BExIS 2.5.0

Lucene-Based Search Provider Administration Manual

Content

[1. Introduction 3](#_Toc350268642)

[2. Important Files/Folders 3](#_Toc350268643)

[3. How-To Modify Configuration File 4](#_Toc350268644)

# Introduction

Data search in BEXIS++ is carried out by data search module (henceforth referred to as data providers). Different implementation of the search provider module (using different algorithms and technologies) can be developed and configured to be used by the system. This document however describes the main concepts and configuration of a search provider developed based on Apache lucene search library.

Apache lucene is a high-performance, full-featured text search engine library using a powerful, accurate and efficient search algorithms. Some of its features include ranked searching, many powerful query types (phrase queries, wildcard queries, proximity queries, range queries), fielded searching (e.g. title, author, contents), flexible faceting, highlighting, joins and result grouping, fast, memory-efficient and typo-tolerant suggesters, etc. Our lucene-based search provider is based on lucene.net version 2.9 library

A search in Lucene is done against an index which has to be first created. In an overly simplified form, a lucene index can be viewed as one big table with several columns (referred to as fields), where each row represent a document, and each cell can contain 0:\* values and each value is made of 1:\* terms. For more information on lucene, visit

In order to create the index, therefore, elements of the metadata are mapped to the field/column in the lucene index. Based on the assumption that the metadata is stored in xml format in the database, we define the schema of the lucene index in a file luceneConfig.xml This file contains mappings of xml node values (specified using xpath expression) to some lucene field which we also defined in the file. One major challenge is how to flatten an hierarchical data structure such as xml or relational structure with several joins into one index table, and the system admin must take care of this.

The way lucene indexes a text is specified by the system administrator and this depends on factors such as which search feature or functionality is going to be used to search a field, and its primitive data type. The configuration file provides a means to specify how to index the fields.

In addition, because there is a strong dependency between search features and patterns provided and the index, the luceneConfig.xml file is also set to specify which is configured on a field and as well as some configuration about what is displayed in the search GUI.

Three main search patterns of search are supported by this provider

* Free text search (with auto-completion feature, and german language synomyn support) on some fields which we call “CATEGORY” in the search provider
* Faceted search which we call FACET in the search provider
* Filter based searches on some fields with very small number unique primary data, and/or data with numeric and date ranges. We refer to this as “PROPERTY” in the provider.

# Important Files/Folders

Lucene Configuration File: This file provides the mapping between metadata elements and the lucene index. It also configures the search pattern on each done against each lucene index field, in addition to specifying the each *how* each index field is indexed. This can be found in Application\Workspace\Modules\Search\Lucene\Config\LuceneConfig.xml. The system admin should take care that this is properly configured to conform to his metadata schema, and suit his search needs.

Search Index Folder: This is found at Application\Workspace\Modules\Search\Lucene\index. It contains the various indexes created for different search functionalities. The indexes created and their respective folders are given below

* Main search index: This is created in the folder BexisSearchIndex and it is the main index used in searching. The schema is based on the configuration file and it is created during the indexing process.
* Autocomplete Index: This is the index created for auto-completion of terms. This is created/populated during the indexing process. This is created in the AutoCompleteBexisDir folder
* Synonym indexes: These indexes are used to expand searches to include synonyms. Two synonyms indexes are provided by with the search provider. This include the English synonyms index based on wordnet and is created in the “WordnetIndex” folder. Similarly, we have German-to-English and English-to-German synonyms for some selected words (more than 5000), and this is created in the “EnglishGermanIndex” folder. At the moment, the list is fixed, but you can develop tools to manage this index if you so wish. The index consists of two fields namely, “word” and “syn” which respectively refers to the word and its corresponding synonyms.

# How-To Modify Configuration File

The configuration files consist of one element – the field element, and several attributes. The element represent each lucene field and its attribute are used to configure indexing, searching, and display.

We, therefore, go through each of the elements attributes of the configuration file.

* display\_name : This is the name which is displayed in the search UI for the field.
* lucene\_name : This is the name of the field in the lucene index.
* metadata\_name : Add one or more xpaths from the metadata elements to be mapped against the lucene field.
* type : This specifies the search pattern that will be should be used against this field (as discussed in the introduction). Therefore, the value of this attribute can be any of “category\_field” for creating a category-based search field, “facet\_field” for creating faceted search field, “property\_field” for creating a property search field, and “general\_field” which creates an indexed field which is not displayed in the UI, but however, searched.
* store : This specifies if the field value should be stored. If the field is not stored, you can only search against the terms in the field, however, you cannot retrieve the value. For minimal display of the search result, it is recommended that some fields be store. This value of the store attribute can be either yes or no
* primitive\_type : This specifies the primitive data type of the value to be indexed. E.g. string, integer, double, date
* analyzed : This specifies if the field should be analyzed or not. Only analyzed field can be searched.
* multivalued: This specifies if there are several values of a metadata element in a given field. E.g. if a dataset can have several owners in a metadata, then, the owners field in the index in this case, will be multi-valued
* norm : This can have a value of yes or no and it is used to specify if a norm should be created for the field. Norms can be used for similarities search between documents. They can also significantly increase index sizes. So, you must take care to of what field should contain norms
* boost: This specifies the importance or weight of a field relative to others in a search. E.g. you may want the terms in “title” of a document to carry more weight than the content of the “footnotes” while indexing and searching
* header\_item: This specifies if the field is to be used as an header item in the result table in the UI
* default\_visible\_item : Because not all columns of a result table are visible by default, this attributes specifies if this header item should be visible by default. The system admin should take care that the display is not clumsy.