class="solr.ApostropheFilterFactory"/>
  <filter class="solr.TurkishLowerCaseFilterFactory"/>
  <filter class="solr.ASCIIFoldingFilterFactory" preserveOriginal="true"/>
  <filter class="solr.KeywordRepeatFilterFactory"/>
  <filter class="solr.TruncateTokenFilterFactory" prefixLength="5"/>
  <filter class="solr.RemoveDuplicatesTokenFilterFactory"/>
</analyzer>
```