

High Speed Searching with Lucene

A practical Introduction to the Programming of
Apache's Lucene Framework

Eric Schreiner
Managing Director
Contecon Software GmbH





Prerequisites

- Basic Understanding of Java would be helpful.



Agenda

- What is Lucene
- Understanding the indexing process
- Understanding searching
- Tools and Applications
 - Regain
 - Luke



Conventions used in this presentation

- *Blue italic* will be used for keywords and operators in the text (not in example code)
- All class names for Java classes used in my examples will start with a capital *J* to differ them from Groovy
- Additional example code is indicated in *green* at the bottom of the slides. *Grey* indicates that we will skip the sample during the presentation (time is limited)



Definitions

- **Apache**

Apache Lucene is a high-performance, full-featured text search engine library written entirely in Java. It is a technology suitable for nearly any application that requires full-text search, especially cross-platform.

- **Wikipedia**

Lucene is a free/open source information retrieval library, originally created in Java by Doug Cutting. It is supported by the Apache Software Foundation and is released under the Apache Software License. Lucene has been ported to programming languages including Delphi, Perl, C#, C++, Python, Ruby and PHP.

While suitable for any application which requires full text indexing and searching capability, Lucene has been widely recognized for its utility in the implementation of Internet search engines and local, single-site searching. Lucene itself is just an indexing and search library and does not contain crawling and HTML parsing functionality.



Platforms (Apache)

- **Apache Lucene** is a high-performance, full-featured text search engine library written entirely in Java. It is a technology suitable for nearly any application that requires full-text search, especially cross-platform.
- **Lucy** – a C port of the java Lucene search engine with pearl and Ruby bindings
- **Lucene.Net** is a source code, class-per-class, API-per-API and algorithmic port of the Java Lucene search engine to the C# and .NET platform utilizing Microsoft .NET Framework.



Powered by Lucene

- see <http://wiki.apache.org/lucene-java/PoweredByLucene>
- Software
 - more than 120 entries
 - Eclipse, JIRA, Apache Roller, *etc.*
- Websites
 - also more than 120 entries
 - Wikipedia, jGuru, *etc.*



What are we doing with Lucene?

- We provide software for managing monitoring events
- Our biggest installation:
 - >34million entries on Intel/PC platform
 - new entries permanently added (~50000 entry's a day)
 - new events are immediately visible for queries
- We use Regain as our intranet search engine
 - Documents: PDF, Open office, JPEG, Java, *etc.*
 - MediaWiki

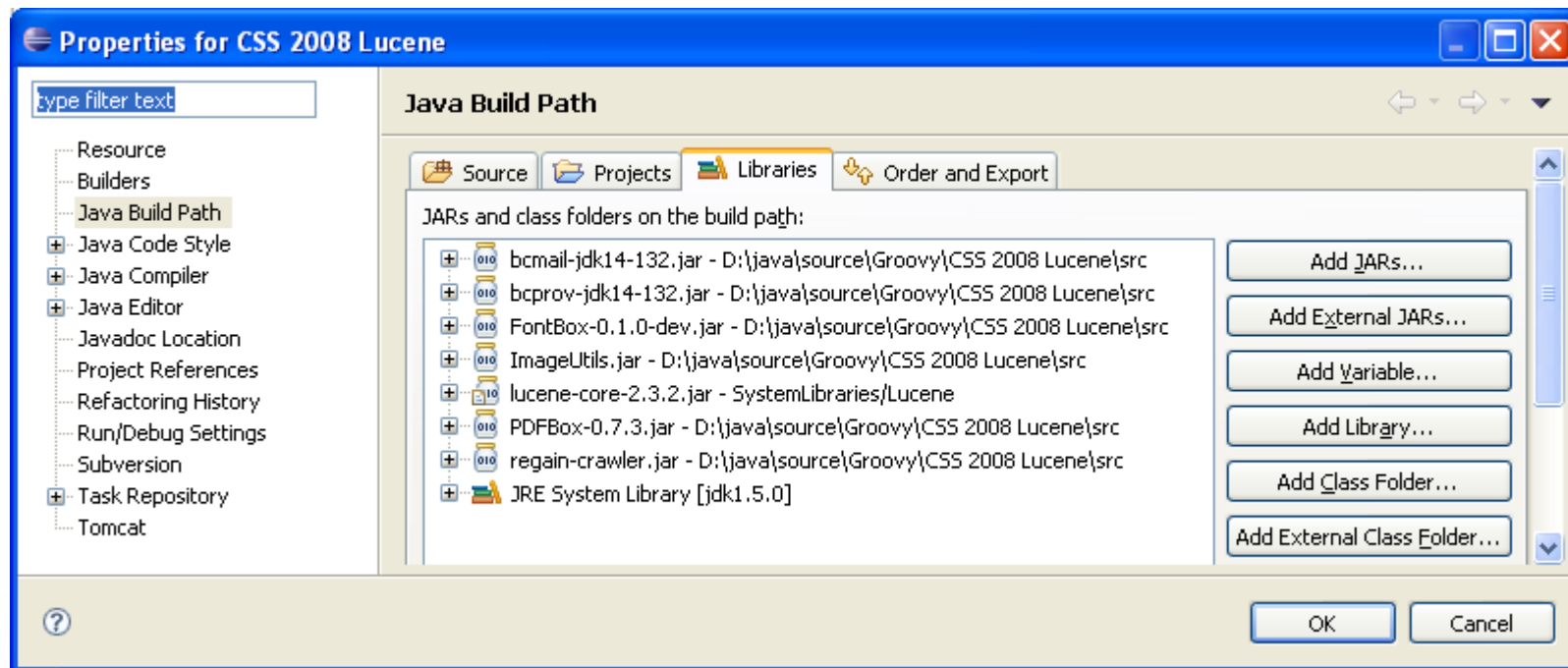


Benchmarks on our server

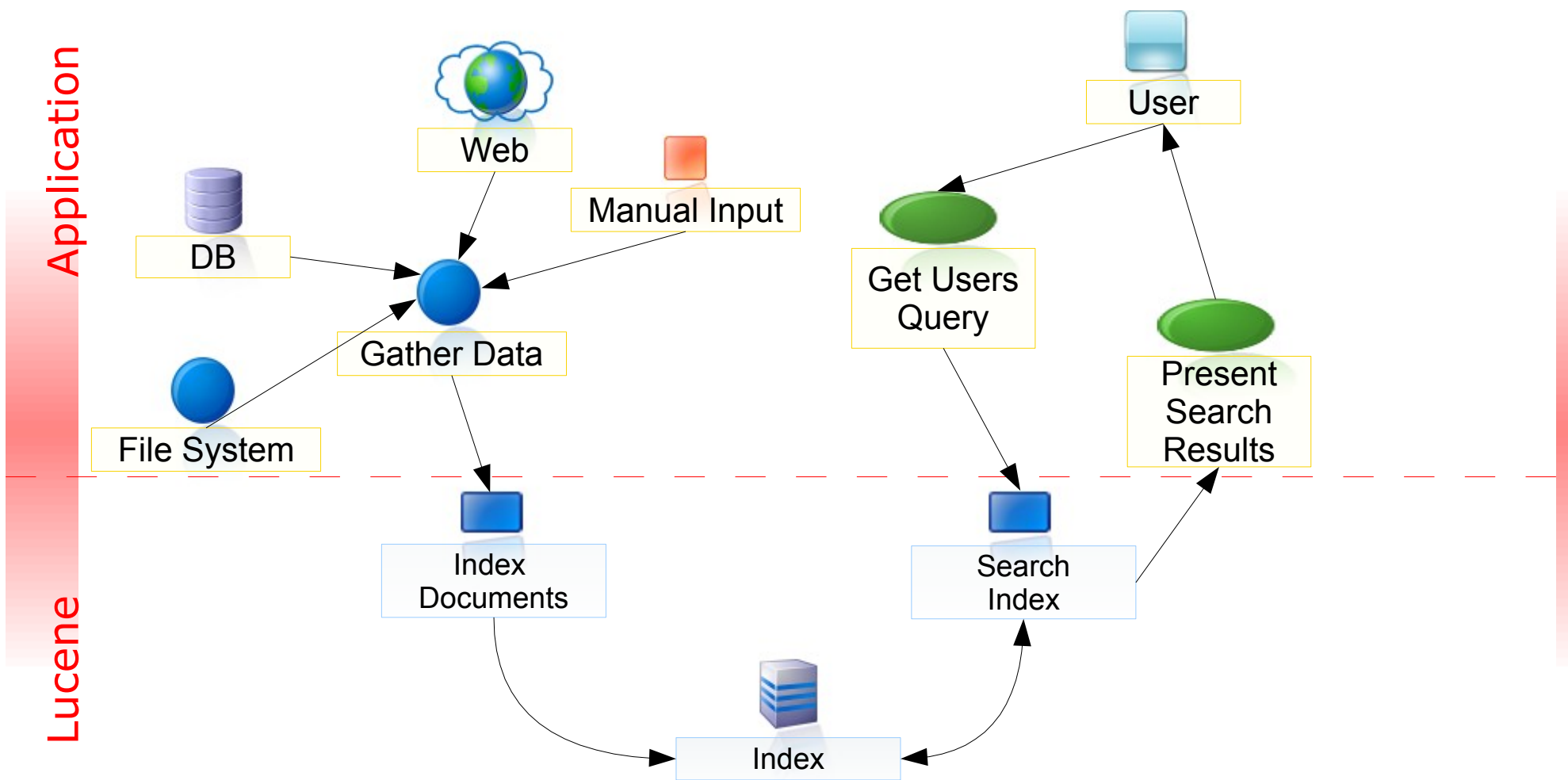
- Full text event search on all 34 million entries with 3 results: < 1 second
 - +1002004300
- Date range query
all events from a specific device sorted by date with 8000 results: 10 seconds
 - +date:[20080101 TO 10080201]
+devid:4711

Eclipse setup

- Just download lucene-core-2.3.2.jar
- ImageUtils.jar will be used for the Regain example
- Lucene \geq 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

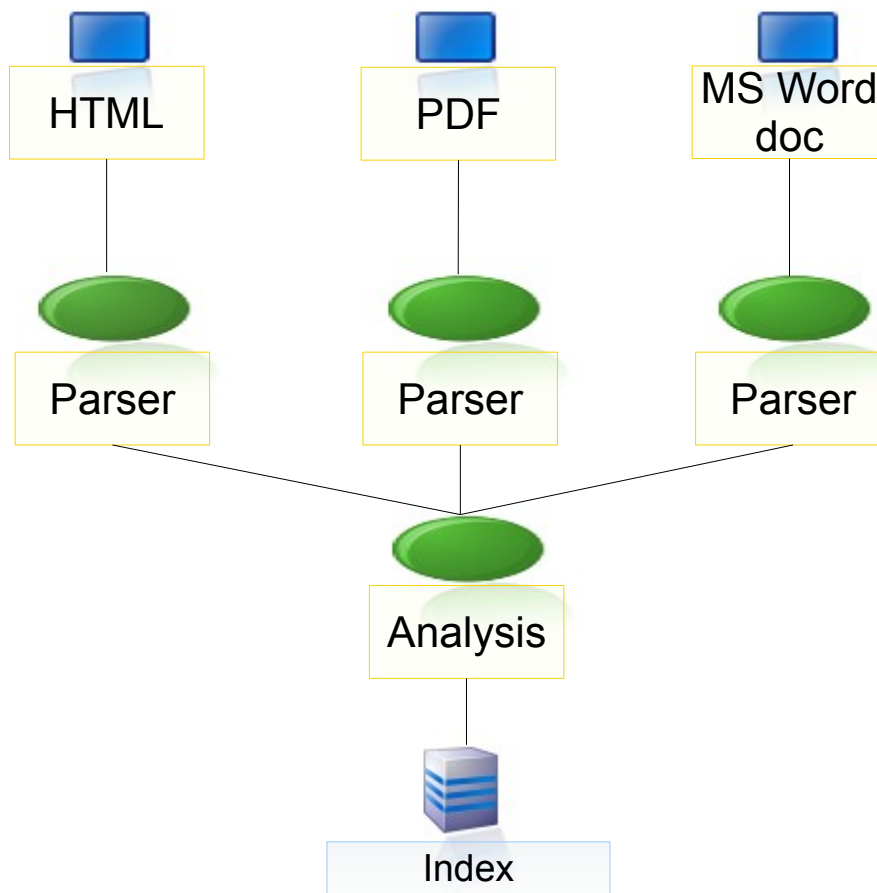


How it's integrated

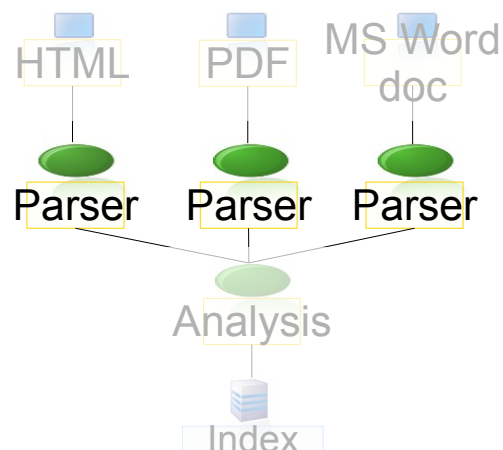




How the indexing process works



How the indexing process works



■ Parsing or text extraction

- Process of removing the meta data or formatting data from a dokument.

```

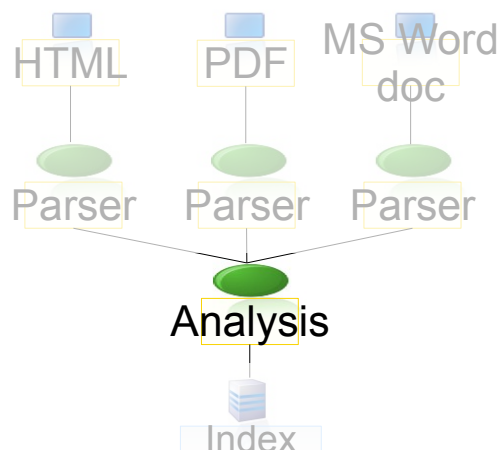
<html>
  <title>Eric's mini book</title>
  <body>
    <h1>1st chapter</h2>
    <p>
      This is the boring
      story
      about my
      childhood.
    </p>
  </body>
</html>
  
```

text extraction

```

</html>
  </title>Eric's mini book</title>
  </body>
    </h1>1st chapter</h2>
    <p>
      This is the boring
      story
      about my
      childhood.
    </p>
  </body>
</html>
  
```

How the indexing process works



- Analyzers (more detailed later)

- strip stop words like: a, an, the, is and, so, etc.
- strip punctuation characters
- strip whitespace characters

analyzing

result

Eric's mini book
1st chapter
This is the boring
story<CR>
about my childhood.

~~Eric's~~ mini book
~~1st~~ chapter
~~This is the boring~~
~~story<CR>~~
about my childhood.

Eric mini book
chapter
boring story
childhood



Libraries for text extraction

- PDF
 - <http://www.pdfbox.org/>
- HTML
 - <http://htmlparser.sourceforge.net/>
- XML
 - SAX
 - DOM
 - other



Analyzers provided by Lucene

- Analyzers inherit from the abstract class *org.apache.lucene.analysis.Analyzer*
- An Analyzer builds TokenStreams, which analyze text. It thus represents a policy for extracting index terms from text.
 - *org.apache.lucene.analysis.WhitespaceAnalyzer*
splits tokens at whitespace
 - *org.apache.lucene.analysis.SimpleAnalyzer*
divides text at non letters and lowercase
 - *org.apache.lucene.analysis.StopAnalyzer*
divides text at non letters and lowercase and remove stop words
 - *org.apache.lucene.analysis.KeywordAnalyzer*
used for ZIP-code id's, etc.
 - *org.apache.lucene.analysis.standard.StandardAnalyzer*
Most sophisticated general purpose analyzer



A simple pdf indexer

- all PDF files are from http://www.planetpdf.com/free_pdf_ebooks.asp

Create an *IndexWriter*

for each Document

Create a *Document*

add *Fields* to *Document*

add *Document* to *IndexWriter*

Close *IndexWriter*



Let's use Luke to examine the results

- Java 1.5 or higher required
- we use
lukeall-0.8.1.jar
- Start with
java -jar lukeall-0.8.1.jar



core indexing classes

- *org.apache.lucene.index.IndexWriter*
- *org.apache.lucene.store.Directory* (abstract)
 - *FSDirectory*
 - *RAMDirectory*
- *org.apache.lucene.analysis.Analyzer* (abstract)
 - see slide (*Analyzers provided by Lucene*)
- *org.apache.lucene.document.Document*
- *org.apache.lucene.document.Field*



Field.Store

- ***Field.Store.YES***

Store the original field value in the index. This is useful for short texts like a document's title which should be displayed with the results. The value is stored in its original form, *i.e.* no analyzer is used before it is stored.

- ***Field.Store.NO***

Do not store the field value in the index.

- ***Field.Store.COMPRESS***

Store the original field value in the index in a compressed form. This is useful for long documents and for binary valued fields.



Field.Index

- *Field.Index.NO*

Do not index the field value. This field can thus not be searched, but one can still access its contents.

- *Field.Index.TOKENIZED*

Index the field's value so it can be searched. An Analyzer will be used to tokenize and possibly further normalize the text before its terms will be stored in the index. This is useful for common text.

- *Field.Index.UN_TOKENIZED*

Index the field's value without using an Analyzer, so it can be searched. As no analyzer is used the value will be stored as a single term. This is useful for unique IDs like product numbers.

- *Field.Index.NO_NORMS*

Index the field's value without an Analyzer, and disable the storing of norms. No norms means that index-time boosting. Once a given field has been used with norms enabled, disabling norms will have no effect.



core searching classes

- *org.apache.lucene.search.IndexSearcher*
- *org.apache.lucene.index.Term*
Basic unit for searching. Similar to the *Field* class it consists of a pair of String elements. The name of the field and the value for that field.
- *org.apache.lucene.search.Query* (abstract)
 - see next slide for implementations
- *org.apache.lucene.search.Hits*
A ranked list of documents, used to hold search results.
- *org.apache.lucene.document.Document*
can be obtained with the *Hits.doc(n)* Method.



A simple pdf searcher

Create an *IndexSearcher*
Create a *Query*

Hits = searcher.search(query)
for each hit in Hits
do something with the results

Close *IndexSearcher*



Querys

- *TermQuery*
Search for a specific Term
- *RangeQuery*
Search within a Range
- *PrefixQuery*
Search terms beginning with a specified String
- *BooleanQuery*
Combine querys
- *PhraseQuery*
Find terms within a certain distance, *e.g.* Monster near Frankenstein
- *QueryParser*
Processes human entered query's

QueryParser – Cheat sheet

- <http://lucene.apache.org/java/docs/queryparsersyntax.html>

Terms	Search for a single word like <i>java</i> or " <i>java language</i> ". A term can be combined with boolean operators.
Fields	<i>title:"The Right Way" AND text:go</i> (Fieldname and term is separated by a colon)
Wildcards	To perform a single character wildcard search use the "?" symbol. To perform a multiple character wildcard search use the "*" symbol. ? and * are not allowed as the first character of a search
Fuzzy searches	Lucene supports fuzzy searches based on the Levenshtein Distance, or Edit Distance algorithm. Just use ~ at the end of a single word term. <i>roam~</i> will find terms like <i>foam</i> and <i>roams</i> .
Proximity Searches	Lucene supports finding words are a within a specific distance (words) away <i>"jakarta apache"~10</i>
Range Searches	Inclusive range searches – square brackets: <i>mod_date:[20020101 TO 20030101]</i> Exclucive range searches - curly brackets: <i>mod_date:{20020101 TO 20030101}</i>
Term Boosting	To boost a term use the caret, "^", symbol with a boost factor. By default boost factor is one. <i>"jakarta apache"^4 "Apache Lucene"</i>
Boolean Operators	AND, "+", OR, NOT and "-" (capital letters MUST be used) (OR can be substituted with)
Grouping	Use parentheses to group: <i>(jakarta OR apache) AND website</i>



A word about out scoring

- Very sophisticated
- Very complex
- The boost factors are used in the calculation of the scoring
- see
<http://hudson.zones.apache.org/hudson/job/Lucene-trunk/javadoc/org/apache/lucene/search/Similarity.html>
http://en.wikipedia.org/wiki/Vector_Space_Model



What is regain?

- **Definition**

regain is a search engine similar to web search engines like Google, with the difference that you don't search the web, but your own files and documents. Using regain you can search through large portions of data (several gigabytes!) in split seconds!

(from: <http://regain.sourceforge.net/?lang=en>)

- **100% Java**

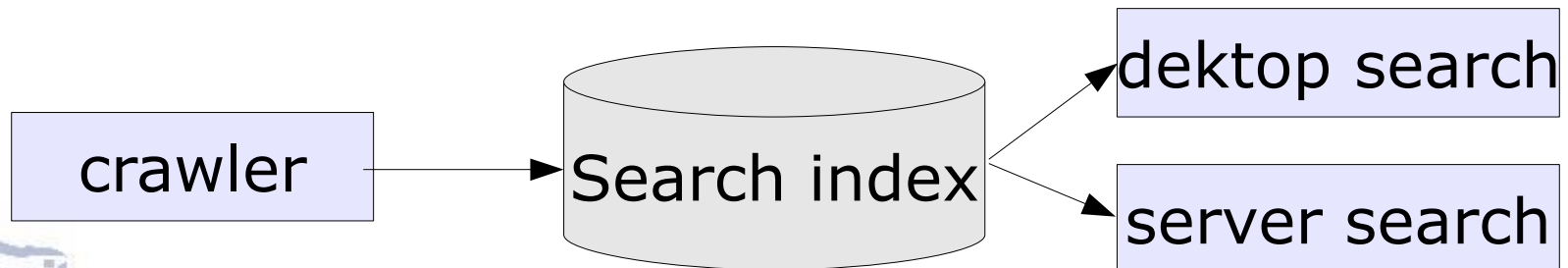
- **Released under the LGPL**

- **Uses Lucene as index database**

- **Can be extended easily by providing your own Preparator (We'll see an example later)**

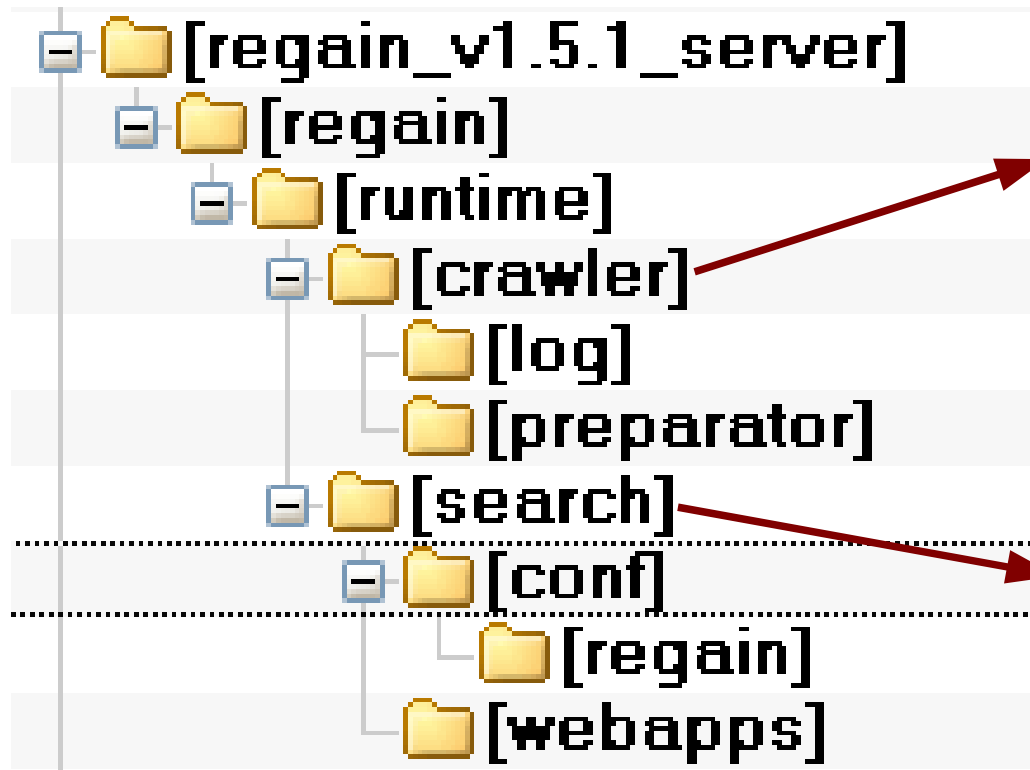
How regain works

- The work of regain is split in two parts:
 - The creation of an Index by using a crawler
 - ✓ The desktop search (Java app)
 - ✓ The server search (war – file)
 - The search on the created index
 - ✓ The desktop search (Java app)
 - ✓ The server search (war – file)
- There is a good documentation based on MediaWiki available under http://regain.murfman.de/wiki/en/index.php/Main_Page



The regain distribution

- The structure of regain_v1.5.1_server.zip



just unpack it to a directory

e.g. *d:/Css2008/crawler*

just unpack it to your tomcat directory

D:\Programme\Apache Software Foundation\Tomcat 5.5



Let's run the crawler

- configuration is done by editing *CrawlerConfiguration.xml*
- *java -jar regain-crawler.jar*
- Examine the result in the *RegainDB* directory with luke



Lets run the server search

- configuration is done by editing *Tomcat\conf\regain\SearchConfiguration.xml*
- start tomcat
- open the browser and run some queries

Custom preparators

- Extend *net.sf.regain.crawler.document.AbstractPreparator*
- Put all required classes in one jar-file
- Put the jar file in the crawler's *preparator* directory

Important:

the jar file must have a Manifest Entry with a list of the preparator classes:

....

Preparator-Classes: de.contecon.imageutils.JPEGRegainPreparator

....

References

■ Lucene

- Apache
<http://lucene.apache.org/>
- Luke – Lucene Index Toolbox
<http://www.getopt.org/luke/>
- Regain
<http://regain.sourceforge.net/index.php?lang=en>

■ Books

- Lucene in Action
- Building Search Applications: Lucene, Lingpipe, and Gate





Contact me



Eric Schreiner

Eric.Schreiner@contecon.de

<http://www.contecon.de>

Please fill out the evaluations



Thank you

....if you have questions

ask now

