On Using the Lucene Based Indexer and Search Application

Version 2.1

Tom Louton, Schering

トム （戸夢）ルートン

Table of Contents

KYUUDOU -- seek enlightenment

Search Engines

In this context, the creation of a search engine is a three step process

1. Indexing
2. Searching
3. Presentation

I will briefly describe each process.

Indexing. This is the fundamental step to all searching. Roughly described as follows: a document is read (parsed) and the text is then divided into tokens (this means really, words -- collections of letters separated by blank spaces). If this is the first document and the first token, the token is added to the "index". Further, to the token the number of the document (in this case 1) is added to the token. If the index already has tokens, then if the token is in the index, the document number of the current document is added to the token, otherwise the token with document number is added to the index. Consider an index as a collection of tokens (words for our purposes) and a vector of document numbers.

The process of tokenizing (extracting the tokens) is also often accompanied by a filtering and transformation process. In general it is useful to change all characters to lower case. Further it might be advantageous

to label the words with their part of speech in the context in which found (POS labeling). So called "stop words" are dropped and not indexed. By stop words we mean words such as "I", "it"(but IT, information technology) , "the", which occur in every document and do not contribute in any way to the being able to fish this document out of a group. Sometimes it is advantages to reduce a word to its stem (grammatically). For example, an adjective big, bigger, biggest would all be reduced to "big", or a verb to go, went, gone, going would all be reduced to "go".

So the tokens and the associated vectors of document numbers are stored in the index. But that is not all. The documents represented as Lucene documents are also stored. This is not the Word, PDF or what have you, document, but an invocation of a Lucene class called document. When indexing, the user (programmer actually) decides what information is to be stored in the Lucene document, in our case for example, the name of the file in which the document is stored, the date it was created, last changed, as well as a sort of summary.

Searching. Searching is then the process of taking a key word or an expression (e.g., "a" AND "b" -- with short cut +a +b) and finding those documents in which it is present. There are many variations and the tools we have at hand allow simple key word searches (where there is just one key word), so called query searches (this allows a more complex syntax involving more than one key word. The query search also allows wild card searches. Query searches are the only type of search realized in this system.

Range searches, for example with dates and such, have not been realized, but could be. Also phrase searches and fuzzy searches (where tokens which in some way are "near" the desired word) are not realized -- Query searches support fuzzy searches and phrase searches to an extent. See the chapter on searching.

A search results in a collection of documents or in the case we have, references to documents.

Presentation. The step of presenting the information to the user involves deciding which information contained in the documents won is to be presented and how. Recall, that the information retrieved from a search is not the physical document, but the parsed information which is stored in the index. The user is offered an HTML file containing salient information and a reference to the document. This HTML file can be configured (appearance only) by the user. There are rough hints as to how to do this.

About the application and distribution

This application consists of two separate user interfaces, one to allow the user to index a collection of documents and the other to search that index and find the documents in it. The general usage is to create an index and then use the index as a basis for searching.

The program is started via a DOS batch file (which can be started by double clicking on the file name in the explore (file explorer). There are two batch files, index.bat and search.bat, with, not surprisingly, index.bat starts the indexer interface and search the search interface. For the potential Linux or other Unix user, I have included index.sh and search.sh as batch files for such systems. Do not edit these files with Windows editors! In a Windows environment, each line ends with CR and LF (carriage return, 0D, and line feed, 0A), whereas in a Unix environment, only LF is used.

I have added a possibility to start the program with a default index directory (that is the directory in which the index is to be written and the directory in which the search machine finds its index.)

The program is stored in the directory, C:\searchTest, for example. The directory contains one directory "data" and data contains several files. Some images, tkl.png and lucene.gif used for the entry screen, screen shots for this document, searchresults.css (more later about this), the two configuration files (indexer.defaults and search.defaults) and diverse documents, this included. The searchTest directory contains the following

index.bat	This program starts the application to do indexing in a Windows environment. It need no parameters and can be started with the windows file explorer with a double click of the mouse.
index.sh	This program starts the application to do indexing in a Linux (Unix) environment. It needs no parameters and can be started with the file explorer with a double click of the mouse. The user must set the permissions to rwxr.xr.x (as an example). It can also be set up as a launcher application.
search.bat	This batch program starts the search application. It also has no parameters and as above can be started with a double mouse click.
search.sh	This batch program starts the search application in a Linux (or Unix) environment. It can be started from the file explorer or it can be set up as a program launcher. Be sure to set the permissions (suggest at least rwxr.xr.x).

indexUpdate.bat	A batch file to start the update user interface. As with the other batch files this can be started from the windows explorer.
indexUpdate.sh	The UNIX (Linux) version to start the index update user interface.
search.defaults (in directory data)	This is a Java properties file. See the file for an explanation. At the moment it is set for Windows and the only item realize now is the call for the Internet explorer which is used to view the log, the documentation and the results. This option is for users in other environments (Linux, Mac, etc) who wish this functionality.
index.defaults (in directory data)	This is as the search.defaults in that it needs to communicate to the system how it can find firefox or iexplore.exe (for viewing the documentation) and notepad or gedit for viewing the log. But it has more information. Namely a list of abbreviations for the file suffix and the corresponding class to read them. See the section
indexandsearch.jar	This is the jar file (jar means Java Archive) which contains the indexing program and the search program, as well as all of the necessary support classes. (A class is in effect a sort of complex variable with associated programs to manipulate the associated data.)
jtidy.jar	This contains the Java classes for the JTidy application. This application transforms an HTML file (or data stream) into syntactically correct XML. The problem is that in HTML it is not essential for a tag to have a closing tag and in XML every tag must have a closing tag. e.g., (Break) is OK in HTML, but in an XML file this would be in error, it must be either, or </br>. This is necessary to read the HTML files we have. (See http://tidy.sourceforge.net/)
log4j-1.2.9.jar	This is necessary for PDFBox. This is a so called logger, it servers to organize messaging of events and errors. We do not use it directly, it is simply necessary for PDFBox. See http://logging.apache.org/ .
lucene-1.4.3.jar	The Lucene application is the basis for our index and search programs. It is also an Apache application. See
PDFBox-0.7.1.jar	PDFBox is a package to render PDF files (in effect to extract a text version from a PDF file). See http://www.pdfbox.org/. PDF Box is also a SourceForgeNet project.
tm-extractors-0.4.jar	Text mining extractors is a collection used to render Microsoft Word documents into text. See http://www.textmining.org . I chose this over POI, simply because this works well and POI doesn't seem to be so developed. Be ware that there are some classes org.apache.poi.utils which are not in concordance with PowerPointExtractor. So be sure to have poi-3.0... before text mining jar.
xercesImpl.jar	Xerces is also an Apache project for XML parsing. To parse in this sense means to read with respect to the language (XML). That is to not only read the contents, but put read with respect to the XML tags. See http://xml.apache.org/xerces2-j/.
xml-apis.jar	This jar file belongs to the Xerces project. See above.
xmlParserAPIs.jar	The XML parser API collection is also part of the Xerces project. I need it for the class XML Serializer. I need this to write the results of the search to an HTML file (which is also XML compliant, a so called XHTML).
poi-scratch-pad-3.0-alpha1-20050704.jar	Poor Obfuscation Implementation (referring to Microsoft propitiatory data formats) is the name of the Apache project. We use this to read Power Point presentations . http://jakarta.apache.org/poi/ . We use a subproject titled HSLF, I do not yet know what this stands for, but HSSF (for working with EXCEL) stands for "Horrible Spread Sheet Format". See Poi Terms .
poi-3.0-alpha1-20050704.jar	This is the main jar for the POI project. This supports the PowerPointExtractor.

As one can see this application is based on the work of others, all in the open source community. I would like to explicitly acknowledge this here. The source code of this application is available in the distribution.

Indexing

"Eine führerlose Menge ist zu nichts nütze" Machiavelli, "Discorsi",43

New with version 1. The following file types can now can be indexed.

blk	I have labeled this blank, which includes files with no suffix , for example "license" or files with names such as "read.me".
csv	This is the standard EXCEL format called comma separated variable, which in spite of the name, I had until recently only seen semi-colons used. In a German environment this is clearly necessary. However, in the samples I have from our outside sources indeed us a comma. I use a comma in my realization. Should a European file (semi-colon) be the separator, the consequence would be that lots of "numbers" would be indexed. e.g., 1;0 or 123;78;0 etc.
doc	WinWord document format. I use the tool form textmining.org for this.
html	For HTML (Hyper Text Markup Language) files. I use JTidy to put this into XML and then use DOM (w3.org) model to read the various nodes.
pdf	Adobe PDF (Portable Document Format). I use the PDFBox tool kit to extract the text from these.
ppt	Microsoft Power Point document format. I use the POI PowerPointExtractor to the text. This is not a really released version, but according to my tests it functions. See POI Terms)
rtf	This is the Rich Text Format, a Microsoft format which has been published. I use the Java standard RTFToolKit (javax.swing.text.rtf.)
txt	Text files are of course the easiest to handle.
xls	This is the Excel format (see POI Terms). Only the text is extracted and indexed from such files.

The basic procedure is to get the contents of the file as a text string and feed the text string to the analyzer.

Technical issues on the realization of the file readers. It is possible in Java to bind a class (classes are sort of like programs) while the program is running. This means that for the indexer, we can decide which readers to use via the configuration file. At the time of this writing, I have written the following classes:

```
# The term filetype followed by the file suffix (if more than one, use comma(,) to s
# and after the semicolon followed by the class name and path.
filetype.csv:com.theloutons.search.specialreaders.CSVReader
filetype.doc:com.theloutons.search.specialreaders.DOCReader
filetype.xls:com.theloutons.search.specialreaders.XLSReader
filetype.html,htm:com.theloutons.search.specialreaders.HTMLReader
filetype.pdf:com.theloutons.search.specialreaders.PDFReader
filetype.ppt:com.theloutons.search.specialreaders.PPTReader
filetype.rtf:com.theloutons.search.specialreaders.RTFReader
filetype.xml,xsl:com.theloutons.search.specialreaders.XMLReader
filetype.txt:com.theloutons.search.specialreaders.TXTReader
filetype.blk,me,:com.theloutons.search.specialreaders.TXTReader
```

This is in the so called "properties format" that is a key word followed by : (or !) as a separator and then the name of the class to be loaded. I have structured the key word as filetype to tell the program that information of this type is here. Then the suffix (or suffixes) followed by then the class name after the separator. There is a button to start the configuration editor associated with this system. This data is necessary only for the indexer. There are two configuration files, one `indexer.defaults` and `search.defaults`.

Thus it is possible to choose which types of files one wishes to index as well as the reader. The reader simply extracts the textual information from the file and returns a byte stream (a string, or a series of characters) to the indexer. For any file type the user wishes not to use, put a hash (#) mark before it, as in the first two lines.

Details about the indexing process. At the moment the indexing procedure is limited to filtering out common English language stop words and setting all letters to small case. It would be possible to extend this to be able to choose different analyzing methods.

The decisions to be made on the part of the user are a) where to store the index and b) what to index.

The index must be stored in a completely different place from the collection of documents to be indexed. Also the program is at the moment conceived that it will index only one directory. The suggested procedure then would be to chose the target, the directory in which to write the index and then to chose the source, the directory containing the files to be indexed. The tell the system to begin -- push the button "index now".

A word of *CAUTION* once you start to create the index, any contents found in the directory chosen will be erased. A log book is written to the directory containing the index (log.txt). This contains a list of files which for some reason or another could not be indexed, as well as the number of files indexed and start and end time. Time is in "Java time", that is number of milliseconds since January 1, 1970, 00:00:00.000 GMT. I do not bother with the date. Further, a sort of copy of the indexer.defaults file so that one can later see what sort of files were indexed.

Note

A view of the indexing user interface

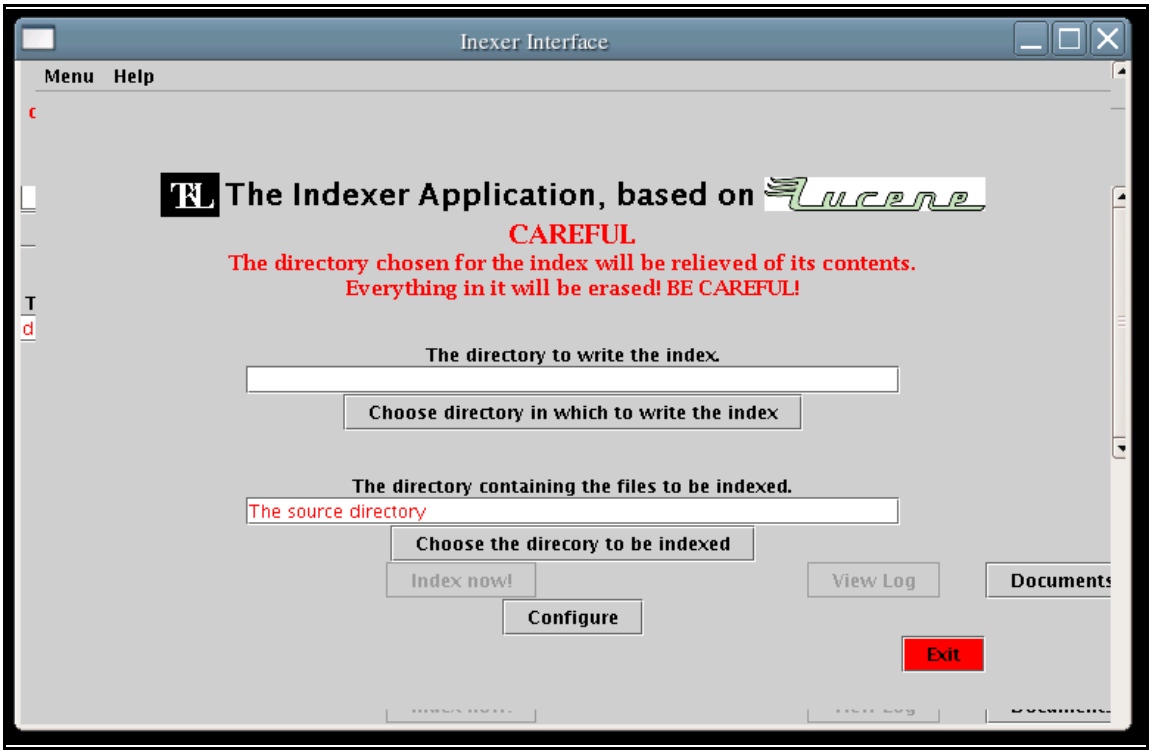

Figure 1. Indexer Application Screenshot

Known Indexing Problems

There is the potential for problems occurring and this section is intended to support the user who may encounter a serious problem. That is problem which stops the application and prevents indexing. The following is a list of the known problems and suggestions on how to handle them.

An "out of memory" exception is an exception I cannot catch and handle. The program stops. I have observed this with PDF and PPT files only.

blnk (miscellaneous collection of file types, but viewed as text files)	I have not encountered any problems here. Suffixes me (read.me), blank. More can be added if needed.
CSV (Comma Separated Variable)	This is the Excel export format. I have not had any problems as such here. One point though, I filter out numbers etc. That is only cells with "non" number contents are indexed.

DOC (WinWord File)	I have not had any problem here.
HTML (Hyper Text Markup Language)	The user will see an occasional log (in the DOS window or Terminal) entry mentioning XML problems. This means that JTidy (this is a program which takes HTML files and adds corrections to make them XHTML. I need this to use the Document Object Model to extract the text. This is not important for the user.
PDF (Adobe Portable Document Format)	If an "Out of Memory" error occurs, then find the file and remove it. I do not know why this occurs nor do I know how to prevent this. I have noticed 6 such files from Decision Resources which caused this. Otherwise I have not had any problems.
PPT (Microsoft Power Point)	I had one occurrence of an "out of memory" exception. This was a file with no text whatsoever, it had only pictures. I added text at one place and that removed the problem, even when I later erased the text, there was still no problem.
TXT (Text files)	Text files have up to now caused no problems.
XLS (Excel)	I extract only the text fields from the Excel file. No problems have occurred here.
XML	If the XML file is not a valid XML file(e.g., a tag not completed) an exception occurs and the error is written to the log and the parsing stops.

Updating an Index

> God, don't let me die, I have got so much to do yet! (Attributed to The Kingfish, Huey Long, 1935, as his last words.)

There is a separate screen to update an index. To use this it is for practical purposes the same as creating an index. One must select the directory which contains the index. Should this directory not contain an index, a message with a note to correct this is given. The directory to containing the documents to be indexed (updated) must not be identical with the original document directory. This means that one can index one directory and at a later point in time add a second one to the original.

How does this work? The program first checks to make sure the index directory really contains an index. Then it recursively (as the indexer) parses the directory containing the documents to be indexed. If a document is not in the index it is added then and there. If it be present in the index the dates are compared and if the file be newer, the old one is removed and the newer version is added.

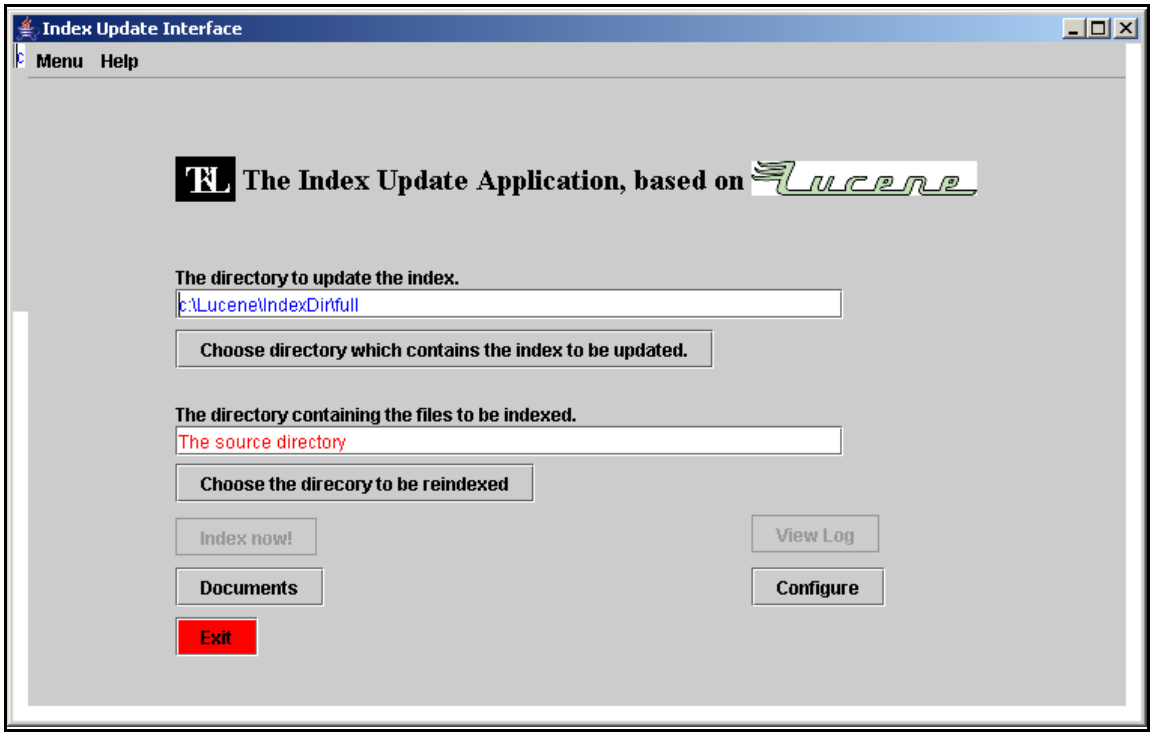

Figure 2. Index Update Screenshot

Searching

"Suchet, so werdet ihr finden" Lukas, 11.9

There is one type of search realized: a parsed search. Parsed in this case means that the string which is read in is analyzed. This is a very flexible search engine. That is a search can be started with several words using AND OR etc. (See below for complete instructions).

It would be possible to add different search functionality in the future.

Quick introduction to Lucene Parsed Query Search Syntax. The symbol back slash "\" is called in this context the escape symbol, and it is used to "escape characters". If you wish to search for any of the following symbols in your search string, they must be escaped:

```
\ + - !( ) : ^ ] { } ~ * ?
```

Thus if one wishes to search for ? then \? must be entered. The boolean (yes/no) terms AND OR can be used (*Must be written in caps.)*. Placing NOT in front of a term negates it. NOT term would look for those documents without term in them. NOT may not be the first item in an expression.

Warning

The boolean terms MUST be written in capital letters!

Boolean examples

a AND b	Can also be written +a +b. Searches for all documents containing a and b.
a OR b	Can also be written a b -- a and b separated with blank. Find all documents with a or b present.
a AND NOT b	Can also be written as +a -b. All documents with a and not b.

The use of parentheses is allowed to group boolean terms.

Two words such as bbbb cccc would be taken to mean bbbb OR cccc. If one writes (bbbb and cccc) the system looks for "bbbb" or "and" or "cccc". The word and is a stop word (for us) and therefor never present. Thanks to Peter for noting this!

Exact searches

"term1 term2"	will find those documents which contain term2 exactly proceeded by term1. See proximity searches below. Note that if a document contains Schering-Plough, Schering_Plough or Schering Plough, the term "schering plough" would find all three variants because -,_. and blank are token separators.

Grouping examples

(a AND b) OR (c OR NOT d)	Either the document contains a and b or it contains c but not d.

Wild cards can also be used. By a wild card we mean the use of the symbol * (zero or more characters) ? zero or one character. The wild card character may not begin the search.

Wild card examples

wood*	would deliver all documents which contain a text string beginning with to and with 0 or n characters following, so here we might find wood, woodruff, woodpecker, woodnote, woodman, woodenman etc.
wood?	would only deliver wood or woody. ? is often used to find things abc?def, words starting with abc and ending with def either 6 or 7 characters long.

Proximity searches

"term1 term2"~n	This will find those documents which contain term1 and term2 where term2 is proceeded by term1 with a gap of max n words. Recall here too that aside from blank, "-" and "_" function to separate tokens.

Parsed Query Fuzzy Search

wuzza~	fuzzy for example would be found. This is based on a complex algorithm called the Levenshtein distance [http://www.merriampark.com/ld.htm]. I have not studied this in detail and for the moment it seems to me rather unpredictable. I have looked at the web site however, and the explanation is clear and I think understandable, but still sufficiently complex that I could not quickly guess the results of a fuzzy search.

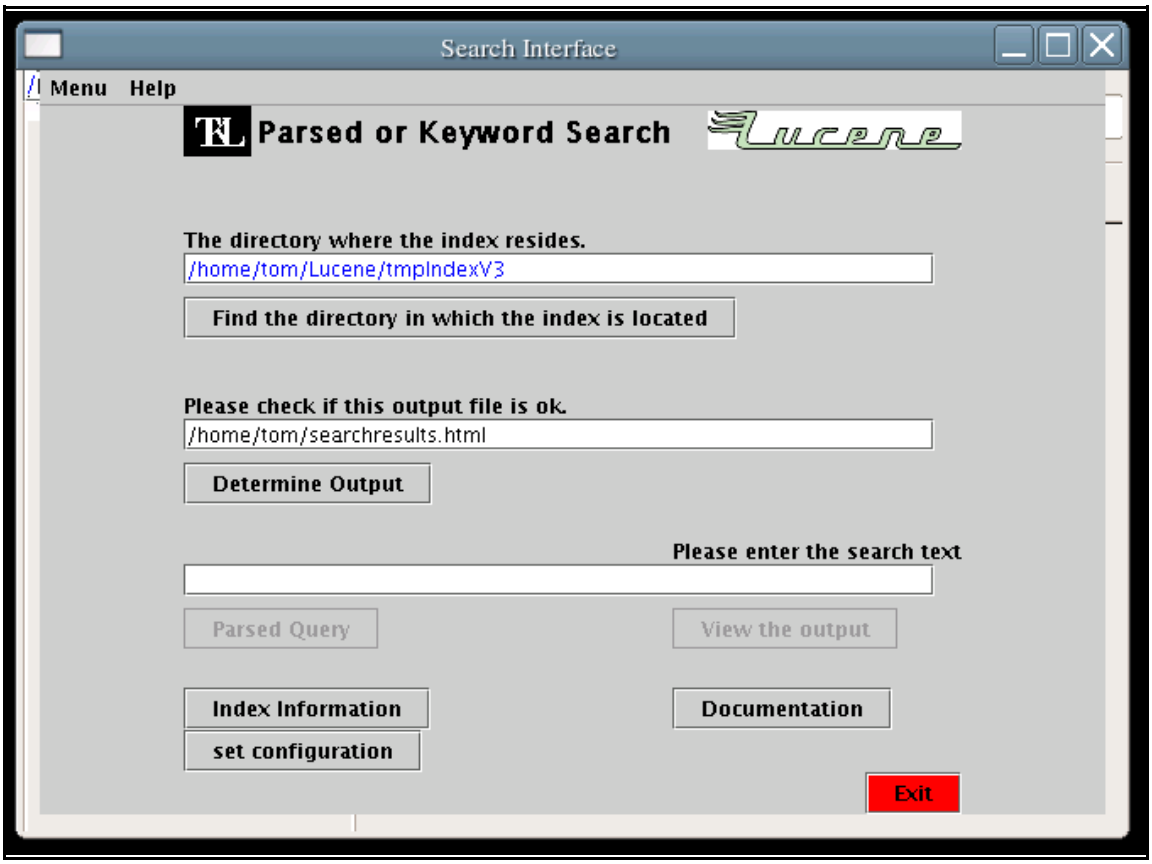

Figure 3. Search Interrface Screenshot

Presentation

The results are presented as HTML files (actually XHTML) and the styles are done with CSS (Cascading Style Sheets). I have a default programmed and there is a possibility to read in a CSS file (searchresults.css) in the directory "data" just in the directory from which the search application starts. Here are a few hints as to how to change the CSS, and a reference to a document called "SelfHTML" [http://de.selfhtml.org/] by Stefan Münz. If you like this can be downloaded and set up on you PC. I also have it on my server (bes601). This is the best document I have seen on HTML.

In the case of our presentation the CSS is copied to the directory in which the search results are written and is imported into the HTML document. As mentioned above you may edit the file searchresults.css in the data directory to suit your needs or wishes. Here is a sample, an excerpt from a CSS file with each layer colored differently and with many options as examples.

```
/*The style for the search results*/
body {background-color:#7F7F7F; color:#000000;font-family:'Century Schoolbook',seri
h1 {background-color:#FFFF00; color:#800000;font-family:'Century Schoolbook',serif;
h2 {background-color:#FFBF00; color:#800000;font-family:'Century Schoolbook',serif;
h3 {background-color:#00FF3F; color:#800000;font-family:'Century Schoolbook',serif;
h4{background-color:#FF7F00; color:#800000;font-family:'Century Schoolbook',serif;
h5 {background-color:#00FF3F; color:#800000;font-family:'Century Schoolbook',serif;
a {background-color:#FF9933; color:#5940BF;font-family:'Century Schoolbook',serif;
p {background-color:#FF9F00; color:#2F2F2F;font-family:'Century Schoolbook',serif;
ul {background-color:#FF7F7F; color:#800000;font-family:'Century Schoolbook',serif;
ul li {background-color:#FFDF00; color:#800000;font-family:'Century Schoolbook',ser
      border-color:#000000; border-style:solid; border-width:thick; margin-bottom:3
ul li p {background-color:#C0C0C0; color:#800000;font-family:'Century Schoolbook',s
ul li ul {background-color:#FF7F00; color:#800000;font-family:'Century Schoolbook',
ul li ul li {background-color:#C0C000; color:#800000;font-family:'Century Schoolboo
      border-color:#C000C0; margin-bottom:0em}
```

```
ul li ul li p {background-color:#F0F0F0; color:#800F00;font-family:'Century Schoolb
```

With the style sheet one can control the format and layout of the report. I have added a variety of colors to allow you the user to see the effect of each of the colors on the layout. Also it is possible to put "boxes" around items and padding between the text and the "box" (border) and the space between the box and the next item, the margin, can also be formatted.

Recall the basic structure of the output HTML file is the following

```
<!doctype html public "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<meta content="Text" name="D.C. Type"/>
<style type="text/css">
@import url(searchresults.css);
</style>
</head>
<body text="#000000" bgcolor="#FFFFFF" link="#FF0000" alink="#FF0000" vlink="#FF000
<!-- repeated for each document the list -->
<ul> <!= the top level list, just once in document -->
     <li> <!-- the list item for each document one li for each document-->
        <ul> <!-- list of information for the document -->
            <li> <!-- list item for each type of information, title, synopisis, summ
            </li>
        </ul>
     </li>
</ul>
</body>
</html>
```

The style tag is part of the head (header) tag along with various meta data one may chose to include in the file. The meta data are important for allowing web crawlers and such to classify the document. My experience is that these are often not present or very sloppily filled out. I have included a few, but this is hardly necessary for such a document. In any event the CSS style sheet is either a in a comment section of the style tag, or an external file, e.g., searchresults.css, which is referenced via the import directive (@import url(file name);. In HTML a comment starts with <!-- and ends with -->.

The current default is much simpler and plain. The above is just to support the user in modifying the CSS file to create a "personal" touch to the presentation.

Just a few hints. The tags used to present the data are ul (unordered list) and li (list item). I use them in a nested fashion. That is as below.

```
<ul>The documents
  <li>Document 1
    <ul>
     <li>File ref for doc 1<a href=.... >file ref </a></li>
     <li>Summary for document 1<p>.....</p></li>
    </ul>
    <ul> .... </ul>
.
.
.
</ul>
```

Thus to change the appearance of the results document, changes to the styles for ul and li will cover most of the document. The style items such as padding, spacing etc are clear or can be made clear with experiment. To change the colors it is more complex. Here is color reference to Netscape 256 colors palate [http://de.selfhtml.org/diverses/anzeige/farbpalette_216.htm] . This is in the so called RGB model. The letters and numbers are in HEX format (hexadecimal, 0-9,A,B,C,D,E,F, that is from 0 to 15. There are much more detailed charts, which when you click on them, the code is put in the memory and can be pasted into the field.

Organizing the application, batch files

The applications are started by a DOS batch files, called index.bat and search.bat. Both of these are set so that the user can set an initial directory for the index. So for example from a DOS box (enter "start" "run"

and cmd in the box in the run window.) one could enter the following command from the directory where the application is installed:

```
index C:\Lucene\IndexDir\full
```

This would start the indexing program with the index directory set to C:\Lucene\IndexDir\full. However this is too much if one wants to start this over again, so one creates a batch file (in the directory where the application is installed or somewhere elsewhere {but then the path to the index.bat file must be complete} put the above text into a file whose suffix is "bat".

It is also possible to have a shortcut with the above text as a command.

The same holds for the search.bat file. Actually this is more useful, as one indexes once and searches often.

Bibliography

2005. Otis Gospodnetic. Erick Hatcher. *Lucene in Action*. Manning. 209 Bruce Park Ave, Greenwich, CT 06830 (www.manning.com).

2004. Manfred Hardt. Fabian Thies. *Suchmaschinen entwickeln mit Apache Lucene*. Software & Support Verlag. www.software-support-verlag.de.

Glossary

Selected Terminology related to text processing

Boolean Search	Choosing a document based on the presence or absence of items.
Chunking	The process of recognizing higher level structures in a sentence, chunks. e.g., (The post office) (will) (hold out) (discounts).
Corpus	A collection of documents used for learning (machine learning), as a source for a treebank.
Data Mining	The exploration of data (usually well defined) sets to find quantitative associations between variables.
Document Classification	This activity involves parsing a document (with a program) and placing it in a group, that is classifying it. The standard method involves using a set of pre-classificied documents to develop a rule for classification.
Document Clustering	This is the process of examining a collection of documents and finding associations be tween them and breaking the group into subgroups (clusters)
Index	In terms of search engines, an index is a list of tokens (words in effect) found in some document. Associated with each token there is a list of documents in which the token was found. To then each document the position of the token in the document is also stored.
Information Retrieval (IR) or Information Extraction (IE)	Searching or gleaning information from defined sources. There are subtitle differences between these terms. For example, when we use a tool such as this to find a given document in a large collection of documents, this is information retrieval.
Named Entity Detection	Named Entities are in effect a classification of words. This is used especially in areas such as PoS tagging (see below) and Natural Language Processing
Natural Language Processing (NLP)	The process of analyzing documents (sentences) in terms of language structure. Do not confuse with Neural Linguistic Programming (related to Hypnosis Link Partners (really)

Part of Speech Tagging (PoS Tagging)	For each token detected, to affix the part of speech it had in the context in which it was found.
Rating	This is the procedure of affixing a value or number to a document to allow the collection to be sorted to put the "most interesting" at the beginning of the list. The exact procedures are usually not open for public perusal. Lucene rates via the following: • frequency of term in the document • inverse document frequency of the term (how many documents in index contain the term) • booster factor -- default 1 -- can be set by user (programmer) as a function of the term and the document. • Normalization factor of a index, given the number of terms within the index (thus not dependent on the term) • coordination factor based on the number of terms the document contains
Sentence Detection	Filtering a document to find or break down into sentences. The is more difficult than meets the eye.
Stemming	The process of reducing a word to its stem (linguistically). For example, gone, going, went, go would all be reduced to go. Big, bigger, biggest would all map to big. This can be done with the use of a dictionary or even via an algorithm. For the latter see: M.F. Porter, 1980. An algorithm for suffix stripping, [Program, 14](3) 130-137. This is the most Porter Stemming Algorithm.
Stop Words	Words which are to excluded from the potential search (during indexing). In English for example, "he", "she", "the", and "and".
Text Mining	The process of finding associations between words (not known in advance)
Tree Bank	A collection of correctly parsed sentences. The best known is the PennTreebank.
Token	A word or a unit in a text string. Usually separated by blanks from its neighbors, but "-" and "_" are also used.

POI Terms

HDF	HSSF Horrible Data Format: Java API to read Microsoft Excel
HSLF	Java API for PowerPoint presentation (documents)
HPSF	Horrible Property Set Format: Java API for reading property sets using (only) Java
HSSF	Horrible Spreadsheet Format: Java API to read Microsoft Excel
HWPF	Java API for reading and writing WinWord documents.
POIFS	Poor Obfuscation Implementation File System: Java APIs for reading and writing OLE (Object Linking and Embedding) 2 compound document formats

Without the search engine Lucene from Apache I could not have developed this tool.

This document is in XML using the DocBook and Norman Walshe's stylesheets.