```
<?xml version="1.0" encoding="UTF-8" ?>
<!--
 Licensed to the Apache Software Foundation (ASF) under one or more
 contributor license agreements.  See the NOTICE file distributed with
 this work for additional information regarding copyright ownership.
 The ASF licenses this file to You under the Apache License, Version 2.0
 (the "License"); you may not use this file except in compliance with
 the License.  You may obtain a copy of the License at

     http://www.apache.org/licenses/LICENSE-2.0

 Unless required by applicable law or agreed to in writing, software
 distributed under the License is distributed on an "AS IS" BASIS,
 WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 See the License for the specific language governing permissions and
 limitations under the License.
-->

<!--
     For more details about configurations options that may appear in
     this file, see http://wiki.apache.org/solr/SolrConfigXml.
-->
<config>
  <!-- In all configuration below, a prefix of "solr." for class names
       is an alias that causes solr to search appropriate packages,
       including org.apache.solr.(search|update|request|core|analysis)

       You may also specify a fully qualified Java classname if you
       have your own custom plugins.
    -->

  <!-- Controls what version of Lucene various components of Solr
       adhere to.  Generally, you want to use the latest version to
       get all bug fixes and improvements. It is highly recommended
       that you fully re-index after changing this setting as it can
       affect both how text is indexed and queried.
  -->
  <luceneMatchVersion>4.6</luceneMatchVersion>

  <!-- <lib/> directives can be used to instruct Solr to load an Jars
       identified and use them to resolve any "plugins" specified in
       your solrconfig.xml or schema.xml (ie: Analyzers, Request
       Handlers, etc...).

       All directories and paths are resolved relative to the
       instanceDir.

       Please note that <lib/> directives are processed in the order
       that they appear in your solrconfig.xml file, and are "stacked"
       on top of each other when building a ClassLoader - so if you have
       plugin jars with dependencies on other jars, the "lower level"
       dependency jars should be loaded first.
```

```
     If a "./lib" directory exists in your instanceDir, all files
     found in it are included as if you had used the following
     syntax...

            <lib dir="./lib" />
  -->

<!-- A 'dir' option by itself adds any files found in the directory
     to the classpath, this is useful for including all jars in a
     directory.

     When a 'regex' is specified in addition to a 'dir', only the
     files in that directory which completely match the regex
     (anchored on both ends) will be included.

     If a 'dir' option (with or without a regex) is used and nothing
     is found that matches, a warning will be logged.

     The examples below can be used to load some solr-contribs along
     with their external dependencies.
  -->
<lib dir="../../../contrib/extraction/lib" regex=".*\.jar" />
<lib dir="../../../dist/" regex="solr-cell-\d.*\.jar" />

<lib dir="../../../contrib/clustering/lib/" regex=".*\.jar" />
<lib dir="../../../dist/" regex="solr-clustering-\d.*\.jar" />

<lib dir="../../../contrib/langid/lib/" regex=".*\.jar" />
<lib dir="../../../dist/" regex="solr-langid-\d.*\.jar" />

<lib dir="../../../contrib/velocity/lib" regex=".*\.jar" />
<lib dir="../../../dist/" regex="solr-velocity-\d.*\.jar" />

<!-- an exact 'path' can be used instead of a 'dir' to specify a
     specific jar file.  This will cause a serious error to be logged
     if it can't be loaded.
  -->
<!--
   <lib path="../a-jar-that-does-not-exist.jar" />
-->

<!-- Data Directory

     Used to specify an alternate directory to hold all index data
     other than the default ./data under the Solr home.  If
     replication is in use, this should match the replication
     configuration.
  -->
<dataDir>${solr.data.dir:}</dataDir>


<!-- The DirectoryFactory to use for indexes.
```

```
       solr.StandardDirectoryFactory is filesystem
       based and tries to pick the best implementation for the current
       JVM and platform.  solr.NRTCachingDirectoryFactory, the default,
       wraps solr.StandardDirectoryFactory and caches small files in memory
       for better NRT performance.

       One can force a particular implementation via solr.MMapDirectoryFactory,
       solr.NIOFSDirectoryFactory, or solr.SimpleFSDirectoryFactory.

       solr.RAMDirectoryFactory is memory based, not
       persistent, and doesn't work with replication.
    -->
  <directoryFactory name="DirectoryFactory"
                    class="${solr.directoryFactory:solr.NRTCachingDirectoryFact
ory}"/>

  <!-- The CodecFactory for defining the format of the inverted index.
       The default implementation is SchemaCodecFactory, which is the official
Lucene
       index format, but hooks into the schema to provide per-field customizati
on of
       the postings lists and per-document values in the fieldType element
       (postingsFormat/docValuesFormat). Note that most of the alternative impl
ementations
       are experimental, so if you choose to customize the index format, its a
good
       idea to convert back to the official format e.g. via IndexWriter.addInde
xes(IndexReader)
       before upgrading to a newer version to avoid unnecessary reindexing.
  -->
  <codecFactory class="solr.SchemaCodecFactory"/>

  <!-- To enable dynamic schema REST APIs, use the following for <schemaFactory
>:

       <schemaFactory class="ManagedIndexSchemaFactory">
         <bool name="mutable">true</bool>
         <str name="managedSchemaResourceName">managed-schema</str>
       </schemaFactory>

       When ManagedIndexSchemaFactory is specified, Solr will load the schema f
rom
       he resource named in 'managedSchemaResourceName', rather than from schem
a.xml.
       Note that the managed schema resource CANNOT be named schema.xml.  If th
e managed
       schema does not exist, Solr will create it after reading schema.xml, the
n rename
       'schema.xml' to 'schema.xml.bak'.

       Do NOT hand edit the managed schema - external modifications will be ign
ored and
```

```
       overwritten as a result of schema modification REST API calls.

       When ManagedIndexSchemaFactory is specified with mutable = true, schema
       modification REST API calls will be allowed; otherwise, error responses
will be
       sent back for these requests.
  -->
  <schemaFactory class="ClassicIndexSchemaFactory"/>

  <!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
       Index Config - These settings control low-level behavior of indexing
       Most example settings here show the default value, but are commented
       out, to more easily see where customizations have been made.

       Note: This replaces <indexDefaults> and <mainIndex> from older versions
       ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ --
>
  <indexConfig>
    <!-- maxFieldLength was removed in 4.0. To get similar behavior, include a
         LimitTokenCountFilterFactory in your fieldType definition. E.g.
     <filter class="solr.LimitTokenCountFilterFactory" maxTokenCount="10000"/>
    -->
    <!-- Maximum time to wait for a write lock (ms) for an IndexWriter. Default
: 1000 -->
    <!-- <writeLockTimeout>1000</writeLockTimeout>  -->

    <!-- The maximum number of simultaneous threads that may be
         indexing documents at once in IndexWriter; if more than this
         many threads arrive they will wait for others to finish.
         Default in Solr/Lucene is 8. -->
    <!-- <maxIndexingThreads>8</maxIndexingThreads>  -->

    <!-- Expert: Enabling compound file will use less files for the index,
         using fewer file descriptors on the expense of performance decrease.
         Default in Lucene is "true". Default in Solr is "false" (since 3.6) --
>
    <!-- <useCompoundFile>false</useCompoundFile> -->

    <!-- ramBufferSizeMB sets the amount of RAM that may be used by Lucene
         indexing for buffering added documents and deletions before they are
         flushed to the Directory.
         maxBufferedDocs sets a limit on the number of documents buffered
         before flushing.
         If both ramBufferSizeMB and maxBufferedDocs is set, then
         Lucene will flush based on whichever limit is hit first.
         The default is 100 MB.  -->
    <!-- <ramBufferSizeMB>100</ramBufferSizeMB> -->
    <!-- <maxBufferedDocs>1000</maxBufferedDocs> -->

    <!-- Expert: Merge Policy
         The Merge Policy in Lucene controls how merging of segments is done.
         The default since Solr/Lucene 3.3 is TieredMergePolicy.
         The default since Lucene 2.3 was the LogByteSizeMergePolicy,
```

```
          Even older versions of Lucene used LogDocMergePolicy.
      -->
    <!--
        <mergePolicy class="org.apache.lucene.index.TieredMergePolicy">
          <int name="maxMergeAtOnce">10</int>
          <int name="segmentsPerTier">10</int>
        </mergePolicy>
      -->

    <!-- Merge Factor
         The merge factor controls how many segments will get merged at a time.
         For TieredMergePolicy, mergeFactor is a convenience parameter which
         will set both MaxMergeAtOnce and SegmentsPerTier at once.
         For LogByteSizeMergePolicy, mergeFactor decides how many new segments
         will be allowed before they are merged into one.
         Default is 10 for both merge policies.
      -->
    <!--
    <mergeFactor>10</mergeFactor>
      -->

    <!-- Expert: Merge Scheduler
         The Merge Scheduler in Lucene controls how merges are
         performed.  The ConcurrentMergeScheduler (Lucene 2.3 default)
         can perform merges in the background using separate threads.
         The SerialMergeScheduler (Lucene 2.2 default) does not.
     -->
    <!--
       <mergeScheduler class="org.apache.lucene.index.ConcurrentMergeScheduler"
/>
       -->

    <!-- LockFactory

         This option specifies which Lucene LockFactory implementation
         to use.

         single = SingleInstanceLockFactory - suggested for a
                  read-only index or when there is no possibility of
                  another process trying to modify the index.
         native = NativeFSLockFactory - uses OS native file locking.
                  Do not use when multiple solr webapps in the same
                  JVM are attempting to share a single index.
         simple = SimpleFSLockFactory  - uses a plain file for locking

         Defaults: 'native' is default for Solr3.6 and later, otherwise
                   'simple' is the default

         More details on the nuances of each LockFactory...
         http://wiki.apache.org/lucene-java/AvailableLockFactories
    -->
    <lockType>${solr.lock.type:native}</lockType>
```

```
    <!-- Unlock On Startup

         If true, unlock any held write or commit locks on startup.
         This defeats the locking mechanism that allows multiple
         processes to safely access a lucene index, and should be used
         with care. Default is "false".

         This is not needed if lock type is 'single'
     -->
    <!--
    <unlockOnStartup>false</unlockOnStartup>
      -->

    <!-- Expert: Controls how often Lucene loads terms into memory
         Default is 128 and is likely good for most everyone.
      -->
    <!-- <termIndexInterval>128</termIndexInterval> -->

    <!-- If true, IndexReaders will be opened/reopened from the IndexWriter
         instead of from the Directory. Hosts in a master/slave setup
         should have this set to false while those in a SolrCloud
         cluster need to be set to true. Default: true
      -->
    <!--
    <nrtMode>true</nrtMode>
      -->

    <!-- Commit Deletion Policy
         Custom deletion policies can be specified here. The class must
         implement org.apache.lucene.index.IndexDeletionPolicy.

         The default Solr IndexDeletionPolicy implementation supports
         deleting index commit points on number of commits, age of
         commit point and optimized status.

         The latest commit point should always be preserved regardless
         of the criteria.
    -->
    <!--
    <deletionPolicy class="solr.SolrDeletionPolicy">
    -->
      <!-- The number of commit points to be kept -->
      <!-- <str name="maxCommitsToKeep">1</str> -->
      <!-- The number of optimized commit points to be kept -->
      <!-- <str name="maxOptimizedCommitsToKeep">0</str> -->
      <!--
          Delete all commit points once they have reached the given age.
          Supports DateMathParser syntax e.g.
        -->
      <!--
         <str name="maxCommitAge">30MINUTES</str>
         <str name="maxCommitAge">1DAY</str>
      -->
```

```
    <!--
    </deletionPolicy>
    -->

    <!-- Lucene Infostream

         To aid in advanced debugging, Lucene provides an "InfoStream"
         of detailed information when indexing.

         Setting the value to true will instruct the underlying Lucene
         IndexWriter to write its info stream to solr's log. By default,
         this is enabled here, and controlled through log4j.properties.
      -->
     <infoStream>true</infoStream>
  </indexConfig>


  <!-- JMX

       This example enables JMX if and only if an existing MBeanServer
       is found, use this if you want to configure JMX through JVM
       parameters. Remove this to disable exposing Solr configuration
       and statistics to JMX.

       For more details see http://wiki.apache.org/solr/SolrJmx
    -->
  <jmx />
  <!-- If you want to connect to a particular server, specify the
       agentId
    -->
  <!-- <jmx agentId="myAgent" /> -->
  <!-- If you want to start a new MBeanServer, specify the serviceUrl -->
  <!-- <jmx serviceUrl="service:jmx:rmi:///jndi/rmi://localhost:9999/solr"/>
    -->

  <!-- The default high-performance update handler -->
  <updateHandler class="solr.DirectUpdateHandler2">

    <!-- Enables a transaction log, used for real-time get, durability, and
         and solr cloud replica recovery.  The log can grow as big as
         uncommitted changes to the index, so use of a hard autoCommit
         is recommended (see below).
         "dir" - the target directory for transaction logs, defaults to the
                solr data directory.  -->
    <updateLog>
      <str name="dir">${solr.ulog.dir:}</str>
    </updateLog>

    <!-- AutoCommit

         Perform a hard commit automatically under certain conditions.
         Instead of enabling autoCommit, consider using "commitWithin"
         when adding documents.
```

```
       http://wiki.apache.org/solr/UpdateXmlMessages

       maxDocs - Maximum number of documents to add since the last
                 commit before automatically triggering a new commit.

       maxTime - Maximum amount of time in ms that is allowed to pass
                 since a document was added before automatically
                 triggering a new commit.
       openSearcher - if false, the commit causes recent index changes
         to be flushed to stable storage, but does not cause a new
         searcher to be opened to make those changes visible.

       If the updateLog is enabled, then it's highly recommended to
       have some sort of hard autoCommit to limit the log size.
    -->
   <autoCommit>
     <maxTime>${solr.autoCommit.maxTime:15000}</maxTime>
     <openSearcher>false</openSearcher>
   </autoCommit>

  <!-- softAutoCommit is like autoCommit except it causes a
       'soft' commit which only ensures that changes are visible
       but does not ensure that data is synced to disk.  This is
       faster and more near-realtime friendly than a hard commit.
    -->

   <autoSoftCommit>
     <maxTime>${solr.autoSoftCommit.maxTime:-1}</maxTime>
   </autoSoftCommit>

  <!-- Update Related Event Listeners

       Various IndexWriter related events can trigger Listeners to
       take actions.

       postCommit - fired after every commit or optimize command
       postOptimize - fired after every optimize command
    -->
  <!-- The RunExecutableListener executes an external command from a
       hook such as postCommit or postOptimize.

       exe - the name of the executable to run
       dir - dir to use as the current working directory. (default=".")
       wait - the calling thread waits until the executable returns.
              (default="true")
       args - the arguments to pass to the program.  (default is none)
       env - environment variables to set.  (default is none)
    -->
  <!-- This example shows how RunExecutableListener could be used
       with the script based replication...
       http://wiki.apache.org/solr/CollectionDistribution
    -->
```

```
    <!--
       <listener event="postCommit" class="solr.RunExecutableListener">
         <str name="exe">solr/bin/snapshooter</str>
         <str name="dir">.</str>
         <bool name="wait">true</bool>
         <arr name="args"> <str>arg1</str> <str>arg2</str> </arr>
         <arr name="env"> <str>MYVAR=val1</str> </arr>
       </listener>
      -->

  </updateHandler>

  <!-- IndexReaderFactory

       Use the following format to specify a custom IndexReaderFactory,
       which allows for alternate IndexReader implementations.

       ** Experimental Feature **

       Please note - Using a custom IndexReaderFactory may prevent
       certain other features from working. The API to
       IndexReaderFactory may change without warning or may even be
       removed from future releases if the problems cannot be
       resolved.


       ** Features that may not work with custom IndexReaderFactory **

       The ReplicationHandler assumes a disk-resident index. Using a
       custom IndexReader implementation may cause incompatibility
       with ReplicationHandler and may cause replication to not work
       correctly. See SOLR-1366 for details.

    -->
  <!--
  <indexReaderFactory name="IndexReaderFactory" class="package.class">
    <str name="someArg">Some Value</str>
  </indexReaderFactory >
  -->
  <!-- By explicitly declaring the Factory, the termIndexDivisor can
       be specified.
    -->
  <!--
     <indexReaderFactory name="IndexReaderFactory"
                         class="solr.StandardIndexReaderFactory">
       <int name="setTermIndexDivisor">12</int>
     </indexReaderFactory >
    -->

  <!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
       Query section - these settings control query time things like caches
       ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ --
>
```

```
  <query>
    <!-- Max Boolean Clauses

         Maximum number of clauses in each BooleanQuery,  an exception
         is thrown if exceeded.

         ** WARNING **

         This option actually modifies a global Lucene property that
         will affect all SolrCores.  If multiple solrconfig.xml files
         disagree on this property, the value at any given moment will
         be based on the last SolrCore to be initialized.

      -->
    <maxBooleanClauses>1024</maxBooleanClauses>


    <!-- Solr Internal Query Caches

         There are two implementations of cache available for Solr,
         LRUCache, based on a synchronized LinkedHashMap, and
         FastLRUCache, based on a ConcurrentHashMap.

         FastLRUCache has faster gets and slower puts in single
         threaded operation and thus is generally faster than LRUCache
         when the hit ratio of the cache is high (> 75%), and may be
         faster under other scenarios on multi-cpu systems.
    -->

    <!-- Filter Cache

         Cache used by SolrIndexSearcher for filters (DocSets),
         unordered sets of *all* documents that match a query.  When a
         new searcher is opened, its caches may be prepopulated or
         "autowarmed" using data from caches in the old searcher.
         autowarmCount is the number of items to prepopulate.  For
         LRUCache, the autowarmed items will be the most recently
         accessed items.

         Parameters:
           class - the SolrCache implementation LRUCache or
               (LRUCache or FastLRUCache)
           size - the maximum number of entries in the cache
           initialSize - the initial capacity (number of entries) of
               the cache.  (see java.util.HashMap)
           autowarmCount - the number of entries to prepopulate from
               and old cache.
      -->
    <filterCache class="solr.FastLRUCache"
                 size="1024"
                 initialSize="512"
                 autowarmCount="0"/>
```

```
    <!-- Query Result Cache

         Caches results of searches - ordered lists of document ids
         (DocList) based on a query, a sort, and the range of documents request
ed.
      -->
    <queryResultCache class="solr.LRUCache"
                      size="512"
                      initialSize="512"
                      autowarmCount="0"/>

    <!-- Document Cache

         Caches Lucene Document objects (the stored fields for each
         document).  Since Lucene internal document ids are transient,
         this cache will not be autowarmed.
      -->
    <documentCache class="solr.LRUCache"
                   size="512"
                   initialSize="512"
                   autowarmCount="0"/>

    <!-- custom cache currently used by block join -->
    <cache name="perSegFilter"
      class="solr.search.LRUCache"
      size="10"
      initialSize="0"
      autowarmCount="10"
      regenerator="solr.NoOpRegenerator" />

    <!-- Field Value Cache

         Cache used to hold field values that are quickly accessible
         by document id.  The fieldValueCache is created by default
         even if not configured here.
      -->
    <!--
       <fieldValueCache class="solr.FastLRUCache"
                        size="512"
                        autowarmCount="128"
                        showItems="32" />
      -->

    <!-- Custom Cache

         Example of a generic cache.  These caches may be accessed by
         name through SolrIndexSearcher.getCache(),cacheLookup(), and
         cacheInsert().  The purpose is to enable easy caching of
         user/application level data.  The regenerator argument should
         be specified as an implementation of solr.CacheRegenerator
         if autowarming is desired.
      -->
    <!--
```

```
     <cache name="myUserCache"
            class="solr.LRUCache"
            size="4096"
            initialSize="1024"
            autowarmCount="1024"
            regenerator="com.mycompany.MyRegenerator"
            />
    -->


  <!-- Lazy Field Loading

       If true, stored fields that are not requested will be loaded
       lazily.  This can result in a significant speed improvement
       if the usual case is to not load all stored fields,
       especially if the skipped fields are large compressed text
       fields.
  -->
  <enableLazyFieldLoading>true</enableLazyFieldLoading>

 <!-- Use Filter For Sorted Query

      A possible optimization that attempts to use a filter to
      satisfy a search.  If the requested sort does not include
      score, then the filterCache will be checked for a filter
      matching the query. If found, the filter will be used as the
      source of document ids, and then the sort will be applied to
      that.

      For most situations, this will not be useful unless you
      frequently get the same search repeatedly with different sort
      options, and none of them ever use "score"
   -->
 <!--
    <useFilterForSortedQuery>true</useFilterForSortedQuery>
   -->

 <!-- Result Window Size

      An optimization for use with the queryResultCache.  When a search
      is requested, a superset of the requested number of document ids
      are collected.  For example, if a search for a particular query
      requests matching documents 10 through 19, and queryWindowSize is 50,
      then documents 0 through 49 will be collected and cached.  Any further
      requests in that range can be satisfied via the cache.
   -->
 <queryResultWindowSize>20</queryResultWindowSize>

 <!-- Maximum number of documents to cache for any entry in the
      queryResultCache.
   -->
 <queryResultMaxDocsCached>200</queryResultMaxDocsCached>
```

```
    <!-- Query Related Event Listeners

         Various IndexSearcher related events can trigger Listeners to
         take actions.

         newSearcher - fired whenever a new searcher is being prepared
         and there is a current searcher handling requests (aka
         registered).  It can be used to prime certain caches to
         prevent long request times for certain requests.

         firstSearcher - fired whenever a new searcher is being
         prepared but there is no current registered searcher to handle
         requests or to gain autowarming data from.


      -->
    <!-- QuerySenderListener takes an array of NamedList and executes a
         local query request for each NamedList in sequence.
      -->
    <listener event="newSearcher" class="solr.QuerySenderListener">
      <arr name="queries">
        <!--
           <lst><str name="q">solr</str><str name="sort">price asc</str></lst>
           <lst><str name="q">rocks</str><str name="sort">weight asc</str></lst
>
          -->
      </arr>
    </listener>
    <listener event="firstSearcher" class="solr.QuerySenderListener">
      <arr name="queries">
        <lst>
          <str name="q">static firstSearcher warming in solrconfig.xml</str>
        </lst>
      </arr>
    </listener>

    <!-- Use Cold Searcher

         If a search request comes in and there is no current
         registered searcher, then immediately register the still
         warming searcher and use it.  If "false" then all requests
         will block until the first searcher is done warming.
      -->
    <useColdSearcher>false</useColdSearcher>

    <!-- Max Warming Searchers

         Maximum number of searchers that may be warming in the
         background concurrently.  An error is returned if this limit
         is exceeded.

         Recommend values of 1-2 for read-only slaves, higher for
         masters w/o cache warming.
```

```
      -->
    <maxWarmingSearchers>2</maxWarmingSearchers>

  </query>


  <!-- Request Dispatcher

       This section contains instructions for how the SolrDispatchFilter
       should behave when processing requests for this SolrCore.

       handleSelect is a legacy option that affects the behavior of requests
       such as /select?qt=XXX

       handleSelect="true" will cause the SolrDispatchFilter to process
       the request and dispatch the query to a handler specified by the
       "qt" param, assuming "/select" isn't already registered.

       handleSelect="false" will cause the SolrDispatchFilter to
       ignore "/select" requests, resulting in a 404 unless a handler
       is explicitly registered with the name "/select"

       handleSelect="true" is not recommended for new users, but is the default
       for backwards compatibility
    -->
  <requestDispatcher handleSelect="false" >
    <!-- Request Parsing

         These settings indicate how Solr Requests may be parsed, and
         what restrictions may be placed on the ContentStreams from
         those requests

         enableRemoteStreaming - enables use of the stream.file
         and stream.url parameters for specifying remote streams.

         multipartUploadLimitInKB - specifies the max size (in KiB) of
         Multipart File Uploads that Solr will allow in a Request.

         formdataUploadLimitInKB - specifies the max size (in KiB) of
         form data (application/x-www-form-urlencoded) sent via
         POST. You can use POST to pass request parameters not
         fitting into the URL.

         addHttpRequestToContext - if set to true, it will instruct
         the requestParsers to include the original HttpServletRequest
         object in the context map of the SolrQueryRequest under the
         key "httpRequest". It will not be used by any of the existing
         Solr components, but may be useful when developing custom
         plugins.

         *** WARNING ***
         The settings below authorize Solr to fetch remote files, You
         should make sure your system has some authentication before
```

```
       using enableRemoteStreaming="true"

    -->
  <requestParsers enableRemoteStreaming="true"
                  multipartUploadLimitInKB="2048000"
                  formdataUploadLimitInKB="2048"
                  addHttpRequestToContext="false"/>

  <!-- HTTP Caching

       Set HTTP caching related parameters (for proxy caches and clients).

       The options below instruct Solr not to output any HTTP Caching
       related headers
    -->
  <httpCaching never304="true" />
  <!-- If you include a <cacheControl> directive, it will be used to
       generate a Cache-Control header (as well as an Expires header
       if the value contains "max-age=")

       By default, no Cache-Control header is generated.

       You can use the <cacheControl> option even if you have set
       never304="true"
    -->
  <!--
     <httpCaching never304="true" >
       <cacheControl>max-age=30, public</cacheControl>
     </httpCaching>
    -->
  <!-- To enable Solr to respond with automatically generated HTTP
       Caching headers, and to response to Cache Validation requests
       correctly, set the value of never304="false"

       This will cause Solr to generate Last-Modified and ETag
       headers based on the properties of the Index.

       The following options can also be specified to affect the
       values of these headers...

       lastModFrom - the default value is "openTime" which means the
       Last-Modified value (and validation against If-Modified-Since
       requests) will all be relative to when the current Searcher
       was opened.  You can change it to lastModFrom="dirLastMod" if
       you want the value to exactly correspond to when the physical
       index was last modified.

       etagSeed="..." is an option you can change to force the ETag
       header (and validation against If-None-Match requests) to be
       different even if the index has not changed (ie: when making
       significant changes to your config file)

       (lastModifiedFrom and etagSeed are both ignored if you use
```

```
         the never304="true" option)
      -->
    <!--
       <httpCaching lastModifiedFrom="openTime"
                    etagSeed="Solr">
         <cacheControl>max-age=30, public</cacheControl>
       </httpCaching>
      -->
  </requestDispatcher>

  <!-- Request Handlers

       http://wiki.apache.org/solr/SolrRequestHandler

       Incoming queries will be dispatched to a specific handler by name
       based on the path specified in the request.

       Legacy behavior: If the request path uses "/select" but no Request
       Handler has that name, and if handleSelect="true" has been specified in
       the requestDispatcher, then the Request Handler is dispatched based on
       the qt parameter.  Handlers without a leading '/' are accessed this way
       like so: http://host/app/[core/]select?qt=name  If no qt is
       given, then the requestHandler that declares default="true" will be
       used or the one named "standard".

       If a Request Handler is declared with startup="lazy", then it will
       not be initialized until the first request that uses it.

    -->
  <!-- SearchHandler

       http://wiki.apache.org/solr/SearchHandler

       For processing Search Queries, the primary Request Handler
       provided with Solr is "SearchHandler" It delegates to a sequent
       of SearchComponents (see below) and supports distributed
       queries across multiple shards
    -->
  <requestHandler name="/select" class="solr.SearchHandler">
    <!-- default values for query parameters can be specified, these
         will be overridden by parameters in the request
      -->
     <lst name="defaults">
       <str name="echoParams">explicit</str>
       <int name="rows">10</int>
       <str name="df">text</str>
     </lst>
    <!-- In addition to defaults, "appends" params can be specified
         to identify values which should be appended to the list of
         multi-val params from the query (or the existing "defaults").
      -->
    <!-- In this example, the param "fq=instock:true" would be appended to
         any query time fq params the user may specify, as a mechanism for
```

```
        partitioning the index, independent of any user selected filtering
        that may also be desired (perhaps as a result of faceted searching).

        NOTE: there is *absolutely* nothing a client can do to prevent these
        "appends" values from being used, so don't use this mechanism
        unless you are sure you always want it.
      -->
    <!--
       <lst name="appends">
         <str name="fq">inStock:true</str>
       </lst>
      -->
    <!-- "invariants" are a way of letting the Solr maintainer lock down
         the options available to Solr clients.  Any params values
         specified here are used regardless of what values may be specified
         in either the query, the "defaults", or the "appends" params.

         In this example, the facet.field and facet.query params would
         be fixed, limiting the facets clients can use.  Faceting is
         not turned on by default - but if the client does specify
         facet=true in the request, these are the only facets they
         will be able to see counts for; regardless of what other
         facet.field or facet.query params they may specify.

         NOTE: there is *absolutely* nothing a client can do to prevent these
         "invariants" values from being used, so don't use this mechanism
         unless you are sure you always want it.
      -->
    <!--
       <lst name="invariants">
         <str name="facet.field">cat</str>
         <str name="facet.field">manu_exact</str>
         <str name="facet.query">price:[* TO 500]</str>
         <str name="facet.query">price:[500 TO *]</str>
       </lst>
      -->
    <!-- If the default list of SearchComponents is not desired, that
         list can either be overridden completely, or components can be
         prepended or appended to the default list.  (see below)
      -->
    <!--
       <arr name="components">
         <str>nameOfCustomComponent1</str>
         <str>nameOfCustomComponent2</str>
       </arr>
      -->
    </requestHandler>

  <!-- A request handler that returns indented JSON by default -->
  <requestHandler name="/query" class="solr.SearchHandler">
     <lst name="defaults">
       <str name="echoParams">explicit</str>
       <str name="wt">json</str>
```

```
      <str name="indent">true</str>
      <str name="df">text</str>
    </lst>
  </requestHandler>


  <!-- realtime get handler, guaranteed to return the latest stored fields of
       any document, without the need to commit or open a new searcher.  The
       current implementation relies on the updateLog feature being enabled. --
>
  <requestHandler name="/get" class="solr.RealTimeGetHandler">
    <lst name="defaults">
      <str name="omitHeader">true</str>
      <str name="wt">json</str>
      <str name="indent">true</str>
    </lst>
  </requestHandler>


  <!-- A Robust Example

       This example SearchHandler declaration shows off usage of the
       SearchHandler with many defaults declared

       Note that multiple instances of the same Request Handler
       (SearchHandler) can be registered multiple times with different
       names (and different init parameters)
    -->
  <requestHandler name="/browse" class="solr.SearchHandler">
    <lst name="defaults">
      <str name="echoParams">explicit</str>

      <!-- VelocityResponseWriter settings -->
      <str name="wt">velocity</str>
      <str name="v.template">browse</str>
      <str name="v.layout">layout</str>
      <str name="title">Solritas</str>

      <!-- Query settings -->
      <str name="defType">edismax</str>
      <str name="qf">
         text^0.5 features^1.0 name^1.2 sku^1.5 id^10.0 manu^1.1 cat^1.4
         title^10.0 description^5.0 keywords^5.0 author^2.0 resourcename^1.0
      </str>
      <str name="df">text</str>
      <str name="mm">100%</str>
      <str name="q.alt">*:*</str>
      <str name="rows">10</str>
      <str name="fl">*,score</str>

      <str name="mlt.qf">
        text^0.5 features^1.0 name^1.2 sku^1.5 id^10.0 manu^1.1 cat^1.4
        title^10.0 description^5.0 keywords^5.0 author^2.0 resourcename^1.0
```

```
        </str>
        <str name="mlt.fl">text,features,name,sku,id,manu,cat,title,description,
keywords,author,resourcename</str>
        <int name="mlt.count">3</int>

        <!-- Faceting defaults -->
        <str name="facet">on</str>
        <str name="facet.field">cat</str>
        <str name="facet.field">manu_exact</str>
        <str name="facet.field">content_type</str>
        <str name="facet.field">author_s</str>
        <str name="facet.query">ipod</str>
        <str name="facet.query">GB</str>
        <str name="facet.mincount">1</str>
        <str name="facet.pivot">cat,inStock</str>
        <str name="facet.range.other">after</str>
        <str name="facet.range">price</str>
        <int name="f.price.facet.range.start">0</int>
        <int name="f.price.facet.range.end">600</int>
        <int name="f.price.facet.range.gap">50</int>
        <str name="facet.range">popularity</str>
        <int name="f.popularity.facet.range.start">0</int>
        <int name="f.popularity.facet.range.end">10</int>
        <int name="f.popularity.facet.range.gap">3</int>
        <str name="facet.range">manufacturedate_dt</str>
        <str name="f.manufacturedate_dt.facet.range.start">NOW/YEAR-10YEARS</str
>
        <str name="f.manufacturedate_dt.facet.range.end">NOW</str>
        <str name="f.manufacturedate_dt.facet.range.gap">+1YEAR</str>
        <str name="f.manufacturedate_dt.facet.range.other">before</str>
        <str name="f.manufacturedate_dt.facet.range.other">after</str>

        <!-- Highlighting defaults -->
        <str name="hl">on</str>
        <str name="hl.fl">content features title name</str>
        <str name="hl.encoder">html</str>
        <str name="hl.simple.pre">&lt;b&gt;</str>
        <str name="hl.simple.post">&lt;/b&gt;</str>
        <str name="f.title.hl.fragsize">0</str>
        <str name="f.title.hl.alternateField">title</str>
        <str name="f.name.hl.fragsize">0</str>
        <str name="f.name.hl.alternateField">name</str>
        <str name="f.content.hl.snippets">3</str>
        <str name="f.content.hl.fragsize">200</str>
        <str name="f.content.hl.alternateField">content</str>
        <str name="f.content.hl.maxAlternateFieldLength">750</str>

        <!-- Spell checking defaults -->
        <str name="spellcheck">on</str>
        <str name="spellcheck.extendedResults">false</str>
        <str name="spellcheck.count">5</str>
        <str name="spellcheck.alternativeTermCount">2</str>
        <str name="spellcheck.maxResultsForSuggest">5</str>
```

```
      <str name="spellcheck.collate">true</str>
      <str name="spellcheck.collateExtendedResults">true</str>
      <str name="spellcheck.maxCollationTries">5</str>
      <str name="spellcheck.maxCollations">3</str>
    </lst>

    <!-- append spellchecking to our list of components -->
    <arr name="last-components">
      <str>spellcheck</str>
    </arr>
  </requestHandler>


  <!-- Update Request Handler.

       http://wiki.apache.org/solr/UpdateXmlMessages

       The canonical Request Handler for Modifying the Index through
       commands specified using XML, JSON, CSV, or JAVABIN

       Note: Since solr1.1 requestHandlers requires a valid content
       type header if posted in the body. For example, curl now
       requires: -H 'Content-type:text/xml; charset=utf-8'

       To override the request content type and force a specific
       Content-type, use the request parameter:
         ?update.contentType=text/csv

       This handler will pick a response format to match the input
       if the 'wt' parameter is not explicit
    -->
  <requestHandler name="/update" class="solr.UpdateRequestHandler">
    <!-- See below for information on defining
         updateRequestProcessorChains that can be used by name
         on each Update Request
      -->
    <!--
       <lst name="defaults">
         <str name="update.chain">dedupe</str>
       </lst>
       -->
  </requestHandler>

  <!-- for back compat with clients using /update/json and /update/csv -->
  <requestHandler name="/update/json" class="solr.JsonUpdateRequestHandler">
        <lst name="defaults">
         <str name="stream.contentType">application/json</str>
       </lst>
  </requestHandler>
  <requestHandler name="/update/csv" class="solr.CSVRequestHandler">
        <lst name="defaults">
         <str name="stream.contentType">application/csv</str>
       </lst>
```

```
  </requestHandler>

  <!-- Solr Cell Update Request Handler

       http://wiki.apache.org/solr/ExtractingRequestHandler

    -->
  <requestHandler name="/update/extract"
                  startup="lazy"
                  class="solr.extraction.ExtractingRequestHandler" >
    <lst name="defaults">
      <str name="lowernames">true</str>
      <str name="uprefix">ignored_</str>

      <!-- capture link hrefs but ignore div attributes -->
      <str name="captureAttr">true</str>
      <str name="fmap.a">links</str>
      <str name="fmap.div">ignored_</str>
    </lst>
  </requestHandler>


  <!-- Field Analysis Request Handler

       RequestHandler that provides much the same functionality as
       analysis.jsp. Provides the ability to specify multiple field
       types and field names in the same request and outputs
       index-time and query-time analysis for each of them.

       Request parameters are:
       analysis.fieldname - field name whose analyzers are to be used

       analysis.fieldtype - field type whose analyzers are to be used
       analysis.fieldvalue - text for index-time analysis
       q (or analysis.q) - text for query time analysis
       analysis.showmatch (true|false) - When set to true and when
           query analysis is performed, the produced tokens of the
           field value analysis will be marked as "matched" for every
           token that is produces by the query analysis
   -->
  <requestHandler name="/analysis/field"
                  startup="lazy"
                  class="solr.FieldAnalysisRequestHandler" />


  <!-- Document Analysis Handler

       http://wiki.apache.org/solr/AnalysisRequestHandler

       An analysis handler that provides a breakdown of the analysis
       process of provided documents. This handler expects a (single)
       content stream with the following format:
```

```
       <docs>
         <doc>
           <field name="id">1</field>
           <field name="name">The Name</field>
           <field name="text">The Text Value</field>
         </doc>
         <doc>...</doc>
         <doc>...</doc>
         ...
       </docs>

    Note: Each document must contain a field which serves as the
    unique key. This key is used in the returned response to associate
    an analysis breakdown to the analyzed document.

    Like the FieldAnalysisRequestHandler, this handler also supports
    query analysis by sending either an "analysis.query" or "q"
    request parameter that holds the query text to be analyzed. It
    also supports the "analysis.showmatch" parameter which when set to
    true, all field tokens that match the query tokens will be marked
    as a "match".
  -->
  <requestHandler name="/analysis/document"
                  class="solr.DocumentAnalysisRequestHandler"
                  startup="lazy" />

  <!-- Admin Handlers

       Admin Handlers - This will register all the standard admin
       RequestHandlers.
    -->
  <requestHandler name="/admin/"
                  class="solr.admin.AdminHandlers" />
  <!-- This single handler is equivalent to the following... -->
  <!--
     <requestHandler name="/admin/luke"       class="solr.admin.LukeRequestHand
ler" />
     <requestHandler name="/admin/system"     class="solr.admin.SystemInfoHandl
er" />
     <requestHandler name="/admin/plugins"    class="solr.admin.PluginInfoHandl
er" />
     <requestHandler name="/admin/threads"    class="solr.admin.ThreadDumpHandl
er" />
     <requestHandler name="/admin/properties" class="solr.admin.PropertiesReque
stHandler" />
     <requestHandler name="/admin/file"       class="solr.admin.ShowFileRequest
Handler" >
    -->
  <!-- If you wish to hide files under ${solr.home}/conf, explicitly
       register the ShowFileRequestHandler using:
    -->
  <!--
     <requestHandler name="/admin/file"
```

```
                    class="solr.admin.ShowFileRequestHandler" >
      <lst name="invariants">
        <str name="hidden">synonyms.txt</str>
        <str name="hidden">anotherfile.txt</str>
      </lst>
    </requestHandler>
  -->

<!-- ping/healthcheck -->
<requestHandler name="/admin/ping" class="solr.PingRequestHandler">
  <lst name="invariants">
    <str name="q">solrpingquery</str>
  </lst>
  <lst name="defaults">
    <str name="echoParams">all</str>
  </lst>
  <!-- An optional feature of the PingRequestHandler is to configure the
       handler with a "healthcheckFile" which can be used to enable/disable
       the PingRequestHandler.
       relative paths are resolved against the data dir
    -->
  <!-- <str name="healthcheckFile">server-enabled.txt</str> -->
</requestHandler>

<!-- Echo the request contents back to the client -->
<requestHandler name="/debug/dump" class="solr.DumpRequestHandler" >
  <lst name="defaults">
   <str name="echoParams">explicit</str>
   <str name="echoHandler">true</str>
  </lst>
</requestHandler>

<!-- Solr Replication

     The SolrReplicationHandler supports replicating indexes from a
     "master" used for indexing and "slaves" used for queries.

     http://wiki.apache.org/solr/SolrReplication

     It is also necessary for SolrCloud to function (in Cloud mode, the
     replication handler is used to bulk transfer segments when nodes
     are added or need to recover).

     https://wiki.apache.org/solr/SolrCloud/
  -->
<requestHandler name="/replication" class="solr.ReplicationHandler" >
  <!--
     To enable simple master/slave replication, uncomment one of the
     sections below, depending on whether this solr instance should be
     the "master" or a "slave".  If this instance is a "slave" you will
     also need to fill in the masterUrl to point to a real machine.
  -->
  <!--
```

```
      <lst name="master">
        <str name="replicateAfter">commit</str>
        <str name="replicateAfter">startup</str>
        <str name="confFiles">schema.xml,stopwords.txt</str>
      </lst>
    -->
    <!--
      <lst name="slave">
        <str name="masterUrl">http://your-master-hostname:8983/solr</str>
        <str name="pollInterval">00:00:60</str>
      </lst>
    -->
  </requestHandler>

  <!-- Search Components

       Search components are registered to SolrCore and used by
       instances of SearchHandler (which can access them by name)

       By default, the following components are available:

       <searchComponent name="query"     class="solr.QueryComponent" />
       <searchComponent name="facet"     class="solr.FacetComponent" />
       <searchComponent name="mlt"       class="solr.MoreLikeThisComponent" />
       <searchComponent name="highlight" class="solr.HighlightComponent" />
       <searchComponent name="stats"     class="solr.StatsComponent" />
       <searchComponent name="debug"     class="solr.DebugComponent" />

       Default configuration in a requestHandler would look like:

       <arr name="components">
         <str>query</str>
         <str>facet</str>
         <str>mlt</str>
         <str>highlight</str>
         <str>stats</str>
         <str>debug</str>
       </arr>

       If you register a searchComponent to one of the standard names,
       that will be used instead of the default.

       To insert components before or after the 'standard' components, use:

       <arr name="first-components">
         <str>myFirstComponentName</str>
       </arr>

       <arr name="last-components">
         <str>myLastComponentName</str>
       </arr>

       NOTE: The component registered with the name "debug" will
```

```
       always be executed after the "last-components"

     -->

   <!-- Spell Check

        The spell check component can return a list of alternative spelling
        suggestions.

        http://wiki.apache.org/solr/SpellCheckComponent
     -->
  <searchComponent name="spellcheck" class="solr.SpellCheckComponent">

    <str name="queryAnalyzerFieldType">text_general</str>

    <!-- Multiple "Spell Checkers" can be declared and used by this
         component
      -->

    <!-- a spellchecker built from a field of the main index -->
    <lst name="spellchecker">
      <str name="name">default</str>
      <str name="field">text</str>
      <str name="classname">solr.DirectSolrSpellChecker</str>
      <!-- the spellcheck distance measure used, the default is the internal le
venshtein -->
      <str name="distanceMeasure">internal</str>
      <!-- minimum accuracy needed to be considered a valid spellcheck suggesti
on -->
      <float name="accuracy">0.5</float>
      <!-- the maximum #edits we consider when enumerating terms: can be 1 or 2
-->
      <int name="maxEdits">2</int>
      <!-- the minimum shared prefix when enumerating terms -->
      <int name="minPrefix">1</int>
      <!-- maximum number of inspections per result. -->
      <int name="maxInspections">5</int>
      <!-- minimum length of a query term to be considered for correction -->
      <int name="minQueryLength">4</int>
      <!-- maximum threshold of documents a query term can appear to be conside
red for correction -->
      <float name="maxQueryFrequency">0.01</float>
      <!-- uncomment this to require suggestions to occur in 1% of the document
s
        <float name="thresholdTokenFrequency">.01</float>
      -->
    </lst>

    <!-- a spellchecker that can break or combine words.  See "/spell" handler
below for usage -->
    <lst name="spellchecker">
      <str name="name">wordbreak</str>
      <str name="classname">solr.WordBreakSolrSpellChecker</str>
```

```
      <str name="field">name</str>
      <str name="combineWords">true</str>
      <str name="breakWords">true</str>
      <int name="maxChanges">10</int>
    </lst>

    <!-- a spellchecker that uses a different distance measure -->
    <!--
       <lst name="spellchecker">
         <str name="name">jarowinkler</str>
         <str name="field">spell</str>
         <str name="classname">solr.DirectSolrSpellChecker</str>
         <str name="distanceMeasure">
           org.apache.lucene.search.spell.JaroWinklerDistance
         </str>
       </lst>
     -->

    <!-- a spellchecker that use an alternate comparator

         comparatorClass be one of:
          1. score (default)
          2. freq (Frequency first, then score)
          3. A fully qualified class name
      -->
    <!--
       <lst name="spellchecker">
         <str name="name">freq</str>
         <str name="field">lowerfilt</str>
         <str name="classname">solr.DirectSolrSpellChecker</str>
         <str name="comparatorClass">freq</str>
      -->

    <!-- A spellchecker that reads the list of words from a file -->
    <!--
       <lst name="spellchecker">
         <str name="classname">solr.FileBasedSpellChecker</str>
         <str name="name">file</str>
         <str name="sourceLocation">spellings.txt</str>
         <str name="characterEncoding">UTF-8</str>
         <str name="spellcheckIndexDir">spellcheckerFile</str>
       </lst>
      -->
  </searchComponent>

  <!-- A request handler for demonstrating the spellcheck component.

       NOTE: This is purely as an example.  The whole purpose of the
       SpellCheckComponent is to hook it into the request handler that
       handles your normal user queries so that a separate request is
       not needed to get suggestions.

       IN OTHER WORDS, THERE IS REALLY GOOD CHANCE THE SETUP BELOW IS
```

```
       NOT WHAT YOU WANT FOR YOUR PRODUCTION SYSTEM!

       See http://wiki.apache.org/solr/SpellCheckComponent for details
       on the request parameters.
    -->
  <requestHandler name="/spell" class="solr.SearchHandler" startup="lazy">
    <lst name="defaults">
      <str name="df">text</str>
      <!-- Solr will use suggestions from both the 'default' spellchecker
           and from the 'wordbreak' spellchecker and combine them.
           collations (re-written queries) can include a combination of
           corrections from both spellcheckers -->
      <str name="spellcheck.dictionary">default</str>
      <str name="spellcheck.dictionary">wordbreak</str>
      <str name="spellcheck">on</str>
      <str name="spellcheck.extendedResults">true</str>
      <str name="spellcheck.count">10</str>
      <str name="spellcheck.alternativeTermCount">5</str>
      <str name="spellcheck.maxResultsForSuggest">5</str>
      <str name="spellcheck.collate">true</str>
      <str name="spellcheck.collateExtendedResults">true</str>
      <str name="spellcheck.maxCollationTries">10</str>
      <str name="spellcheck.maxCollations">5</str>
    </lst>
    <arr name="last-components">
      <str>spellcheck</str>
    </arr>
  </requestHandler>

  <!-- Term Vector Component

       http://wiki.apache.org/solr/TermVectorComponent
    -->
  <searchComponent name="tvComponent" class="solr.TermVectorComponent"/>

  <!-- A request handler for demonstrating the term vector component

       This is purely as an example.

       In reality you will likely want to add the component to your
       already specified request handlers.
    -->
  <requestHandler name="/tvrh" class="solr.SearchHandler" startup="lazy">
    <lst name="defaults">
      <str name="df">text</str>
      <bool name="tv">true</bool>
    </lst>
    <arr name="last-components">
      <str>tvComponent</str>
    </arr>
  </requestHandler>

  <!-- Clustering Component
```

```
       You'll need to set the solr.clustering.enabled system property
       when running solr to run with clustering enabled:

            java -Dsolr.clustering.enabled=true -jar start.jar

       http://wiki.apache.org/solr/ClusteringComponent
       http://carrot2.github.io/solr-integration-strategies/
    -->
  <searchComponent name="clustering"
                   enable="${solr.clustering.enabled:false}"
                   class="solr.clustering.ClusteringComponent" >
    <lst name="engine">
      <str name="name">lingo</str>

      <!-- Class name of a clustering algorithm compatible with the Carrot2 fra
mework.

           Currently available open source algorithms are:
           * org.carrot2.clustering.lingo.LingoClusteringAlgorithm
           * org.carrot2.clustering.stc.STCClusteringAlgorithm
           * org.carrot2.clustering.kmeans.BisectingKMeansClusteringAlgorithm

           See http://project.carrot2.org/algorithms.html for more information.

           A commercial algorithm Lingo3G (needs to be installed separately) is
defined as:
           * com.carrotsearch.lingo3g.Lingo3GClusteringAlgorithm
        -->
      <str name="carrot.algorithm">org.carrot2.clustering.lingo.LingoClustering
Algorithm</str>

      <!-- Override location of the clustering algorithm's resources
           (attribute definitions and lexical resources).

           A directory from which to load algorithm-specific stop words,
           stop labels and attribute definition XMLs.

           For an overview of Carrot2 lexical resources, see:
           http://download.carrot2.org/head/manual/#chapter.lexical-resources

           For an overview of Lingo3G lexical resources, see:
           http://download.carrotsearch.com/lingo3g/manual/#chapter.lexical-res
ources
       -->
      <str name="carrot.resourcesDir">clustering/carrot2</str>
    </lst>

    <!-- An example definition for the STC clustering algorithm. -->
    <lst name="engine">
      <str name="name">stc</str>
      <str name="carrot.algorithm">org.carrot2.clustering.stc.STCClusteringAlgo
rithm</str>
```

```
    </lst>

    <!-- An example definition for the bisecting kmeans clustering algorithm. -
->
    <lst name="engine">
      <str name="name">kmeans</str>
      <str name="carrot.algorithm">org.carrot2.clustering.kmeans.BisectingKMean
sClusteringAlgorithm</str>
    </lst>
  </searchComponent>

  <!-- A request handler for demonstrating the clustering component

       This is purely as an example.

       In reality you will likely want to add the component to your
       already specified request handlers.
    -->
  <requestHandler name="/clustering"
                  startup="lazy"
                  enable="${solr.clustering.enabled:false}"
                  class="solr.SearchHandler">
    <lst name="defaults">
      <bool name="clustering">true</bool>
      <bool name="clustering.results">true</bool>
      <!-- Field name with the logical "title" of a each document (optional) --
>
      <str name="carrot.title">name</str>
      <!-- Field name with the logical "URL" of a each document (optional) -->
      <str name="carrot.url">id</str>
      <!-- Field name with the logical "content" of a each document (optional)
-->
      <str name="carrot.snippet">features</str>
      <!-- Apply highlighter to the title/ content and use this for clustering.
-->
      <bool name="carrot.produceSummary">true</bool>
      <!-- the maximum number of labels per cluster -->
      <!--<int name="carrot.numDescriptions">5</int>-->
      <!-- produce sub clusters -->
      <bool name="carrot.outputSubClusters">false</bool>

      <!-- Configure the remaining request handler parameters. -->
      <str name="defType">edismax</str>
      <str name="qf">
        text^0.5 features^1.0 name^1.2 sku^1.5 id^10.0 manu^1.1 cat^1.4
      </str>
      <str name="q.alt">*:*</str>
      <str name="rows">10</str>
      <str name="fl">*,score</str>
    </lst>
    <arr name="last-components">
      <str>clustering</str>
    </arr>
```

```
  </requestHandler>

  <!-- Terms Component

       http://wiki.apache.org/solr/TermsComponent

       A component to return terms and document frequency of those
       terms
    -->
  <searchComponent name="terms" class="solr.TermsComponent"/>

  <!-- A request handler for demonstrating the terms component -->
  <requestHandler name="/js" class="org.apache.solr.handler.js.JavaScriptReques
tHandler" startup="lazy"/>
  <requestHandler name="/terms" class="solr.SearchHandler" startup="lazy">
     <lst name="defaults">
      <bool name="terms">true</bool>
      <bool name="distrib">false</bool>
    </lst>
    <arr name="components">
      <str>terms</str>
    </arr>
  </requestHandler>


  <!-- Query Elevation Component

       http://wiki.apache.org/solr/QueryElevationComponent

       a search component that enables you to configure the top
       results for a given query regardless of the normal lucene
       scoring.
    -->
  <searchComponent name="elevator" class="solr.QueryElevationComponent" >
    <!-- pick a fieldType to analyze queries -->
    <str name="queryFieldType">string</str>
    <str name="config-file">elevate.xml</str>
  </searchComponent>

  <!-- A request handler for demonstrating the elevator component -->
  <requestHandler name="/elevate" class="solr.SearchHandler" startup="lazy">
    <lst name="defaults">
      <str name="echoParams">explicit</str>
      <str name="df">text</str>
    </lst>
    <arr name="last-components">
      <str>elevator</str>
    </arr>
  </requestHandler>

  <!-- Highlighting Component

       http://wiki.apache.org/solr/HighlightingParameters
```

```
    -->
  <searchComponent class="solr.HighlightComponent" name="highlight">
    <highlighting>
      <!-- Configure the standard fragmenter -->
      <!-- This could most likely be commented out in the "default" case -->
      <fragmenter name="gap"
                  default="true"
                  class="solr.highlight.GapFragmenter">
        <lst name="defaults">
          <int name="hl.fragsize">100</int>
        </lst>
      </fragmenter>

      <!-- A regular-expression-based fragmenter
           (for sentence extraction)
        -->
      <fragmenter name="regex"
                  class="solr.highlight.RegexFragmenter">
        <lst name="defaults">
          <!-- slightly smaller fragsizes work better because of slop -->
          <int name="hl.fragsize">70</int>
          <!-- allow 50% slop on fragment sizes -->
          <float name="hl.regex.slop">0.5</float>
          <!-- a basic sentence pattern -->
          <str name="hl.regex.pattern">[-\w ,/\n\&amp;quot;&amp;apos;]{20,200}</str>
        </lst>
      </fragmenter>

      <!-- Configure the standard formatter -->
      <formatter name="html"
                 default="true"
                 class="solr.highlight.HtmlFormatter">
        <lst name="defaults">
          <str name="hl.simple.pre"><![CDATA[<em>]]></str>
          <str name="hl.simple.post"><![CDATA[</em>]]></str>
        </lst>
      </formatter>

      <!-- Configure the standard encoder -->
      <encoder name="html"
               class="solr.highlight.HtmlEncoder" />

      <!-- Configure the standard fragListBuilder -->
      <fragListBuilder name="simple"
                       class="solr.highlight.SimpleFragListBuilder"/>

      <!-- Configure the single fragListBuilder -->
      <fragListBuilder name="single"
                       class="solr.highlight.SingleFragListBuilder"/>

      <!-- Configure the weighted fragListBuilder -->
      <fragListBuilder name="weighted"
                       default="true"
```

```
                        class="solr.highlight.WeightedFragListBuilder"/>

      <!-- default tag FragmentsBuilder -->
      <fragmentsBuilder name="default"
                        default="true"
                        class="solr.highlight.ScoreOrderFragmentsBuilder">
        <!--
        <lst name="defaults">
          <str name="hl.multiValuedSeparatorChar">/</str>
        </lst>
        -->
      </fragmentsBuilder>

      <!-- multi-colored tag FragmentsBuilder -->
      <fragmentsBuilder name="colored"
                        class="solr.highlight.ScoreOrderFragmentsBuilder">
        <lst name="defaults">
          <str name="hl.tag.pre"><![CDATA[
               <b style="background:yellow">,<b style="background:lawgreen">,
               <b style="background:aquamarine">,<b style="background:magenta">
,
               <b style="background:palegreen">,<b style="background:coral">,
               <b style="background:wheat">,<b style="background:khaki">,
               <b style="background:lime">,<b style="background:deepskyblue">]]
></str>
          <str name="hl.tag.post"><![CDATA[</b>]]></str>
        </lst>
      </fragmentsBuilder>

      <boundaryScanner name="default"
                       default="true"
                       class="solr.highlight.SimpleBoundaryScanner">
        <lst name="defaults">
          <str name="hl.bs.maxScan">10</str>
          <str name="hl.bs.chars">.,!? &#9;&#10;&#13;</str>
        </lst>
      </boundaryScanner>

      <boundaryScanner name="breakIterator"
                       class="solr.highlight.BreakIteratorBoundaryScanner">
        <lst name="defaults">
          <!-- type should be one of CHARACTER, WORD(default), LINE and SENTENC
E -->
          <str name="hl.bs.type">WORD</str>
          <!-- language and country are used when constructing Locale object.
-->
          <!-- And the Locale object will be used when getting instance of Brea
kIterator -->
          <str name="hl.bs.language">en</str>
          <str name="hl.bs.country">US</str>
        </lst>
      </boundaryScanner>
    </highlighting>
```

```
  </searchComponent>

  <!-- Update Processors

       Chains of Update Processor Factories for dealing with Update
       Requests can be declared, and then used by name in Update
       Request Processors

       http://wiki.apache.org/solr/UpdateRequestProcessor

    -->
  <!-- Deduplication

       An example dedup update processor that creates the "id" field
       on the fly based on the hash code of some other fields.  This
       example has overwriteDupes set to false since we are using the
       id field as the signatureField and Solr will maintain
       uniqueness based on that anyway.

    -->
  <!--
     <updateRequestProcessorChain name="dedupe">
       <processor class="solr.processor.SignatureUpdateProcessorFactory">
         <bool name="enabled">true</bool>
         <str name="signatureField">id</str>
         <bool name="overwriteDupes">false</bool>
         <str name="fields">name,features,cat</str>
         <str name="signatureClass">solr.processor.Lookup3Signature</str>
       </processor>
       <processor class="solr.LogUpdateProcessorFactory" />
       <processor class="solr.RunUpdateProcessorFactory" />
     </updateRequestProcessorChain>
    -->

  <!-- Language identification

       This example update chain identifies the language of the incoming
       documents using the langid contrib. The detected language is
       written to field language_s. No field name mapping is done.
       The fields used for detection are text, title, subject and description,
       making this example suitable for detecting languages form full-text
       rich documents injected via ExtractingRequestHandler.
       See more about langId at http://wiki.apache.org/solr/LanguageDetection
    -->
    <!--
     <updateRequestProcessorChain name="langid">
       <processor class="org.apache.solr.update.processor.TikaLanguageIdentifie
rUpdateProcessorFactory">
         <str name="langid.fl">text,title,subject,description</str>
         <str name="langid.langField">language_s</str>
         <str name="langid.fallback">en</str>
       </processor>
       <processor class="solr.LogUpdateProcessorFactory" />
```

```
       <processor class="solr.RunUpdateProcessorFactory" />
     </updateRequestProcessorChain>
    -->

  <!-- Script update processor

    This example hooks in an update processor implemented using JavaScript.

    See more about the script update processor at http://wiki.apache.org/solr/S
criptUpdateProcessor
  -->
  <!--
    <updateRequestProcessorChain name="script">
      <processor class="solr.StatelessScriptUpdateProcessorFactory">
        <str name="script">update-script.js</str>
        <lst name="params">
          <str name="config_param">example config parameter</str>
        </lst>
      </processor>
      <processor class="solr.RunUpdateProcessorFactory" />
    </updateRequestProcessorChain>
  -->

  <!-- Response Writers

       http://wiki.apache.org/solr/QueryResponseWriter

       Request responses will be written using the writer specified by
       the 'wt' request parameter matching the name of a registered
       writer.

       The "default" writer is the default and will be used if 'wt' is
       not specified in the request.
    -->
  <!-- The following response writers are implicitly configured unless
       overridden...
    -->
  <!--
     <queryResponseWriter name="xml"
                          default="true"
                          class="solr.XMLResponseWriter" />
     <queryResponseWriter name="json" class="solr.JSONResponseWriter"/>
     <queryResponseWriter name="python" class="solr.PythonResponseWriter"/>
     <queryResponseWriter name="ruby" class="solr.RubyResponseWriter"/>
     <queryResponseWriter name="php" class="solr.PHPResponseWriter"/>
     <queryResponseWriter name="phps" class="solr.PHPSerializedResponseWriter"/
>
     <queryResponseWriter name="csv" class="solr.CSVResponseWriter"/>
     <queryResponseWriter name="schema.xml" class="solr.SchemaXmlResponseWriter
"/>
    -->

  <queryResponseWriter name="json" class="solr.JSONResponseWriter">
```

```
     <!-- For the purposes of the tutorial, JSON responses are written as
      plain text so that they are easy to read in *any* browser.
      If you expect a MIME type of "application/json" just remove this override
.
     -->
    <str name="content-type">text/plain; charset=UTF-8</str>
  </queryResponseWriter>

  <!--
     Custom response writers can be declared as needed...
    -->
    <queryResponseWriter name="velocity" class="solr.VelocityResponseWriter" st
artup="lazy"/>


  <!-- XSLT response writer transforms the XML output by any xslt file found
       in Solr's conf/xslt directory.  Changes to xslt files are checked for
       every xsltCacheLifetimeSeconds.
    -->
  <queryResponseWriter name="xslt" class="solr.XSLTResponseWriter">
    <int name="xsltCacheLifetimeSeconds">5</int>
  </queryResponseWriter>

  <!-- Query Parsers

       http://wiki.apache.org/solr/SolrQuerySyntax

       Multiple QParserPlugins can be registered by name, and then
       used in either the "defType" param for the QueryComponent (used
       by SearchHandler) or in LocalParams
    -->
  <!-- example of registering a query parser -->
  <!--
     <queryParser name="myparser" class="com.mycompany.MyQParserPlugin"/>
    -->

  <!-- Function Parsers

       http://wiki.apache.org/solr/FunctionQuery

       Multiple ValueSourceParsers can be registered by name, and then
       used as function names when using the "func" QParser.
    -->
  <!-- example of registering a custom function parser  -->
  <!--
     <valueSourceParser name="myfunc"
                        class="com.mycompany.MyValueSourceParser" />
    -->


  <!-- Document Transformers
       http://wiki.apache.org/solr/DocTransformers
    -->
```

```
  <!--
     Could be something like:
     <transformer name="db" class="com.mycompany.LoadFromDatabaseTransformer" >
       <int name="connection">jdbc://....</int>
     </transformer>

     To add a constant value to all docs, use:
     <transformer name="mytrans2" class="org.apache.solr.response.transform.Val
ueAugmenterFactory" >
       <int name="value">5</int>
     </transformer>

     If you want the user to still be able to change it with _value:something_
use this:
     <transformer name="mytrans3" class="org.apache.solr.response.transform.Val
ueAugmenterFactory" >
       <double name="defaultValue">5</double>
     </transformer>

      If you are using the QueryElevationComponent, you may wish to mark docume
nts that get boosted.  The
      EditorialMarkerFactory will do exactly that:
     <transformer name="qecBooster" class="org.apache.solr.response.transform.E
ditorialMarkerFactory" />
    -->


  <!-- Legacy config for the admin interface -->
  <admin>
    <defaultQuery>*:*</defaultQuery>
  </admin>

</config>
```