AnalyzersTokenizersTokenFilters

This page exists for the Solr Community to share Tips, Tricks, and Advice about Analyzers, Tokenizers and Filters.

Reference material previously located on this page has been migrated to the Official Solr Reference Guide. If you need help, please consult the Reference Guide for the version of Solr you are using. The sections below will point to corresponding sections of the Reference Guide for each specific feature.

If you'd like to share information about how you use this feature, please add it to this page.
/* cwikimigrated */

Analyzers, Tokenizers, and Token Filters

For a complete list of what Tokenizers and TokenFilters come out of the box, please consult the Lucene javadocs, Solr javadocs, and Automatically generated list at solr-start.com. Please look at analyzer-*. There are quite a few. if you have any tips/tricks you'd like to mention about using any of these classes, please add them below.

Note: For a good background on Lucene Analysis, it's recommended that you read the following sections in Lucene In Action:

- 1.5.3 : Analyzer
- Chapter 4.0 through 4.7 at least

- solr.ICUFoldingFilterFactory
- solr.ICUTransformFilterFactory

High Level Concepts

Stemming

Individual Solr stemmers are documented in the Solr Reference Guide section Filter Descriptions.

Analyzers

Analyzers are documented in the Solr Reference Guide section Analyzers.

Char Filters

CharFilters are documented in the Solr Reference Guide section CharFilterFactories.

Tokenizers

Tokenizers are documented in the Solr Reference Guide section Tokenizers.

Token Filters

Token Filters are documented in the Solr Reference Guide section Filter Descriptions.

Specifying an Analyzer in the schema

If you want to use custom CharFilters, Tokenizers or TokenFilters, you'll need to write a very simple factory that subclasses BaseTokenizerFactory or BaseTokenFilterFactory, something like this...

```
public class MyCustomFilterFactory extends BaseTokenFilterFactory {
  public TokenStream create(TokenStream input) {
    return new MyCustomFilter(input);
  }
}
```

When To use a CharFilter vs a TokenFilter

There are several pairs of CharFilters and TokenFilters that have related (ie: MappingCharFilter and !ASCIIFoldingFilter) or nearly identical functionality (ie: PatternReplaceCharFilterFactory and PatternReplaceFilterFactory) and it may not always be obvious which is the best choice.

The ultimate decision depends largely on what Tokenizer you are using, and whether you need to "out smart" it by preprocessing the stream of characters.

For example, maybe you have a tokenizer such as StandardTokenizer and you are pretty happy with how it works overall, but you want to customize how some specific characters behave.

In such a situation you could modify the rules and re-build your own tokenizer with javacc, but perhaps its easier to simply map some of the characters before tokenization with a CharFilter.

Notes On Specific Factories

CharFilterFactories

solr.MappingCharFilterFactory

Documentation at MappingCharFilterFactory.

solr.PatternReplaceCharFilterFactory

Documentation at PatternReplaceCharFilterFactory.

solr.HTMLStripCharFilterFactory

Documentation at HTMLStripCharFilterFactory.

TokenizerFactories

Solr provides the following TokenizerFactories (Tokenizers and TokenFilters):

solr.KeywordTokenizerFactory

Documentation at Keyword Tokenizer.

solr.LetterTokenizerFactory

Documentation at Letter Tokenizer.

solr.WhitespaceTokenizerFactory

Documentation at White Space Tokenizer.

solr.LowerCaseTokenizerFactory

Documentation at Lower Case Tokenizer.

solr.StandardTokenizerFactory

Documentation at Standard Tokenizer.

Solr Version	Behavior
pre-3.1	Some token types are number, alphanumeric, email, acronym, URL, etc. —

Example: `"I.B.M. cat's can't" ==> ACRONYM: "I.B.M.", APOSTROPHE:"cat's", APOSTROPHE:"can't"|`

⚠️ [	Word boundary rules from http://unicode.org/reports/tr29/#Word_Boundaries] Token types: <ALPHANUM>, <NUM>, <SOUTHEAST_ASIAN>, <IDEOGRAPHIC>, and <HIRAGANA>

Example: `"I.B.M. 8.5 can't!!!" ==> ALPHANUM: "I.B.M.", NUM:"8.5", ALPHANUM:"can't"|`

solr.ClassicTokenizerFactory

Documentation at Classic Tokenizer.

solr.UAX29URLEmailTokenizerFactory

Documentation at UAX29 URL Email Tokenizer.

solr.PatternTokenizerFactory

Documentation at Regular Expression Pattern Tokenizer.

solr.ICUTokenizerFactory

Documentation at ICU Tokenizer.

TokenFilterFactories

Overall documented at Filter Descriptions.

solr.ClassicFilterFactory

Documentation at Classic Filter.

solr.ApostropheFilterFactory

Creates `org.apache.lucene.analysis.tr.ApostropheFilter`.

Strips all characters after an apostrophe (including the apostrophe itself).

- Example: `"Türkiye'de", "2003'te" ==> "Türkiye", "2003"`.

solr.LowerCaseFilterFactory

Documentation at Lower Case Filter.

solr.TypeTokenFilterFactory

Documented at Type Token Filter.

solr.TrimFilterFactory

Documented at Trim Filter.

solr.TruncateTokenFilterFactory

⚠️ Solr4.8

Creates `org.apache.lucene.analysis.miscellaneous.TruncateTokenFilter`.

A token filter for truncating the terms into a specific length.

```
<filter class="solr.TruncateTokenFilterFactory" prefixLength="5"/>
```

- Example: `"abcdefg", "1234567" ==> "abcde", "12345"`.

solr.PatternCaptureGroupFilterFactory

⚠️ Solr4.4

Emits tokens for each capture group in a regular expression

For example, the following definition will tokenize the input text of "http://www.foo.com/index" into "http://www.foo.com" and "www.foo.com".

```
   <fieldType name="url_base" class="solr.TextField" positionIncrementGap="100">
     <analyzer>
       <tokenizer class="solr.KeywordTokenizerFactory"/>
       <filter class="solr.PatternCaptureGroupFilterFactory" pattern="(https?://([a-zA-Z\-_0-9.]+))"
preserve_original="false"/>
     </analyzer>
   </fieldType>
```

If none of the patterns match, or if preserve_original is true, the original token will also be emitted.

solr.PatternReplaceFilterFactory

Documentation at Pattern Replace Filter.

solr.StopFilterFactory

Documentation at Stop Filter.

solr.CommonGramsFilterFactory

Documentation at Common Grams Filter.

solr.EdgeNGramFilterFactory

Documentation at Edge N-Gram Filter.

This FilterFactory is very useful in matching prefix substrings (or suffix substrings if side="back") of particular terms in the index during query time. Edge n-gram analysis can be performed at either index or query time (or both), but typically it is more useful, as shown in this example, to generate the n-grams at index time with all of the n-grams indexed at the same position. At query time the query term can be matched directly without any n-gram analysis. Unlike wildcards, n-gram query terms can be used within quoted phrases.

```
<fieldType name="text_general_edge_ngram" class="solr.TextField" positionIncrementGap="100">
   <analyzer type="index">
      <tokenizer class="solr.LowerCaseTokenizerFactory"/>
      <filter class="solr.EdgeNGramFilterFactory" minGramSize="2" maxGramSize="15"/>
   </analyzer>
   <analyzer type="query">
      <tokenizer class="solr.LowerCaseTokenizerFactory"/>
   </analyzer>
</fieldType>
```

solr.KeepWordFilterFactory

Documentation at Keep Word Filter.

solr.WordDelimiterFilterFactory

Documentation at Word Delimiter Filter.

One use for WordDelimiterFilter is to help match words with different delimiters. One way of doing so is to specify `generateWordParts="1"` `catenateWords="1"` in the analyzer used for indexing, and `generateWordParts="1"` in the analyzer used for querying. Given that the current StandardTokenizer immediately removes many intra-word delimiters, it is recommended that this filter be used after a tokenizer that leaves them in place (such as WhitespaceTokenizer).

```
    <fieldtype name="subword" class="solr.TextField">
      <analyzer type="query">
          <tokenizer class="solr.WhitespaceTokenizerFactory"/>
          <filter class="solr.WordDelimiterFilterFactory"
                generateWordParts="1"
                generateNumberParts="1"
                catenateWords="0"
                catenateNumbers="0"
                catenateAll="0"
                preserveOriginal="1"
                />
          <filter class="solr.LowerCaseFilterFactory"/>
          <filter class="solr.StopFilterFactory"/>
          <filter class="solr.PorterStemFilterFactory"/>
      </analyzer>
      <analyzer type="index">
          <tokenizer class="solr.WhitespaceTokenizerFactory"/>
          <filter class="solr.WordDelimiterFilterFactory"
                generateWordParts="1"
                generateNumberParts="1"
                catenateWords="1"
                catenateNumbers="1"
                catenateAll="0"
                preserveOriginal="1"
                />
          <filter class="solr.LowerCaseFilterFactory"/>
          <filter class="solr.StopFilterFactory"/>
          <filter class="solr.PorterStemFilterFactory"/>
      </analyzer>
    </fieldtype>
```

In some cases you might want to adjust how WordDelimiterFilter splits on a per-character basis. To do this, you can supply a configuration file with the "types" attribute that specifies custom character categories. An example file is in subversion here. This is especially useful to add "hashtag or currency" searches.

solr.SynonymFilterFactory

Documentation at Synonym Filter.

Keep in mind that while the SynonymFilter will happily work with synonyms containing multiple words (ie: "`sea biscuit, sea biscit, seabiscuit`") The recommended approach for dealing with synonyms like this, is to expand the synonym when indexing. This is because there are two potential issues that can arrise at query time:

1. The Lucene QueryParser tokenizes on white space before giving any text to the Analyzer, so if a person searches for the words `sea biscit` the analyzer will be given the words "sea" and "biscit" seperately, and will not know that they match a synonym.
2. Phrase searching (ie: "`sea biscit`") will cause the QueryParser to pass the entire string to the analyzer, but if the SynonymFilter is configured to expand the synonyms, then when the QueryParser gets the resulting list of tokens back from the Analyzer, it will construct a MultiPhraseQuery that will not have the desired effect. This is because of the limited mechanism available for the Analyzer to indicate that two terms occupy the same position: there is no way to indicate that a "phrase" occupies the same position as a term. For our example the resulting MultiPhraseQuery would be "`(sea | sea | seabiscuit) (biscuit | biscit)`" which would not match the simple case of "seabiscuit" occuring in a document

Even when you aren't worried about multi-word synonyms, idf differences still make index time synonyms a good idea. Consider the following scenario:

- An index with a "text" field, which at query time uses the SynonymFilter with the synonym `TV, Televesion` and `expand="true"`
- Many thousands of documents containing the term "text:TV"
- A few hundred documents containing the term "text:Television"

A query for `text:TV` will expand into `(text:TV text:Television)` and the lower docFreq for `text:Television` will give the documents that match "Television" a much higher score then docs that match "TV" comparably – which may be somewhat counter intuitive to the client. Index time expansion (or reduction) will result in the same idf for all documents regardless of which term the original text contained.

solr.RemoveDuplicatesTokenFilterFactory

Documentation at Remove Duplicates Token Filter.

solr.ISOLatin1AccentFilterFactory

Replaces accented characters in the ISO Latin 1 character set (ISO-8859-1) by their unaccented equivalent. In Solr 3.x, this filter is deprecated. This filter does not exist at all in 4.x versions. Use ASCIIFoldingFilterFactory instead.

solr.ASCIIFoldingFilterFactory

Documentation at ASCII Folding Filter.

solr.PhoneticFilterFactory

Documentation at Phonetic Filter.

solr.DoubleMetaphoneFilterFactor

Documentation at Double Metaphone Filter.

solr.BeiderMorseFilterFactory

Documentation at Beider-Morse Filter.

This is especially useful for Central European and Eastern European surnames. For example, one can use this filter factory to find documents that contain the surname "Kracovsky" when the original search term was "Crakowski", or vice versa. For more information, check out the paper about Beider-Morse Phonetic Matching (BMPM) at http://stevemorse.org/phonetics/bmpm.htm.

solr.ShingleFilterFactory

Documentation at Shingle Filter.

solr.PositionFilterFactory

This filter was deprecated and removed from Lucene in 5.0

⚠️ Solr1.4

Creates org.apache.lucene.analysis.position.PositionFilter.

A PositionFilter manipulates the position of tokens in the stream.

Set the positionIncrement of all tokens to the "positionIncrement", except the first return token which retains its original positionIncrement value.

arg	value
positionIncrement	default 0

```
  <filter class="solr.PositionFilterFactory" />
```

PositionFilter can be used with a query Analyzer to prevent expensive Phrase and MultiPhraseQueries. When QueryParser parses a query, it first divides text on whitespace, and then Analyzes each whitespace token. Some TokenStreams such as StandardTokenizer or WordDelimiterFilter may divide one of these whitespace-separate tokens into multiple tokens.

The QueryParser will turn "multiple tokens" into a Phrase or MultiPhraseQuery, but "multiple tokens at the same position with only a position count of 1" is treated as a special case. You can use PositionFilter at the end of your QueryAnalyzer to force any subsequent tokens after the first one to have a position increment of zero, to trigger this case.

For example, by default a query of "Wi-Fi" with StandardTokenizer will create a PhraseQuery:

```
field:"Wi Fi"
```

If you instead wrap the StandardTokenizer with PositionFilter, the "Fi" will have a position increment of zero, creating a BooleanQuery:

```
field:Wi field:Fi
```

Another example is when exact matching hits are wanted for *any* shingle within the query. (This was done at http://sesam.no to replace three proprietary 'FAST Query-Matching servers' with two open sourced Solr indexes, background reading in sesat and on the mailing list). It was needed that in the query all words and shingles to be placed at the same position, so that all shingles to be treated as synonyms of each other.

With only the ShingleFilter the shingles generated are synonyms only to the first term in each shingle group. For example the query "abcd efgh ijkl" results in a query like:

- ("abcd" "abcd efgh" "abcd efgh ijkl") ("efgh" efgh ijkl") ("ijkl")

where "abcd efgh" and "abcd efgh ijkl" are synonyms of "abcd", and "efgh ijkl" is a synonym of "efgh".

ShingleFilter does not offer a way to alter this behaviour.

Using the PositionFilter in combination makes it possible to make all shingles synonyms of each other. Such a configuration could look like:

```
   <fieldType name="shingleString" class="solr.TextField" positionIncrementGap="100" omitNorms="true">
      <analyzer type="index">
        <tokenizer class="solr.KeywordTokenizerFactory"/>
      </analyzer>
      <analyzer type="query">
        <tokenizer class="solr.WhitespaceTokenizerFactory"/>
        <filter class="solr.ShingleFilterFactory" outputUnigrams="true" outputUnigramIfNoNgram="true"
maxShingleSize="99"/>
        <filter class="solr.PositionFilterFactory" />
      </analyzer>
    </fieldType>
```

solr.ReversedWildcardFilterFactory

Documentation at Reversed Wildcard Filter.

Add this filter to the index analyzer, but not the query analyzer. The standard Solr query parser (SolrQuerySyntax) will use this to reverse wildcard and prefix queries to improve performance (for example, translating myfield:*foo into myfield:oof*). To avoid collisions and false matches, reversed tokens are indexed with a prefix that should not otherwise appear in indexed text.

solr.CollationKeyFilterFactory

Documentation at Collation Key Filter. Also discussed in the section Unicode Collation.

solr.ICUCollationKeyFilterFactory

See Unicode Collation.

This filter works like CollationKeyFilterFactory, except it uses ICU for collation. This makes smaller and faster sort keys, and it supports more locales. See UnicodeCollation for some more information, the same concepts apply.

The only configuration difference is that locales should be specified to this filter with RFC 3066 locale IDs.

```
<fieldType name="icu_sort_en" class="solr.TextField">
  <analyzer>
    <tokenizer class="solr.KeywordTokenizerFactory"/>
    <filter class="solr.ICUCollationKeyFilterFactory" locale="en" strength="primary"/>
  </analyzer>
</fieldType>
```

Note: 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

solr.ICUNormalizer2FilterFactory

Documentation at ICU Normalizer 2 Filter.

solr.ICUFoldingFilterFactory

Documentation at ICU Folding Filter.

solr.ICUTransformFilterFactory

Documentation at ICU Transform Filter.