LexEVS Code Decoupling

Document Information

Author: Craig Stancl, Scott Bauer, Cory Endle
Email: Stancl.craig@mayo.edu, bauer.scott@mayo.edu, endle.cory@mayo.edu
Team: LexEVS
Contract: S13-500 MOD4
Client: NCI CBIIT
National Institutes of Heath
US Department of Health and Human Services

Table of Contents

Goal

The goal of the decoupling work is to remove references of the Lucene search implementation from the LexEVS API layer. In the current implementation of LexEVS, Lucene objects are embedded in the LexEVS code base.
Reasons for decoupling are:

- This is a "**best coding practice**" - the search specific implementation of Lucene should not be embedded in the LexEVS code base. This work, whether it is completed or not will have no impact on the overall Lucene 5.0 implementation.
- The decoupling task will move the search specific code to an implementation of a new search interface. If the need ever arose to swap out Lucene for a different search engine, we would be able to create a new implementation of the search interface with the new search engine and not have to make changes in the LexEVS code base.

Recommendation

Based on our review of the code and the large effort needed to complete this decoupling task, we recommend that this decoupling task be lowered in priority and postponed until the main Lucene 5.0 implementation is complete. At this time we can consider if we should take on this task.

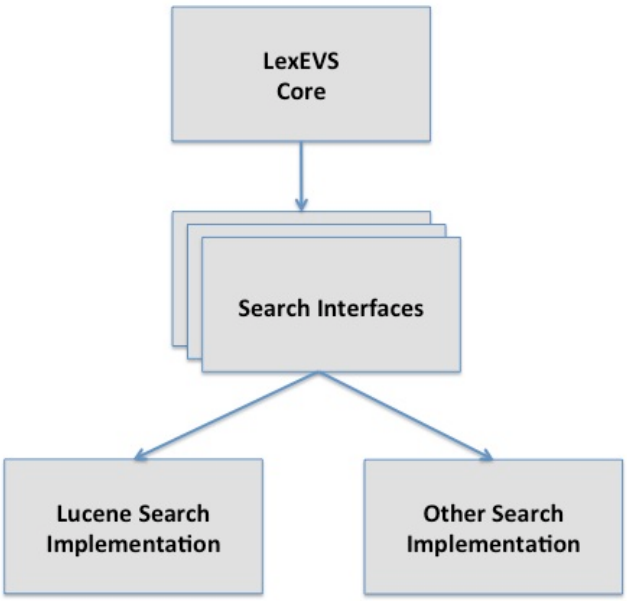
Approach

Before we look at the decoupling task, the approach will be to first update to Lucene 5.0 and fix all of the old Lucene references to get the new Lucene working. We will need to upgrade to Lucene 5.0 first because there will be a lot of Lucene API changes as well as some obsolete Lucene objects that will need to be removed or replaced. If the decoupling task was done before this with the existing Lucene version, we would have to make additional changes once Lucene 5.0 is implemented.

Once the Lucene 5.0 implementation is complete, we should discuss the priority of this task again.

Design

In order to remove the Lucene references from the LexEVS core code base, we will design a new search API interface. We would then be able to create a specific Lucene implementation of this interface. If there was ever a need to substitute different search engine, all that would be necessary would be to create a new implementation of the search API interface for the new search engine. The LexEVS core code base would not be needed to be modified.



There are several classes that will need to evaluated when doing the actual implementation. These cases include code where Lucene objects are intermixed throughout LexEVS methods. (This is not an all inclusive list)

- /lbImpl/src/org/LexGrid/LexBIG/Impl/codednodeset/LuceneOnlyToNodeListCodedNodeSet.java
- /lbImpl/src/org/LexGrid/LexBIG/Impl/codednodeset/UnionSingleLuceneIndexCodedNodeSet.java
- /lbImpl/src/org/LexGrid/LexBIG/Impl/codednodeset/SingleLuceneIndexCodedNodeSet
- org.LexGrid.LexBIG.Impl.helpers.lazyloading/AbstractNonProxyLazyCodeToReturn
- org.LexGrid.LexBIG.Impl.helpers.lazyloading/CommonIndexLazyLoadableCodeToReturn
- org.LexGrid.LexBIG.Impl.helpers.lazyloading/NonProxyCodeHolderFactory
- org.LexGrid.LexBIG.Impl.helpers.lazyloading/NonProxyLazyCodeToReturn
- org.LexGrid.LexBIG.Impl.helpers.lazyloading/AbstractLazyCodeHolderFactory
- org.LexGrid.LexBIG.Impl/CodedNodeSetImpl

This is one example of how an interface could be created to remove Lucene objects from the LexEVS core codebase. These methods can be pushed into an implementation of the Query interface below. The interface would be used instead of calling Lucene directly.

Query Interface

```
//code decoupling

// Interface for creating Queries
public interface Query {

        // methods required for CodedNodeSetImpl
        public  Query getCodingSchemeQuery(String uri, String internalVersionString);
        public  Query getRestrictionQuery(Restriction restriction, String internalCodeSystemName, String
internalVersionString);

        // methods required for AbstractLazyCodeHolderFactory
        private  Query getBooleanQuery(List<Query> queries);
        public Query getFilteredQuery(List<Filter> filters, BooleanQuery combinedQuery, Filter chainedFilter);
}

// Lucene Implementation
public class LuceneQuery implements Query {

}
```

This is another example of an Interface for a ScoreDoc Factory. AbstractLazyCodeHolderFactory.buildCodeHolder is currently using ScoreDocs.

ScoreDocFactory

```
public interface ScoreDocFactory {

     List<ScoreDoc> getScoreDocs (EntityIndexService service, AbsoluteCodingSchemeVersionReference ref,
List<BooleanQuery> combinedQuery,List<Query> bitSetQueries);
}
```

Different types of Queries and Filter types will need to be defined as well. We could create an abstract class for each of them. CodedNodeSetImpl and AbstractLazyCodeHolderFactory will not need to reference Lucene objects directly then.

Abstract QueryType

```
// Potential abstract classes for defining different types of Lucene objects in a generic manner.
public abstract class QueryType {
}

public abstract class FilterType {
}

public abstract class FilteredQuery {
}
```