LuceneFAQ

This is the official Lucene FAQ.

If you have a question about using Java Lucene, please do not add it directly to this FAQ. Join the Java User mailing list and email your question there.

Questions should only be added to this Wiki page when they already have an answer that can be added at the same time.

Lucene FAQ

General

How do I start using Lucene?

Lucene has no external dependencies, so just add lucene-core-x.y-dev.jar to your development environment's classpath. After that,

- read the Javadoc introduction
- if you want to try the demos, also read the Getting Started Guide

If you think Lucene is too low-level for you, you might want to consider using Solr, which usually requires less Java programming.

Are there any mailing lists available?

There's a user list and a developer list, both available at http://lucene.apache.org/java/docs/mailinglists.html .

What Java version is required to run Lucene?

See Lucene System Requirements for most recent Lucene versions

- Lucene 9 requires Java 11
- Lucene 8.8.2 requires Java 8 or greater
- Lucene 7.7.3 requires Java 8 or greater
- Lucene >= 1.9 requires Java 1.4
- Lucene 1.4 will run with JDK 1.3 and up but requires at least JDK 1.4 to compile.

Will Lucene work with my Java application?

Yes, Lucene is 100% pure Java and has no external dependencies.

How can I get the latest greatest development code?

See Lucene Dev Source Code

Where can I get the javadocs for the org.apache.lucene classes?

The docs for all the classes are available online at http://lucene.apache.org/java/docs/api/index.html. In addition, they are a part of the standard distribution, and you can always recreate them by running `ant javadocs`.

Where does the name Lucene come from?

Lucene is Doug Cutting's wife's middle name, and her maternal grandmother's first name.

Are there any alternatives to Lucene?

Besides commercial products which we don't know much about there a variety of other open source search engines available. We suggest searching the web to find out more. Also check the list of Lucene implemenations.

How does Zend Search Lucene (and other non-ASF ports) Relate to Lucene?

Zend Search Lucene is not at all related to the Apache Lucene project, despite the attempt to relate itself to the Lucene project via its name. It, and other attempts at porting Lucene to other languages, outside of the ASF are not supported by the ASF. The ASF currently supports ports of Lucene to Python and .NET. As far as we can tell, Zend Search Lucene was at one point in time a Lucene index file format compatible implementation of a search engine in PHP. Its performance characteristics and APIs are significantly different from Apache Lucene. If you are using Zend Search Lucene, please ask your questions about it in the Zend forums.

Does Lucene have a web crawler?

No, but check out Nutch and the list of Open Source Crawlers in Java.

Why am I getting an IOException that says "Too many open files"?

The number of files that can be opened simultaneously is a system-wide limitation of your operating system. Lucene might cause this problem as it can open quite some files depending on how you use it, but the problem might also be somewhere else.

- Always make sure that you *explicitly* close all file handles you open, especially in case of errors. Use a try/catch/finally block to open the files, i.e. open them in the try block, close them in the finally block. Remember that Java doesn't have destructors, so don't close file handles in a finalize method – this method is not guaranteed to be executed.
- Use the compound file format (it's activated by default starting with Lucene 1.4) by calling IndexWriter's setUseCompoundFile(true)
- Don't set IndexWriter's mergeFactor to large values. Large values speed up indexing but increase the number of files that need to be opened simultaneously.
- Make sure you only open one IndexSearcher, and share it among all of the threads that are doing searches – this is safe, and it will minimize the number of files that are open concurently.
- Try to increase the number of files that can be opened simultaneously. On Linux using bash this can be done by calling `ulimit -n <number>`.

When I compile Lucene x.y.z from source, the version number in the jar file name and MANIFEST.MF is different. What's up with that?

This is intentional. Only the jar files produced by the Lucene release manager will have the exact release number. Any other builds will have a different release number in order to help differentiate them from the code produced by the release process. Feel free to adjust.

How do I contribute an improvement?

Please follow all of these steps to submit a Lucene patch.

Why hasn't patch FOO been committed?

Committers are at their own discretion to decide what patches are suitable for being committed. Generally speaking, committers are encouraged to be conservative about what patches they commit. By committing code into the code base, he or she vouches for the quality of that patch. Any problems that ensue are, to some degree, the responsibility of that committer. If a committer does not feel comfortable making changes to particular sections of the code base, they may wish to consult (or defer to) a more senior committer.

The best way to encourage committers to commit a particular patch is to make it easy to apply. At a minimum it should apply easily to trunk and pass all unit tests. It should confine itself to a single issue: changing as little as possible; adding as little as possible. The patch should include new unit tests which demonstrate the bug the patch fixes (or the new functionality the patch adds). The case is stronger if others report to have successfully applied the patch and found it useful.

If one feels a patch is neglected one should be persistent, polite and patient.

What are the backwards compatibility commitments?

Here are the compatibility commitments.

How do I get code written for Lucene 1.4.x to work with Lucene 2.x?

The upgrade path for Lucene 2.0 was designed around the notion of clear deprecation warnings. Any code designed to use the APIs in Lucene 1.4.x should compile/function with Lucene 1.9 – however many compile time deprecation warnings will be generated identifying methods that should no longer be used, and what new methods should be used instead.

If you have code that worked with Lucene 1.4.x, and you want to "port" it to Lucene 2.x you should start by downloading the 1.9 release of Lucene, and compile the code against it. Make sure deprecation warnings are turned on in your development environment, and gradually change your code until all deprecation warnings go away (the DateField class is an exception, it has not been removed in Lucene 2.0 yet).

At that point, your code should work fine with Lucene 2.x.

If you are looking at example code (in an article or book perhaps) and just need to understand how the example would change to work with 2.0 (without needing to actually compile it) you can review the javadocs for Lucene 1.9 and lookup any methods used in the examples that are no longer part of Lucene. The 1.9 javadocs will have a clear deprecation message explaining how to get the same effect using the 2.x methods.

How is Lucene's indexing and search performance measured?

Check Lucene bench: https://home.apache.org/~mikemccand/lucenebench/

I am having a performance issue. How do I ask for help on the java-user@lucene.apache.org mailing list?

1. Make sure you have looked through the BasicsOfPerformance
2. Describe your problem, giving details about how you are using Lucene
3. What version of Lucene are you using? What JDK? Can you upgrade to the latest?
4. Make sure it truly is a Lucene problem. That is, isolate the problem and/or profile your application.
5. Search the java-user and java-dev Mailing lists, see http://lucene.apache.org/java/docs/mailinglists.html

What does l.a.o and o.a.l.xxxx stand for?

l.a.o is shorthand for lucene.apache.org (ie: the website)

o.a.l.xxxx is shorthand for org.apache.lucene.xxxx (ie: the java package namespace)

What is the difference between field (or document) boosting and query boosting?

Index time field boosts (field.setBoost(boost)) are a way to express things like "this document's title is worth twice as much as the title of most documents". Query time boosts (query.setBoost(boost)) are a way to express "I care about matches on this clause of my query twice as much as I do about matches on other clauses of my query".

Index time field boosts are worthless if you set them on every document.

Index time document boosts (doc.setBoost(float)) are equivalent to setting a field boost on ever field in that document.

Searching

Does Lucene allow searching and indexing simultaneously?

Yes. However, an IndexReader only searches the index as of the "point in time" that it was opened. Any updates to the index, either added or deleted documents, will not be visible until the IndexReader is re-opened. So your application must periodically re-open its IndexReaders to see the latest updates. The IndexReader.isCurrent() method allows you to test whether any updates have occurred to the index since your IndexReader was opened.

Why am I getting no hits / incorrect hits?

Some possible causes:

- The desired term is in a field that was not defined as 'indexed'. Re-index the document and make the field indexed.
- The term is in a field that was not tokenized during indexing and therefore, the entire content of the field was considered as a single term. Re-index the documents and make sure the field is tokenized.
- The field specified in the query simply does not exist. You won't get an error message in this case, you'll just get no matches.
- The field specified in the query has wrong case. Field names are case sensitive.
- The term you are searching is a stop word that was dropped by the analyzer you use. For example, if your analyzer uses the StopFilter, a search for the word 'the' will always fail (i.e. produce no hits).
- You are using different analyzers (or the same analyzer but with different stop words) for indexing and searching and as a result, the same term is transformed differently during indexing and searching.
- The analyzer you are using is case sensitive (e.g. it does not use the LowerCaseFilter) and the term in the query has different case than the term in the document.
- The documents you are indexing are very large. Lucene by default only indexes the first 10,000 terms of a document to avoid OutOfMemory errors. See IndexWriter.setMaxFieldLength(int).
- Make sure to open a new IndexSearcher after adding documents. An IndexSearcher will only see the documents that were in the index when it was opened.
- If you are using the QueryParser, it may not be parsing your BooleanQuerySyntax the way you think it is.
- Span and phrase queries won't work if omitTf() has been called for a field since that causes positional information about tokens to not be saved in the index. Span queries & phrase queries require the positional information in order to work.

If none of the possible causes above apply to your case, this will help you to debug the problem:

- Use the Query's toString() method to see how it actually got parsed.
- Use Luke to browse your index: on the "Documents" tab, navigate to a document, then use the "Reconstruct & Edit" to look at how the fields have been stored ("Stored original" tab) and indexed ("Tokenized" tab)

Why am I getting a TooManyClauses exception?

The following types of queries are expanded by Lucene before it does the search: RangeQuery, PrefixQuery, WildcardQuery, FuzzyQuery. For example, if the indexed documents contain the terms "car" and "cars" the query "ca*" will be expanded to "car OR cars" before the search takes place. The number of these terms is limited to 1024 by default. Here's a few different approaches that can be used to avoid the TooManyClauses exception:

- Use a filter to replace the part of the query that causes the exception. For example, a RangeFilter can replace a RangeQuery on date fields and it will never throw the TooManyClauses exception – You can even use ConstantScoreRangeQuery to execute your RangeFilter as a Query. Note that filters are slower than queries when used for the first time, so you should cache them using CachingWrapperFilter. Using Filters in place of Queries generated by QueryParser can be achieved by subclassing QueryParser and overriding the appropriate function to return a ConstantScore version of your Query.
- Increase the number of terms using BooleanQuery.setMaxClauseCount(). Note that this will increase the memory requirements for searches that expand to many terms. To deactivate any limits, use BooleanQuery.setMaxClauseCount(Integer.MAX_VALUE).

- A specfic solution that can work on very precise fields is to reduce the precision of the data in order to reduce the number of terms in the index. For example, the DateField class uses a microsecond resultion, which is often not required. Instead you can save your dates in the "yyyymmddHHMM" format, maybe even without hours and minutes if you don't need them (this was simplified in Lucene 1.9 thanks to the new DateTools class).

How can I search over multiple fields?

Searching over multiple fields is what people expect as Google searches all the fields by default. You have to parse the query using MultiFieldQueryParser. Note that terms which occur in short fields have a higher effect on the result ranking.

Alternatively you could create a field which concatenates the content you would like to search and search only that field.

What wildcard search support is available from Lucene?

Lucene supports wild card queries which allow you to perform searches such as *book**, which will find documents containing terms such as *book*, *bookstore* , *booklet*, etc. Lucene refers to this type of a query as a 'prefix query'.

Lucene also supports wild card queries which allow you to place a wild card in the middle of the query term. For instance, you could make searches like: *mi *pelling*. That will match both *misspelling*, which is the correct way to spell this word, as well as *mispelling*, which is a common spelling mistake.

Another wild card character that you can use is '?', a question mark. The ? will match a single character. This allows you to perform queries such as *Bra?il*. Such a query will match both *Brasil* and *Brazil*. Lucene refers to this type of a query as a 'wildcard query'.

Leading wildcards (e.g. **ook*) are **not** supported by the QueryParser by default. As of Lucene 2.1, they can be enabled by calling `QueryParser.setAllowLeadingWildcard( true )`. Note that this can be an expensive operation: it requires scanning the list of tokens in the index in its entirety to look for those that match the pattern.

Can I combine wildcard and phrase search, e.g. "foo ba*"?

This is not supported by QueryParser, but you could extend the QueryParser to build a MultiPhraseQuery in those cases.

Is the QueryParser thread-safe?

No, it's not.

How do I restrict searches to only return results from a limited subset of documents in the index (e.g. for privacy reasons)? What is the best way to approach this?

The QueryFilter class is designed precisely for such cases.

Another way of doing it is the following:

Just before calling `IndexSearcher.search()` add a clause to the query to exclude documents in categories not permitted for this search.

If you are restricting access with a prohibited term, and someone tries to require that term, then the prohibited restriction wins. If you are restricting access with a required term, and they try prohibiting that term, then they will get no documents in their search result.

As for deciding whether to use required or prohibited terms, if possible, you should choose the method that names the less frequent term. That will make queries faster.

What is the order of fields returned by Document.fields()?

Fields are returned in the same order they were added to the document.

*NOTE:*This functionality was broken in 2.3 and 2.4, but will be fixed in 2.9; see LUCENE-1727

How does one determine which documents do not have a certain term?

There is no direct way of doing that. You could add a term "x" to every document, and then search for "+x -y" to find all of the documents that don't have "y". Note that for large collections this would be slow because of the high term frequency for term "x".

Lucene 1.9 added MatchAllDocsQuery to make this easier.

How do I get the last document added that has a particular term?

Call:

```
TermDocs td = IndexReader.termDocs(Term);
```

Then grab the last `Term` in `TermDocs` that this method returns.

Does MultiSearcher do anything particularly efficient to search multiple indices or does it simply search one after the other?

`MultiSearcher` searches indices sequentially. Use ParallelMultiSearcher as a searcher that performs multiple searches in parallel. Please note that there's a known bug in Lucene < 1.9 in the MultiSearcher's result ranking.

Is there a way to use a proximity operator (like near or within) with Lucene?

There is a variable called `slop` in `PhraseQuery` that allows you to perform NEAR/WITHIN-like queries.

By default, `slop` is set to 0 so that only exact phrases will match. However, you can alter the value using the `setSlop(int)` method.

When using QueryParser you can use this syntax to specify the slop: "doug cutting"~2 will find documents that contain "doug cutting" as well as ones that contain "cutting doug".

Are Wildcard, Prefix, and Fuzzy queries case sensitive?

No, not by default. Unlike other types of Lucene queries, Wildcard, Prefix, and Fuzzy queries are not passed through the `Analyzer`, which is the component that performs operations such as stemming and lowercasing. The reason for skipping the `Analyzer` is that if you were searching for *"dogs*"* you would not want *"dogs"* first stemmed to *"dog"*, since that would then match *"dog*"*, which is not the intended query. These queries are case-insensitive anyway because `QueryParser` makes them lowercase. This behavior can be changed using the setLowercaseExpandedTerms(boolean) method.

Why does IndexReader's maxDoc() return an 'incorrect' number of documents sometimes?

According to the Javadoc for `IndexReader` `maxDoc()` method *"returns one greater than the largest possible document number"*.

In other words, the number returned by `maxDoc()` does not necessarily match the actual number of undeleted documents in the index.

Deleted documents do not get removed from the index immediately, until they are merged away.

Is there a way to get a text summary of an indexed document with Lucene (a.k.a. a "snippet" or "fragment") to display along with the search result?

You need to store the documents' summary in the index (use Field.Store.YES when creating that field) and then use the Highlighter from the contrib area (distributed with Lucene since version 1.9 as "lucene-highlighter-(version).jar"). It's important to use a rewritten query as the input for the highlighter, i.e. call rewrite() on the query. Otherwise simple queries will work but prefix queries etc will not be highlighted.

For Lucene < 1.9, you can also get the "highlighter-dev.jar" from http://www.lucenebook.com/LuceneInAction.zip. See http://www.gossamer-threads.com/lists/lucene/java-user/31595 for a discussion of this.

Can I cache search results with Lucene?

Lucene does come with a simple cache mechanism, if you use Lucene Filters . The classes to look at are CachingWrapperFilter and QueryFilter.

Also consider using a JSP tag for caching, see http://www.opensymphony.com/oscache/ for one tab library that's easy and works well.

Is the IndexSearcher thread-safe?

Yes, IndexSearcher is thread-safe. Multiple search threads may use the same instance of IndexSearcher concurrently without any problems. It is recommended to use only one IndexSearcher from all threads in order to save memory.

Is there a way to retrieve the original term positions during the search?

Yes, see the Javadoc for IndexReader.termPositions().

How do I retrieve all the values of a particular field that exists within an index, across all documents?

The trick is to enumerate terms with that field. Terms are sorted first by field, then by text, so all terms with a given field are adjacent in enumerations. Term enumeration is also efficient.

```
try
{
    TermEnum terms = indexReader.terms(new Term("FIELD-NAME-HERE", ""));
    while ("FIELD-NAME-HERE".equals(terms.term().field()))
    {
        // ... collect terms.term().text() ...

        if (!terms.next())
            break;
    }
}
finally
{
    terms.close();
}
```

Can Lucene do a "search within search", so that the second search is constrained by the results of the first query?

Yes. There are two primary options:

- Use `QueryFilter` with the previous query as the filter. Doug Cutting recommends against this, because a QueryFilter does not affect ranking.
- Combine the previous query with the current query using `BooleanQuery`, using the previous query as required.

The `BooleanQuery` approach is the recommended one.

Does the position of the matches in the text affect the scoring?

No, the position of matches within a field does not affect ranking.

How do I make sure that a match in a document title has greater weight than a match in a document body?

If you put the title in a separate field from the body, and search both fields, matches in the title will usually be stronger without explicit boosting. This is because the scores are normalized by the length of the field, and the title tends to be much shorter than the body. Therefore, even without boosting, title matches usually come before body matches. But you can also boost queries on title by using query.setBoost(boost) on the relevant clause.

How do I find similar documents?

See the MoreLikeThis class in the org.apache.lucene.search.similar package. In Lucene 1.9 it was in the "similarity" contrib jar, but starting with Lucene 2.1 it was moved to the new "queries" contrib.

Can I filter by score?

Not safely. You can always pick an arbitrary score value and then check the Hits object to see how many results have a score higher than that value (a Binary search might come in handy) but it really doesn't give you any meaningful information because of the way score is calculated...

One Explanation...

```
  > Does anyone have an example of limiting results returned based on a
  > score threshold? For example if I'm only interested in documents with
  > a score > 0.05.

I would not recommend doing this because absolute score values in Lucene
are not meaningful (e.g., scores are not directly comparable across
searches).  The ratio of a score to the highest score returned is
meaningful, but there is no absolute calibration for the highest score
returned, at least at present, so there is not a way to determine from
the scores what the quality of the result set is overall.
```

For more detailed discussion, please read ScoresAsPercentages

How can I cluster results, i.e. create groups of similar documents?

Check out Carrot, a clustering framework that can be used with Lucene.

How do I implement paging, i.e. showing result from 1-10, 11-20 etc?

Just re-execute the search and ignore the hits you don't want to show. As people usually look only at the first results this approach is usually fast enough.

How do I speed up searching?

See ImproveSearchingSpeed.

Does Lucene support Approximate Nearest Neighbor (ANN) / k-Nearest Neighbors (k-NN) search?

Yes, Lucene 9 and greater support ANN / kNN search, whereas see

- https://lucene.apache.org/core/9_1_0/demo/index.html#Embeddings
- https://lucene.apache.org/core/9_1_0/demo/org/apache/lucene/demo/knn/package-summary.html
- **LUCENE-9004** - Getting issue details... STATUS

Does Lucene support auto-suggest / autocomplete?

Yes, see https://lucene.apache.org/core/9_4_2/suggest/index.html

What is the default relevance / similarity implementation of Lucene?

The default implementation to determine the relevance of documents for a particular query is based on BM25.

Also see IndexSearcher#getDefaultSimilarity()

Indexing

Can I use Lucene to crawl my site or other sites on the Internet?

No. Lucene does not know how to access external document, nor does it know how to extract the content and links of HTML and other document format. Lucene focuses on the indexing and searching and does it great. However, several crawlers are available which you could use: list of Open Source Crawlers in Java. regain is an Open Source tool that crawls web sites, stores them in a Lucene index and offers a search web interface. Also see Nutch for a powerful Lucene-based search engine.

How can I use Lucene to index a database?

Connect to the database using JDBC and use an SQL "SELECT" statement to query the database. Then create one Lucene Document object per row and add it to the index. You will probably want to store the ID column so you can later access the matching items. For other (text) columns it might make more sense to only index (not store) them, as the original data is still available in your database.

For a more high level approach you might want to have a look at LuSql (a specialized tool for moving data from JDBC-accessible databases into Lucene), Hibernate Search, Compass, DBSight, or Solr's Data Import Handler which all use Lucene internally.

How do I perform a simple indexing of a set of documents?

The easiest way is to re-index the entire document set periodically or whenever it changes. All you need to do is to create an instance of IndexWriter(), iterate over your document set, create for each document a Lucene Document object and add it to the IndexWriter. When you are done make sure to close the IndexWriter. This will release all of its resources and will close the files it created.

How can I add document(s) to the index?

Simply create an IndexWriter and use its addDocument() method. Make sure to create the IndexWriter with the 'create' flag set to false and make sure to close the IndexWriter when you are done adding the documents.

Where does Lucene store the index it builds?

Typically, the index is stored in a set of files that Lucene creates in a directory of your choice. If your system uses multiple independent indices, simply create a separate directory for each index.

Lucene's API also provide a way to use or implement other storage methods such as a in-memory storage (RAMDirectory), or a mapping of Lucene data to any third party database (not included in Lucene).

Can I store the Lucene index in a relational database?

Lucene does not support that functionality out of the box, but several people have implemented JdbcDirectory's. The reports we have seen so far indicate that performance with such implementations is not great, but it is doable.

Can I store the Lucene index in a BerkeleyDB?

Yes, you use BerkeleyDB as the Lucene index store. Just use DbDirectory implementation from Lucene's contrib section.

I get "No tvx file". What does that mean?

It's a "warning" that can safely be ignored. It has been fixed (i.e. the warning has been removed) in Lucene 1.9.

Does Lucene store a full copy of the indexed documents?

It is up to you. You can tell Lucene what document information to use just for indexing and what document information to also store in the index (with or without indexing).

What is the different between Stored, Tokenized, Indexed, and Vector?

- Stored = as-is value stored in the Lucene index
- Tokenized = field is analyzed using the specified Analyzer - the tokens emitted are indexed
- Indexed = the text (either as-is with keyword fields, or the tokens from tokenized fields) is made searchable (aka inverted)
- Vectored = term frequency per document is stored in the index in an easily retrievable fashion.

What happens when you IndexWriter.add() a document that is already in the index? Does it overwrite the previous document?

No, there will be multiple copies of the same document in the index.

How do I delete documents from the index?

`IndexWriter` allows you to delete by `Term` or by `Query`. The deletes are buffered and then periodically flushed to the index, and made visible once `commit()` or `close()` is called.

If you would like to delete documents by document number, `IndexWriter` provides `tryDeleteDocument`. Note however that this method only succeeds if the segment where the doc ID belongs has not been merged away. It is generally preferred to use a primary key field that holds a unique ID for each document and to use this field to delete by `Term` by passing it to `IndexWriter`'s `deleteDocuments(Term)` method.

Once a document is deleted it will not appear in search results. The presence of this document may still be reflected in the `docFreq` statistics, and thus alter search scores, though this will be corrected eventually as segments containing deletions are merged.

Is there a way to limit the size of an index?

This question is sometimes brought up because of the 2GB file size limit of some 32-bit operating systems.

This is a slightly modified answer from Doug Cutting:

The easiest thing is to use `IndexWriter.setMaxMergeDocs()`.

If, for instance, you hit the 2GB limit at 8M documents set `maxMergeDocs` to 7M. That will keep Lucene from trying to merge an index that won't fit in your filesystem. It will actually effectively round this down to the next lower power of `Index.mergeFactor`.

So with the default `mergeFactor` set to 10 and `maxMergeDocs` set to 7M Lucene will generate a series of 1M document indexes, since merging 10 of these would exceed the maximum.

Why is it important to use the same analyzer type during indexing and search?

The analyzer controls how the text is broken into terms which are then used to index the document. If you are using an analyzer of one type to index and an analyzer of a different type to parse the search query, it is possible that the same word will be mapped to two different terms and this will result in missing or false hits.

NOTE: It's not a rule that the same analyzer be used for both indexing and searching, and there are cases where it makes sense to use different ones (ie: when dealing with synonyms). The analyzers must be compatible though.

Also be careful with Fields that are not tokenized (like Keywords). During indexation, the Analyzer won't be called for these fields, but for a search, the QueryParser can't know this and will pass all search strings through the selected Analyzer. Usually searches for Keywords are constructed in code, but during development it can be handy to use general purpose tools (e.g. Luke) to examine your index. Those tools won't know which fields are tokenized either. In the contrib/analyzers area there's a KeywordTokenizer with an example KeywordAnalyzer for cases like this.

What are Segments?

The index database is composed of 'segments' each stored in a separate file. When you add documents to the index, new segments may be created. These are periodically merged together.

Is Lucene index database platform independent?

Yes, you can copy a Lucene index directory from one platform to another and it will work just as well.

When I recreate an index from scratch, do I have to delete the old index files?

No, creating the IndexWriter with "true" should remove all old files in the old index (actually with Lucene < 1.9 it removes **all** files in the index directory, no matter if they belong to Lucene).

When I uprade Lucene, for example from 8.8.2 to 9.0.0, do I have to reindex?

Not necessarily, you can add a version specific lucene-backward-codecs library, for example lucene-backward-codecs-9.0.0.jar will enable Lucene 9 to use the index of the previous version Lucene 8.

But it is recommended to upgrade, because a Lucene 8.x index may not be able to be read by an eventual Lucene 10 release.

Certain changes always require to reindex though, also see https://solr.apache.org/guide/8_0/reindexing.html

How can I index and search digits and other non-alphabetic characters?

The components responsible for this are various `Analyzers`. Make sure you use the appropriate analyzer. For examaple, StandardAnaylzer does not remove numbers, but it removes most punctuation.

Is the [IndexWriter] class, and especially the method addIndexes(Directory[]) thread safe?

Yes, `IndexWriter.addIndexes(Directory[])` method is thread safe (it is a `synchronized` method). IndexWriter in general is thread safe, i.e. you should use the same IndexWriter object from all of your threads. Actually it's impossible to use more than one IndexWriter for the same index directory, as this will lead to an exception trying to create the lock file.

When is it possible for document IDs to change?

Documents can be re-numbered at anytime by Lucene

If you require a persistent document id, then add it as a field to your documents.

What is the purpose of write.lock file, when is it used, and by which classes?

The write.lock is used to keep processes from concurrently attempting to modify an index.

It is obtained by an IndexWriter while it is open, and by an IndexReader once documents have been deleted and until it is closed.

What is the purpose of the commit.lock file, when is it used, and by which classes?

The commit.lock file is used to coordinate the contents of the 'segments' file with the files in the index. It is obtained by an IndexReader before it reads the 'segments' file, which names all of the other files in the index, and until the IndexReader has opened all of these other files.

The commit.lock is also obtained by the IndexWriter when it is about to write the segments file and until it has finished trying to delete obsolete index files.

The commit.lock should thus never be held for long, since while it is obtained files are only opened or deleted, and one small file is read or written.

Note that as of Lucene 2.1, the commit.lock is no longer used. Instead, to prevent contention on the segments file, Lucene writes to segments-N files where each commit increments the N. The write.lock is still used. See LUCENE-701 for details.

My program crashed and now I get a "Lock obtain timed out." error. Where is the lock and how can i delete it?

When using FSDirectory, Lock files are kept in the directory specified by the "org.apache.lucene.lockdir" system property if it is set, or by default in the directory specified by the "java.io.tmpdir" system property (on Unix boxes this is usually "/var/tmp" or "/tmp").

If for some strange reason "java.io.tmpdir" is not set, then the directory path you specified to create your index is used.

Lock files have names that start with "lucene-" followed by an MD5 hash of the index directory path.

If you are certain that a lock file is not in use, you can delete it manually. You should also look at the methods "IndexReader.isLocked" and "IndexReader.unlock" if you are interested in writing recovery code that can remove locks automatically.

Is there a maximum number of segment infos whose summary (name and document count) is stored in the segments file?

All segments in the index are listed in the segments file. There is no hard limit. For a normal index it is proportional to the log of the number of documents in the index.

How do I update a document or a set of documents that are already indexed?

There is no direct update procedure in Lucene. To update an index incrementally you must first **delete** the documents that were updated, and **then re-add** them to the index.

How do I write my own Analyzer?

Here is an example:

```
public class MyAnalyzer extends ReusableAnalyzerBase {
  private Version matchVersion;

  public MyAnalyzer(Version matchVersion) {
    this.matchVersion = matchVersion;
  }

  @Override
  protected TokenStreamComponents createComponents(String fieldName, Reader reader) {
    final Tokenizer source = new WhitespaceTokenizer(matchVersion, reader);
    TokenStream sink = new LowerCaseFilter(matchVersion, source);
    sink = new LengthFilter(sink, 3, Integer.MAX_VALUE);
    return new TokenStreamComponents(source, sink);
  }
}
```

All that being said, most of the heavy lifting in custom analyzers is done by calls to custom subclasses of TokenFilter.

If you want your custom token modification to come after the filters that lucene's StandardAnalyzer class would normally call, do the following:

```
@Override
protected TokenStreamComponents createComponents(String fieldName, Reader reader) {
  final Tokenizer source = new StandardTokenizer(matchVersion, reader);
  TokenStream sink = new StandardFilter(matchVersion, source);
  sink = new LowerCaseFilter(matchVersion, sink);
  sink = new StopFilter(matchVersion, sink,
                        StopAnalyzer.ENGLISH_STOP_WORDS_SET, false);
  sink = new CaseNumberFilter(sink);
  sink = new NameFilter(sink);
  return new TokenStreamComponents(source, sink);
}
```

How do I index non Latin characters?

Lucene only uses Java strings, so you normally do not need to care about this. Just remember that you may need to specify an encoding when you read in external strings from e.g. a file (otherwise the system's default encoding will be used). If you really need to recode a String you can use this hack:

```
String newStr = new String(someString.getBytes("UTF-8"));
```

How can I index HTML documents?

In order to index HTML documents you need to first parse them to extract text that you want to index from them. Have a look at Tika, the content analysis toolkit.

Alternately...

An example that uses JavaCC to parse HTML into Lucene Document objects is provided in the Lucene web application demo that comes with the Lucene distribution.

The CyberNeko HTML Parser lets you parse HTML documents. It's relatively easy to remove most of the tags from an HTML document (or all if you want), and then use the ones you left in to help create metadata for your Lucene document. NekoHTML also provides a DOM model for navigating through the HTML.

JTidy cleans up HTML, and can provide a DOM interface to the HTML files through a Java API.

The author of FURL recommends TagSoup.

Jericho HTML Parser provides a simple TextExtractor class that converts any segment of an HTML document into a string of space-separated words, optionally including the values from title, alt, label, and summary attributes. The parser is also very tolerant of badly formatted HTML and can also handle server-based source tags such as JSP, ASP, PHP etc.

How can I index XML documents?

In order to index XML documents you need to first parse them to extract text that you want to index from them. Have a look at Tika, the content analysis toolkit.

How can I index file formats like OpenDocument (aka OpenOffice.org), RTF, Microsoft Word, Excel, PowerPoint, Visio, etc?

Have a look at Tika, the content analysis toolkit.

Alternately: Many modern office file formats (.odt, .sxw, .sxc, etc) are ZIP archives that contain XML files. You can uncompress the file using Java's ZIP support, then parse e.g. meta.xml to get the title and e.g. content.xml to get the document's content. You can then add these to the Lucene index, typically using one Lucene field per property.

You can also use LIUS framework for indexing OpenOffice.org documents (http://www.bibl.ulaval.ca/lius/). LIUS allows metadata and fulltext indexing, using XPath.

For MS-Word, MS-Excel, MS-Visio, and MS-Powerpoint you might also want to take a look at Apache POI.

Lucene In Action contains an example of how to extract text from RTF files using the Swing RTFEditorKit class.

How can I index Email (from MS-Exchange or another IMAP server) ?

Take a look at:

- http://www.chencer.com/techno/java/lucene/imap.html
- http://zoe.sourceforge.net/

How can I index PDF documents?

In order to index PDF documents you need to first parse them to extract text that you want to index from them. Here are some PDF parsers that can help you with that:

PDFBox is a Java API from Ben Litchfield that will let you access the contents of a PDF document. It comes with integration classes for Lucene to translate a PDF into a Lucene document.

JPedal is a Java API for extracting text and images from PDF documents.

PDFTextStream is a Java API for extracting text, metadata, and form data from PDF documents. It also comes with an integration module making it easier to convert a PDF document into a Lucene document.

XPDF is an open source tool that is licensed under the GPL. It's not a Java tool, but there is a utility called pdftotext that can translate PDF files into text files on most platforms from the command line.

Based on xpdf, there is a utility called pdftohtml that can translate PDF files into HTML files. This is also not a Java application.

How can I index JSP files?

To index the content of JSPs that a user would see using a Web browser, you would need to write an application that acts as a Web client, in order to mimic the Web browser behaviour (i.e. a web crawler). Once you have such an application, you should be able to point it to the desired JSP, retrieve the contents that the JSP generates, parse it, and feed it to Lucene. See list of Open Source Crawlers in Java.

How to parse the output of the JSP depends on the type of content that the JSP generates. In most cases the content is going to be in HTML format.

Most importantly, do not try to index JSPs by treating them as normal files in your file system. In order to index JSPs properly you need to access them via HTTP, acting like a Web client.

How can I index java source files?

There is an article at onjava.com that describes an example on how to index java sources not just as text files, but distinguishing between the different information like superclass, implented interfaces, methods, imported classes etc.

Note that the article uses an older version of apache lucene. For parsing the java source files and extracting that information, the ASTParser of the eclipse java development tools is used.

What is the difference between [IndexWriter].addIndexes(IndexReader[]) and [IndexWriter].addIndexes(Directory[]), besides them taking different arguments?

When merging lots of indexes (more than the mergeFactor), the Directory-based method will use fewer file handles and less memory, as it will only ever open mergeFactor indexes at once, while the IndexReader-based method requires that all indexes be open when passed.

The primary advantage of the IndexReader-based method is that one can pass it IndexReaders that don't reside in a Directory.

Can I use Lucene to index text in Chinese, Japanese, Korean, and other multi-byte character sets?

Yes, you can. Lucene is not limited to English, nor any other language. To index text properly, you need to use an Analyzer appropriate for the language of the text you are indexing. Lucene's default Analyzers work well for English. There are a number of other Analyzers in Lucene Sandbox, including those for Chinese, Japanese, and Korean.

Why do I have a deletable file (and old segment files remain) after merging?

This is normal behavior on Windows whenever you also have readers (IndexReaders or IndexSearchers) open against the index where a IndexWriter is open in parallel that does merges. Lucene tries to remove old segments files once they have been merged. However, because Windows does not allow removing files that are open for reading, Lucene catches an IOException deleting these files and and then records these pending deletable files into the "deletable" file. On the next segments merge, Lucene will try again to delete these files (and additional ones) and any that still fail will be rewritten to the deletable file.

Note that as of 2.1 the deletable file is no longer used. Instead, Lucene computes which files are no longer referenced by the index and removes them whenever a writer is created.

How do I speed up indexing?

See ImproveIndexingSpeed.