

LexEVS 5.1 Design Document

Table of Contents

- Design Scope
 - GForge items
 - Solution Architecture
 - High Level Architecture
 - Query Performance and Behavior Enhancements
 - Lucene - Lazy Document Loading
 - Update to Lucene 2.4 Code
 - Searching - Plug-in Search Framework
 - Sorting - Plug-in Sort Framework
 - SQL - SQL query optimizations to increase database performance
 - NCI MetaThesaurus Content (RRF)
 - Value Domain Support
 - Improved Loader Framework
 - BDA Support
 - Cross product dependencies
 - Changes in technology
 - Assumptions
 - Risks
- Detailed Design
 - Query Performance and Behavior Enhancements Detailed Design
 - Lucene Lazy Loading
 - Searching
 - Sorting
 - SQL Optimizations
 - The n+1 SELECTS Problem
 - The n+1 SELECTS Problem Example
 - The n+1 SELECTS Problem Example (Solution)
 - Metathesaurus Content (RRF) Detailed Design
 - Data Model Elements
 - Retrieval and API Documentation
 - MRREL.RRF File
 - MRSAT.RRF
 - MRRANK.RRF
 - MRSAB.RRF
 - MRMAP.RRF, MRSMAP.RRF
 - MRHIER.RRF
 - MRDOC.RRF
 - MRDEF.RRF
 - MRCONSO.RRF

- Value Domain Support Detailed Design
 - Scope
 - Architecture
 - LexGrid Value Domain model
 - Value Domain Definition
 - Value Domain Definition Entry
 - Coding Scheme Reference
 - Value Domain References
 - Entity Reference
 - Definition Operator
 - LexGrid Pick List Model
 - Pick List Definition
 - Pick List Entry Node
 - Pick List Entry
 - Pick List Entry Exclusion
 - Value Domain Definitions Possible Forms
 - Value Domain Resolution
 - Pick List Definitions Possible Forms
 - Pick List Resolution
 - LexEVS Value Domain and Pick List Service Class Diagram
 - Common Services Class Diagram
 - Value Domain Class Diagram
 - Pick List Class Diagram
 - LexBIG Services Class Diagram
 - Main Service API
 - LexBIG API
 - LexEVS Value Domain Service API - Loading Value Domain
 - Validate XML resources
 - Query Value Domain
 - Remove Value Domain Definition
 - Drop Value Domain tables
 - LexEVS Pick List Service API
 - Loading Pick List
 - Validate XML resources
 - Query Pick List
 - Remove Pick List Definition
 - Resolved Value Domain Objects
 - ResolvedValueDomainCodedNodeSet
 - ResolvedValueDomainDefinition
 - Resolved Pick List Objects
 - ResolvedPickListEntry
 - ResolvedPickListEntryList
 - Error Handling
 - Load Scripts
 - Value Domain Loader
 - Pick List Loader
 - Sample XML files
 - Value Domain Definitions
 - Pick List Definitions
 - Database structure
 - Value Domain Tables
 - Pick List Tables
 - Installation / Packaging
 - System Testing
 - Value Domain Service
 - Pick List Service

- Improved Loader Framework Detailed Design
 - Document Purpose
 - Implementation Overview
 - Description
 - Scope
 - Architecture
 - Assumptions
 - Dependencies
 - Issues
 - Third Party Tools
 - Implementation Contents
 - Development and Build Environment
 - How to Use the Loader Framework: A Roadmap
 - Key Directories
 - Algorithms
 - Batch Processes
 - Error Handling
 - Database Changes
 - Client
 - JSP/HTML
 - Servlet
 - Security Issues
 - Performance
 - Internationalization
 - Installation / Packaging
 - Migration
 - Documentation Considerations
 - Testing
 - Test Guidelines
 - Test Cases
 - Test Results
 - Custom Loader Feasibility Report and Recommendation
 - BDA Support Detailed Design
- Implementation Plan
 - Technical environment
 - Software (Technology Stack)
 - Server Hardware
 - Storage
 - Networking
 - External dependencies
 - Team/Location performing development
 - Procedures for Development
 - Detailed schedule
 - Training and documentation requirements
 - Download center changes

Document information

Author: Traci St.Martin/Craig Stancl/Kevin Peterson/Scott Bauer/Sridhar Dwarkanath/Mike Turk
Email: smartin.traci@mayo.edu
Team: LexEVS/EVS
Contract: SAIC Subcontract#28XS112
Client: NCI CBIIT
 National Institutes of Health
 US Department of Health and Human Services

Sign off	Date	Role	CBIIT or Stakeholder Organization	Reviewer's Comments (If disapproved indicate specific areas for improvement.)
—	—	—	—	—

The **purpose of this document** is to collect, analyze, and define high-level needs for and designed features of the National Cancer Institute Center for Biomedical Informatics and Information Technology (NCI CBIIT) caCORE LexEVS Release 5.1. The focus is on the functionalities proposed by the stakeholders and target users to make a better product. The use case documents show in detail how the features meet these needs.

Design Scope

The scope of this release of the LexEVS 5.1 is to support the Metathesaurus Browser project by enhancing search and sorting performance as well as RRF loader changes to more accurately reflect the data. In addition we will explore enhancements to the loader framework and provide a recommendation /implementation for value set/domain support.

Please view the [LexEVS 5.1 Scope document](#).

GForge items

Please view the [LexEVS 5.1 GForge items](#).

Solution Architecture

Proposed technical solution to satisfy the following requirements:

- Query Performance and Behavior Enhancements - Improve the API to support the needs of the Metathesaurus Browser's query response.
- Metathesaurus Content (RRF) - Improve the loader to support full loading of RRF data as necessary for proper operation of the Metathesaurus Browser.
- Value Domain Support - Address an important part of the Semantic Infrastructure that is needed in caBIG.
- Improved Loader Framework - Improve the loading capability and allow loaders to be modular.
- BDA Support - Deployment of LexEVS Project Artifacts to Remote Servers

High Level Architecture

The LexEVS 5.1 infrastructure exhibits an n-tiered architecture with client interfaces, server components, domain objects, data sources, and back-end systems (*Figure 1.1*). This n-tiered system divides tasks or requests among different servers and data stores. This isolates the client from the details of where and how data is retrieved from different data stores.

The system also performs common tasks such as logging and provides a level of security for protected content. Clients (browsers, applications) receive information through designated application programming interfaces (APIs). Java applications communicate with back-end objects via domain objects packaged within the client.jar. Non-Java applications can communicate via SOAP (Simple Object Access Protocol) or REST (Representational State Transfer) services.

Most of the LexEVS API infrastructure is written in the Java programming language and leverages reusable, third-party components. The service infrastructure is composed of the following layers:

Application Service layer - accepts incoming requests from all public interfaces and translates them, as required, to Java calls in terms of the native LexEVS API. Non-SDK queries are invoked against the Distributed LexEVS API, which handles client authentication and acts as proxy to invoke the equivalent function against the LexEVS core Java API. The caGrid and SDK-generated services are optionally run in an application server separate from the Distributed LexEVS API.

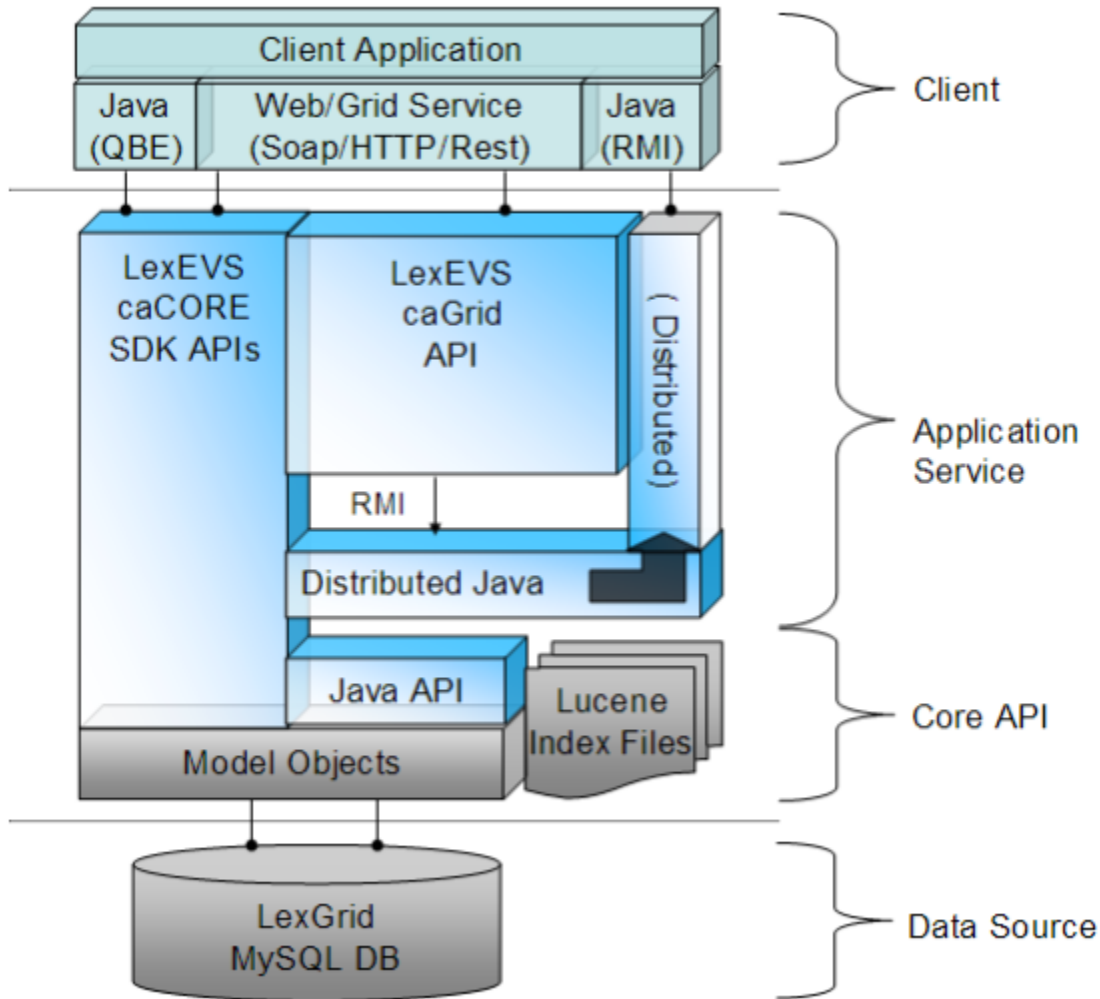
The LexEVS caCORE SDK services work directly against the database, via Hibernate bindings, to resolve stored objects without intermediate translation of calls in terms of the LexEVS API. However, the LexEVS SDK services do still require access to metadata and security information stored by the Distributed and Core LexEVS API environment to resolve the specific database location for requested objects and to verify access to protected resources, respectively.

From the client perspective, the LexEVS services will function as "ports" accessible through the caGrid 1.3 service architectural model. LexEVS services will follow the caGrid architecture for analytical and data services. See the caGrid 1.3 documentation for architectural details: <https://cabig.nci.nih.gov/workspaces/Architecture/caGrid/>

Core API layer - underpins all LexEVS API requests. Search of pre-populated Lucene index files is used to evaluate query results before incurring cost of database access. Access to the LexGrid database is performed as required to populate returned objects using pooled connections.

Data Source layer--- is responsible for storage and access to all data required to represent the objects returned through API invocation.

High Level Design Diagram



Query Performance and Behavior Enhancements

Lucene - Lazy Document Loading

Lucene is very fast as a search engine. Given a text string, Lucene can find matching documents in huge indexes very fast. This is the purpose and strength of Lucene. Lucene is not, however, a database. Retrieving information from the documents that the search found as 'hits' is slow.

Consider this scenario: A user searches for 'heart' in the NCI MetaThesaurus. When Lucene does its search, it will return probably 50,000+ 'hits'. This search is done very fast. LexEVS previously would retrieve all of those documents to populate the ResolvedConceptReference. Retrieving this many documents from Lucene is slow.

The solution is to lazy load the documents as needed. After the Lucene search is complete, we only store the Document Id. Then, when information from the document is needed, it is retrieved from the document. This is helpful in Iterator-type scenarios, where retrieval can be done one at a time.

Update to Lucene 2.4 Code

As we move forward, it is important to keep current with the latest Lucene API. Not only is this important for performance reasons -- it will limit our ability to upgrade our Lucene dependencies if we rely on deprecated methods.

Searching - Plug-in Search Framework

We advertise our Searches as being 'extensions', but in reality it is very difficult (or impossible) for a user to create a plug-in type Search.

The Interface **org.LexGrid.LexBIG.Extensions.Query.Search** will be introduced. The purpose of this interface is to give users a plug-in type Interface to implement different search strategies. This interface will accept a text query string and output a Lucene Query.

Sorting - Plug-in Sort Framework

As with Searching, Sort algorithms are not currently easily extended. A well defined and 'Extension-ready' interface would allow users to add additional search functionality on demand, without rebuilding or recompiling.

The existing Interface **org.LexGrid.LexBIG.Extensions.Query.Search** will be expanded to allow for easy implementation and flexibility, allowing rapid creation of new Sort Algorithms and techniques.

SQL - SQL query optimizations to increase database performance

Join EntityDescription when building AssociatedConcepts

The 'EntityDescription' field of 'Entity' is being retrieved with a separate SQL call. This will allow the building of AssociatedConcepts with minimal calls to the database.

Furthermore, this will allow the 'EntityDescription' to be available without requiring the actual 'CodedEntry' to be resolved. For most usescases, this should enable users to resolve Graphs with 'CodedEntryDepth=0'. Avoiding any resolving of the CodedEntry will keep resolve times to a minimum.

Join EntryState when building CodedEntry

The EntryState is now populated with a separate SQL SELECT query to the database. This results in one SELECT statement per CodedEntry returned - and there is potential for a large number of CodedEntries to be resolved at once. Populating this with a JOIN instead of a SELECT will be more efficient and not require additional unnecessary SELECT queries to the database.

NCI MetaThesaurus Content (RRF)

Loads of the NCI MetaThesaurus RRF formatted data into the LexGrid model require a number of loader adjustments in order to accurately reflect the state of the data as it exists in the current RRF files. No model or API changes will be necessary to accommodate the data; changes will be made directly to the loader.

Value Domain Support

The LexEVS Value Domain and Pick List service will provide ability to load Value Domain and Pick List Definitions into LexGrid repository and provides ability to apply user restrictions and dynamically resolve the definitions during run time. Both Value Domain and Pick List service are integrated part of LexEVS core API.

The LexEVS Value Domain and Pick List service will provide programmatic access to load Value Domain and Pick List Definitions using the domain objects that are available via the LexGrid logical model. The LexEVS Value Domain and Pick List service will provide ability to apply certain user restrictions (ex: pickListId, valueDomain URI etc) and dynamically resolve the Value Domain and Pick List definitions during the run time

The LexEVS Value Domain and Pick List Service meant to expose the API particularly for the Value Domain and Pick List elements of the LexGrid Logical Model. For more information on LexGrid model see <http://informatics.mayo.edu/>

Improved Loader Framework

LexEVS already provides a set of loaders within an existing legacy framework which has served LexEVS developers well over the years. But as LexEVS has gained users, and requests for new loaders has grown , it was decided that a new Loader Framework should be developed. The new framework: provides classes and interfaces that are more modular and easier to extend; improved loader performance; allows dynamic loading of new loaders; is built upon proven open source technologies such as SpringBatch and Hibernate, and finally, the new Loader Framework code is completely independent of the current loader code in LexEVS so there will be no impact to current loaders.

BDA Support

LexEVS uses the BDA (Build and Deployment Automation) system to build and deploy artifacts. This build script that produces these artifacts and deploys them is kicked off via a build server (an instance of Anthill pro).

Cross product dependencies

Include a link to the [Core Product Dependency Matrix](#).

Changes in technology

Include any new dependencies in the [Core Product Dependency Matrix](#) and summarize them here.

- No new dependencies exist.

Assumptions

List any assumptions.

Risks

- Increased load time and storage requirements due to additional Meta (RRF) content.
 - Due to the additional content to load from RRF files, there is a risk of increased loading times and increased storage requirements. The new loader framework should mitigate the increased loading times (providing a faster load while increasing content to be loaded).

Detailed Design

Specify how the solution architecture will satisfy the requirements. This should include high level descriptions of program logic (for example in structured English), identifying container services to be used, and so on.

Query Performance and Behavior Enhancements Detailed Design

Lucene Lazy Loading

Background - Lucene Documents

Lucene stores information in Documents, and these Documents have Fields that are used to hold information. Each Document has a unique id.

For example, an index of People may be indexed in Lucene as:

```
Document: id 1
First Name: John
Last Name: Doe
Sex: Male
Age: 45

Document: id 2
First Name: Jane
Last Name: Doe
Sex: Female
Age: 40

... etc.
```

LexEVS stores information about Entities in this way. Property names and values, as well as Qualifiers, Language, and various other information about the Entity are held in Lucene indexes.

Background - Querying Lucene

Lucene provides a Query mechanism to search through the indexed documents. Given a search query, Lucene will provide the Document id and the score of the match (Lucene assigns every match a 'score', depending on the strength of the match given the query).

So, if the above index is queried for "First Name = Jane AND Last Name = Doe", the result will be the Document id of the match (2), and the score of the match (a float number, usually between 1 and 10).

Notice that none of the other information is returned, such as Sex or Age. It is useful for that extra information to be there, because if it exists in the Lucene indexes we do not have to make a database query for it. BUT, retrieving data from Lucene Documents is expensive, just as retrieving data from a database would be.

- Lazy Retrieval

We can leverage this to increase performance in LexEVS. Consider this simplified LexEVS Entity index:

```
Document: id 1
Code: C12345
Name: Heart

Document: id 2
Code: C67890
Name: Foot

Document: id 3
Code: C98765
Name: Heart Attack
```

If a user constructs a Query (Name = Heart*), the query will return with the matching Document ids (1 and 2). Previously, LexEVS would immediately retrieve the 'Code' and 'Name' fields from the matches, and use them to construct the results that would be ultimately returned to the user. This does not scale well, especially for general queries in large ontologies. In a large ontology, a Query of (Name = Heart*) may match tens of thousands of Documents. Retrieving the information from all these Documents is a significant performance concern.

Instead of retrieving the information up front, LexEVS will simply store the Document id for later use. When this information is actually needed by the user (for example, the information needs to be displayed), it is retrieved on demand.

Searching

To allow users to plug in custom search algorithms, the LexEVS Extension framework needed to be extended to include Searches.

The **org.LexGrid.LexBIG.Extensions.Extendable.Search** interface consists of one method to be implemented:

Class:	org.LexGrid.LexBIG.Extensions.Extendable.Search
Method:	public org.apache.lucene.search.Query buildQuery(String searchText)
Description:	<i>Given a String search string, build a Query object to match indexed Lucene Documents</i>

This enables the user to construct any type of Query given search text. Wildcards may be added, search terms may be grouped, etc.

AND vs. OR

Previously, for most search algorithms Lucene applied an 'OR' to the terms if multiple terms were input as search text. For example, a search of 'heart attack' would match all documents containing 'heart' OR all documents containing 'attack'. This lead to non-intuitive results being returned to the user. Changing Lucene to default to an 'AND' type strategy will increase search precision and in most cases shrink the amount of results returned for a given query, which will in turn increase overall performance.

Algorithms

More precise DoubleMetaphoneQuery

DoubleMetaphoneQueries enable the user to input incorrectly spelled search text, while still returning results. Because this is a 'fuzzy' search, it is important to structure the Query in a way that the most appropriate results are returned to the user first.

For example, the Metaphone computed value for "Breast" and "Prostrate" is the same. Given the search term "Breast", both "Breast" and "Prostrate" will match with exactly the same score. Technically, this is correct behavior, but to the end user this is not desirable. To overcome this, we have introduced a new query, WeightedDoubleMetaphoneQuery.

WeightedDoubleMetaphoneQuery

This algorithm does not automatically assume that the user has spelled the terms incorrectly. Searches are also based on the actual text that the user has input, along with the Metaphone value.

Again, if the user input "Breast", the query will still match "Breast" and "Prostrate", but "Breast" will have a higher match score, because the actual user text is considered. This will add a greater precision to this fuzzy-type query.

Algorithm:

```
get: user text input
2: total score = 0
3: metaphone score = 0
4: actual score = 0
5: metaphone value = lucene.computeMetaphoneValue(user text input)
6: metaphone score = lucene.scoreMetaphoneValue(metaphone value)
7: actual score = lucene.score(user text input)
8: total score = metaphone score + actual score
9: halt
```

Case-insensitive substring

SubStringSearch - This algorithm is intended to find substrings within a large string. For example:

"with a heart attack"

Will match:

"The patient **with a heart attack** was seen today".

Also, a leading and trailing wildcard will be added, so

"th a heart atta"

Will also match:

"The patient **with a heart attack** was seen today".

Algorithm:

```
get: user text input
2: user text input = '*' + user text input + '*'
3: score = lucene.score(user text input)
4: halt
```

Sorting

Sorting matched results is important part of interacting with the LexEVS API. Allowing users to plug in customized Sort algorithms helps LexEVS to be more flexible to more groups of users. To implement a Sorting algorithm, a user must implement the org.LexGrid.LexBIG.Extensions.Extendable.Sort Interface.

Class:	org.LexGrid.LexBIG.Extensions.Extendable.Sort
---------------	--

Method:	public <T> Comparator<T> getComparatorForSearchClass(Class<T> searchClass) throws LBParameterException Description: <i>Given a Class that this Sort is valid for, return the correct Comparator to compare the results and sort.</i>
Method:	public boolean isSortValidForClass(Class<?> clazz); Description: <i>Return whether or not this Sort is valid for Sorting on a given Class</i>

- Sorting on Different Class types
 - A single Sort may be applicable for a variety of Class types. For instance, both an 'Association' and an 'Entity' may be sorted by 'Code', but the actual implementation of retrieving the Code and comparing it may be different between the two. It is the job of the Sort to implement a Comparator for each potential Class that it is eligible to sort.
- Default Sorting
 - All result sets are sorted by default by Lucene Score, meaning that the best match according to Lucene will always be returned first by default. Note that if two or more result sets are being Unioned, Intersected, or Differenced, the user must explicitly call a 'matchToQuery' sort on the result set as a whole to order all of the results.
- Sort Contexts
 - Sorts may be applicable in one or more 'Contexts' (see: org.LexGrid.LexBIG.DataModel.InterfaceElements.types.SortContext) This means that a Sort may apply only to a CodedNodeSet, or only to a CodedNodeGraph, or some combination. Sorts will only be employed by the API if they match the Context in which the results are being sorted.
- Performance Issues
 - Sorting is generally computationally expensive, because in order to correctly sort, the field to be sorted has to be fully retrieved for the entire result set. For very specific or refined queries, this may not be a problem, but for large ontologies or very general queries, performance may be a concern. To alleviate this, 'Post sort' has been introduced.
- Post Sorting
 - In order to minimize the performance impact of sorting, users are encouraged to use a 'Post sort' where possible. A Post sort is done after the result set has been restricted, thus limiting the amount of information that must be retrieved in order to perform the sort. For instance, a query may match a set of Entities:

```
{"Heart", "Heart Failure", "Heart Attack", "Arm", "Finger", ...}
```

As described earlier, all results are by default sorted by Lucene score, so if we limit the result set to the top 3, the result is:

```
{"Heart", "Heart Failure", "Heart Attack"}
```

The restricted set can then be 'Post' sorted - and because the result set has been limited to a reasonable number of matches, sorting and retrieval time can be minimized.

Algorithm:

```
1: get: Sort requested by user
2: get: Context sort is being applied to
3: if: sort is not valid for Context
halt
4: else:
5: get: Class to be sorted on
6: if: sort is not valid for Class
halt
7: get: Comparator for Sort - given (Class to be sorted on)
8: sort results using Comparator for Sort
9: halt
```

SQL Optimizations

The n+1 SELECTS Problem

The n+1 SELECTS Problem refers to how information can optimally be retrieved from the database, preferably using as few queries as possible. This is desirable because:

- Query overhead is a concern. Every query must be packaged and sent to the database engine, processed, packaged again and transferred to the client. Although the overhead may be minimal (a few milliseconds), it does not scale.

To avoid this, a JOIN query can be used.

The n+1 SELECTS Problem Example

Given two database tables, retrieve the Code, Name, and Qualifier for each Code

Table Codes

Code	Name
C01234	Heart
C98765	Heart Attack

Table Qualifiers

Code	Qualifier
C01234	isAnOrgan
C98765	isADisease

```
SELECT * FROM Codes
```

Results in:

Code	Name
C01234	Heart
C98765	Heart Attack

To get the Qualifiers, separate SELECTs must be used for each.

```
SELECT * FROM Qualifiers where Code = C01234
And
SELECT * FROM Qualifiers where Code = C98765
```

This sequence results in 1 Query to retrieve the data from the Codes table, and then n Queries from the Qualifiers table. This results in n+1 total Queries.

The n+1 SELECTS Problem Example (Solution)

Given two database tables, retrieve the Code, Name, and Qualifier for each Code

Table Codes

Code	Name
C01234	Heart
C98765	Heart Attack

Table Qualifiers

Code	Qualifier
C01234	isAnOrgan
C98765	isADisease

```
SELECT * FROM Codes JOIN Qualifiers ON Code
```

Results in:

Code	Name	Qualifier
C01234	Heart	isAnOrgan
C98765	Heart Attack	isADisease

Because of the JOIN, only one Query is needed to retrieve all of the data from the database.

Although sometimes obvious, n+1 queries can remain in a system undetected until scaling problems are noticed.

In LexEVS there were 3 n+1 SELECT queries fixed:

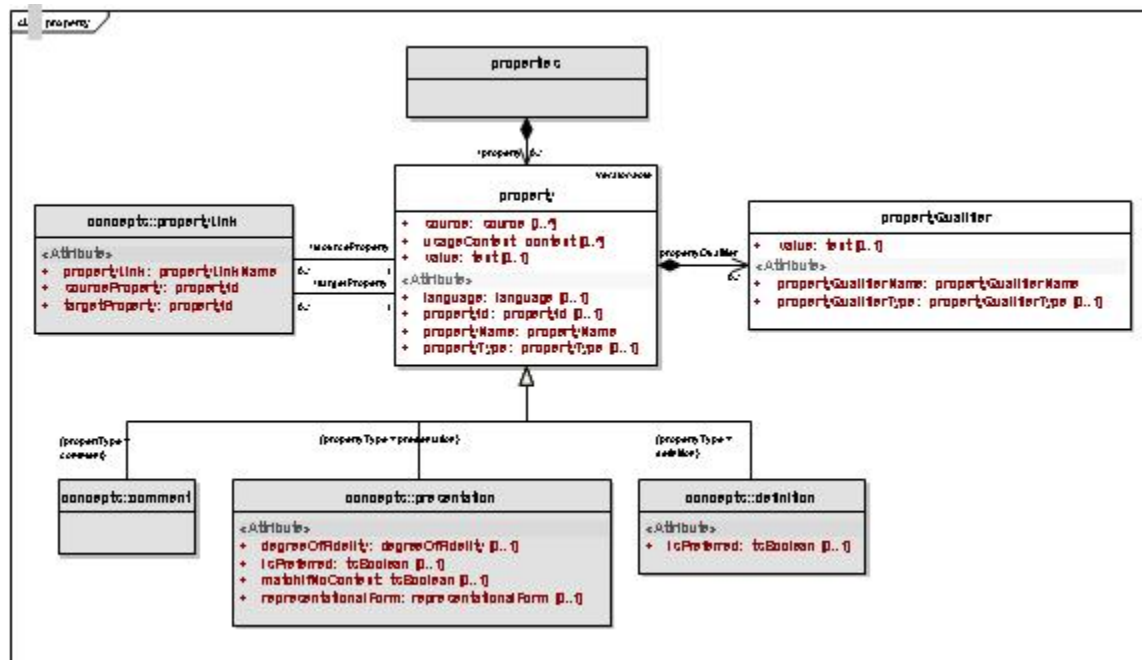
- The EntryState while building the CodedEntry.
- The EntityDescription on AssociatedConcepts
- AssociationQualifiers on AssociatedConcepts

Metathesaurus Content (RRF) Detailed Design

Loads of the NCI MetaThesaurus RRF formatted data into the LexGrid model require a number of adjustments in order to accurately reflect the state of the data as it exists in the current RRF files.

Data Model Elements

Most data elements will be loaded as either properties or property qualifiers:



A few will be loaded as qualifiers to associations.

Retrieval and API Documentation

No new API retrieval methods will be implemented in the scope of LexEVS 5.1. However, some may be required in the scope of 6.0 for any mapping elements implemented as new model elements or model extensions to LexGrid. No changes to user interfaces will occur. Service methods for loading these elements will be consistent with the new Spring Batch loader framework.

MRREL.RRF File

Problem:

REL and RELA column elements from the RRF source need to be connected. Currently these are loaded as separate relationships preventing the user from connecting to the REL/RELA combinations that actually occur in the NCI-META (e.g. RELA may be different for same REL value in different sources).

Requirement:

A single relationship should be loaded for a REL/RELA combination for a particular SAB between two CUIs.

Solution:

Since RELA type RRF elements have been defined as relationship names specific to sources and not independent relationships themselves, these elements will be loaded as association qualifiers in the LexGrid model.

Problem and Requirement:

User is unable to distinguish individual relationships from one source or another. The same association "entity" exists only once but has two "source" qualifiers.

User is unable to distinguish the AUI1/STYPE1 and AUI2/STYPE2 which gives us the information about what source data structures are actually being connected by MRREL entries. Users also need the ability to associate AUI/STYPE fields with SAB.

Users sole choice for rendering a relationship in terms of the strings on either side is to use preferred concept names.

Proposed Solution:

Propose AUI to AUI - the way CUI to CUI are currently handled in the implementation.
Propose entity to entity relationship - will still have to account for CUI to CUI relationships.
Load each unique RUI (would be quite large). They would need to be listed as supported association (this is not traditional how it is used).

Load supporting column elements from MRREL.RRF including contents of:
AUI1, STYPE1, AUI2, STYPE2, SRUI, SAB, RG, SUPPRESS, CVF, RUI

These will be available as elements of the overriding Metathesaurus Association and loaded as association qualifiers

Problem:

Self Referencing Relationships (CUI1 = CUI2) cannot be fully represented in our model. Previously, these were loaded as PropertyLinks. This fit into the LexEVS model well, but left out important RRF information. Most notably, PropertyLinks cannot contain Qualifiers like normal relations can. Because of the increased number of Qualifiers that are required to be placed on relations, much information would be lost representing these relations as PropertyLinks

Solution:

Do not treat a CUI1 = CUI2 relationships differently than a CUI1 != CUI2 relationship. For API and query purposes, qualify these relationships with a 'selfReferencing=true' Qualifier. In this way, we can still avoid cycles in the API, but maintain all relevant Qualifier information in the relation.

MRSAT.RRF**Problem:**

MRSAT.RRF is not loaded but only accessed for given preferred term algorithms. This data should be loaded as concept properties (STYPE=CUI), properties on properties (STYPE=AUI, SAUI, CODE, SCUI, SDUI), qualifiers on associations (STYPE=RUI,SRUI). Some complexity may arise as concept properties can have additional qualifiers, but property-properties cannot and association-qualifiers cannot.

Requirement:

If the STYPE is something other than RUI or SRUI, you can load that row as an entity property. The fields you'd want to capture are:

CUI - We use this as the entityCode and is loaded as such in the table.

METAUI - load as a propertyQualifier (name=METAUI, value)

STYPE - load as a propertyQualifier (name=STYPE, value)

ATUI - load as propertyId

ATN - load as property name

SAB - load as a propertyQualifier (typeName=source)

ATV- load as a propertyValue

SUPPRESS - load as propertyQualifier if value != N

MRRANK.RRF**Problem:**

SAB specific ranking of representational form in MRRANK is not exposed to the user (used in an underlying ranking and specifying of preferred presentations for a given concept)

Requirement:

Load elements of MRRANK so that they are available to the user.

Proposed Solution:

Load MRRANK as property qualifier on Presentation type property with the property Name of "mrrank."

Retrieval:

Available in current LexEVS api

MRSAB.RRF**Problem:**

MRSAB.RRF file data is not loaded or is otherwise unavailable to the user.

Requirement:

Load MRSAB.RRF file data as metadata

Implemented Solution:

Entire content of each row of MRSAB file is loaded as metadata to an external xml file with tags created from column names and value inserted between tags as is appropriate

MRMAP.RRF, MRSMAP.RRF

Problem:

MRMAP.RRF source load is not supported in current load. Currently this RRF file is not populated in NCI Metathesaurus distributions. Mapping is not explicitly supported in the LexGrid Model.

Requirement:

Load MRMAP data.

Solution:

To be evaluated for a load to current model elements or possible new model mapping elements. The general agreement is that this is more appropriately implemented in 6.0.

MRHIER.RRF

Problem:

HCD is loaded as a property on the presentation but the SAB isn't associated with it so we do not know the source of the HCD. (only look at row that has HCD field populated)
Path to Root, (PTR) is also not loaded, but is instead used to determine path to root operations in LexEVS.

Requirement:

These elements need to be loaded and available from the LexEVS api

Solution:

Load HCD associated field SAB as property qualifier when HCD is present. Load PTR as property.

MRDOC.RRF

Problem:

MRDOC contains metadata unavailable to the user. It is not loaded by LexEVS.

Requirement:

This metadata will be made available to the user.

Solution:

MRDOC's column names and content will be processed as tag/value mappings to a metadata file.

MRDEF.RRF

Problem:

Some values from each row are not loaded by LexEVS.

Requirement:

AUI should be loaded to connect it with the presentation
ATUI, SUPPRESS, CVF, SATAUI should be loaded and exposed to the user.

ATUI, SUPPRESS, CVF, SATAUI, column values will be loaded as property qualifiers on the Definition type property derived from MRDEF column.

MRCONSO.RRF

Problem:

Some elements from the columns of MRCONSO.RRF are not loaded by LexEVS.

Requirement:

Load LUI, SUI, SAUI, SDUI, SUPPRESS, CVS fields and expose to the user.

Solution:

All noted values will be loaded as property qualifiers.

Value Domain Support Detailed Design

The LexEVS Value Domain and Pick List service will provide ability to load Value Domain and Pick List Definitions into LexGrid repository and provides ability to apply user restrictions and dynamically resolve the definitions during run time. Both Value Domain and Pick List service are integrated part of LexEVS core API.

Scope

The LexEVS Value Domain and Pick List service will provide programmatic access to load Value Domain and Pick List Definitions using the domain objects that are available via the LexGrid logical model.

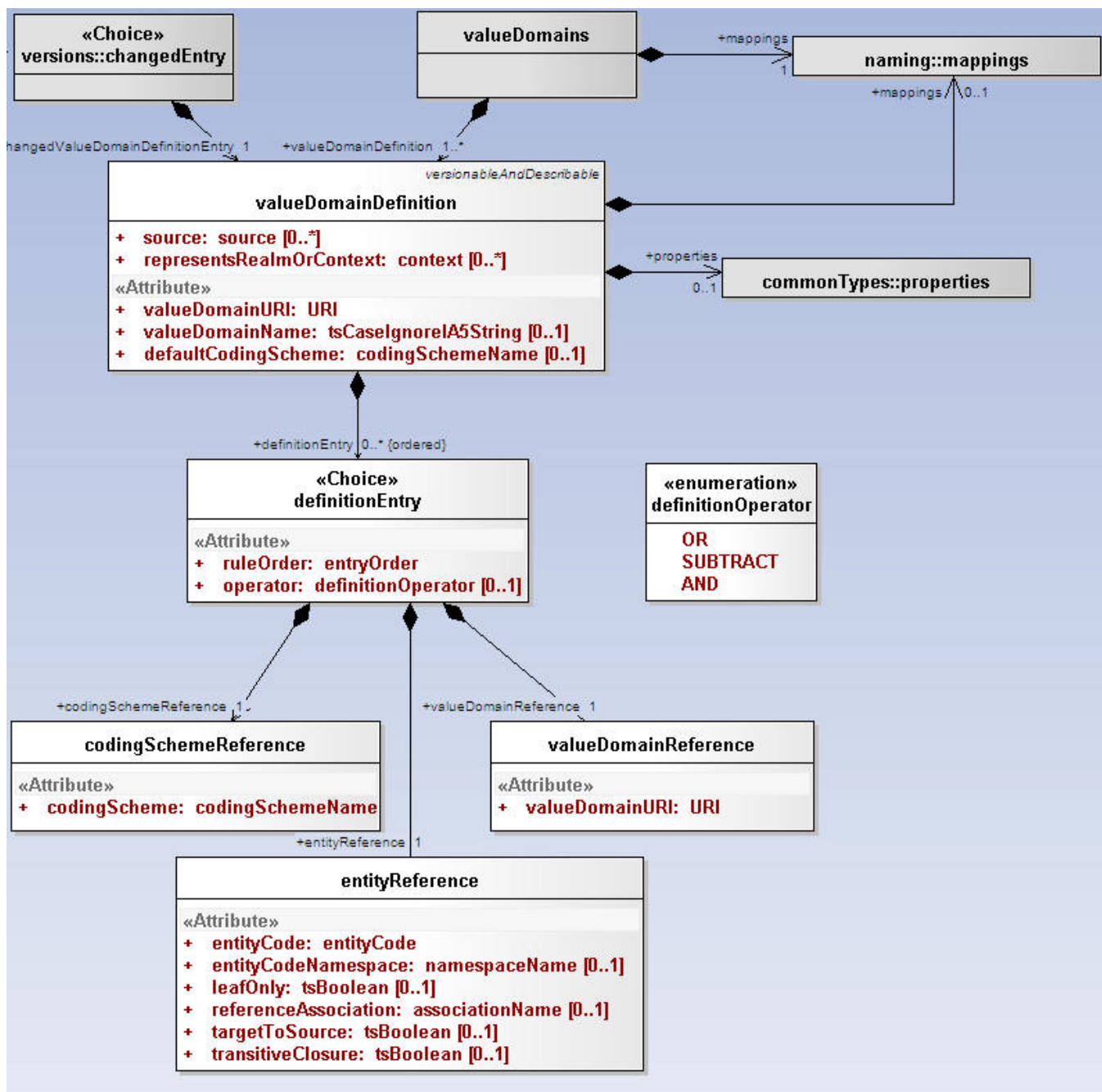
The LexEVS Value Domain and Pick List service will provide ability to apply certain user restrictions (ex: pickListId, valueDomain URI etc) and dynamically resolve the Value Domain and Pick List definitions during the run time.

Architecture

The LexEVS Value Domain and Pick List Service meant to expose the API particularly for the Value Domain and Pick List elements of the LexGrid Logical Model. For more information on LexGrid model see <http://informatics.mayo.edu/>

LexGrid Value Domain model

Here is a UML representation of Value Domain within LexGrid 200901 model.



Value Domain Definition

A definition of a given value domain. A value domain can be a simple description with no associated value domain entries, or it can consist of one or more definitionEntries that resolve to an enumerated list of entityCodes when applied to one or more codingScheme versions.

Attributes of Value Domain Definition :

Source: The local identifiers of the source(s) of this property. Must match a local id of a supportedSource in the corresponding mappings section.

representsRealmOrContext: The local identifiers of the context(s) in which this value domain applies. Must match a local id of a supportedContext in the corresponding mappings section.

valueDomainURI: The URI of this value domain.

valueDomainName: The name of this domain, if any.

defaultCodingScheme: Local name of the primary coding scheme from which the domain is drawn. defaultCodingScheme must match a local id of a supportedCodingScheme in the mappings section.

Value Domain Definition Entry

A reference to an entry code, a coding scheme or another value domain along with the instructions about how the reference is applied. Definition entries are applied in entryOrder, with each successive entry either adding to or subtracting from the final set of entity codes.

Attributes of Value Domain Definition Entry :

ruleOrder: The unique identifier of the definition entry within the definition as well as the relative order in which this entry should be applied

operator: How this entry is to be applied to the value domain

Coding Scheme Reference

A reference to all of the entity codes in a given coding scheme.

Attributes of Coding Scheme Reference :

codingScheme: The local identifier of the coding scheme that the entity codes are drawn from . codingSchemeName must match a local id of a supportedCodingScheme in the mappings section.

Value Domain References

A reference to the set of codes defined in another value domain.

Attributes of Value Domain Reference :

valueDomainURI: The URI of the value domain to apply the operator to. This value domain may be contained within the local service or may need to be resolved externally.

Entity Reference

A reference to an entityCode and/or one or more entityCodes that have a relationship to the specified entity code.

Attributes of Entity Reference :

entityCode: The entity code being referenced.

entityCodeNamespace: Local identifier of the namespace of the entityCode. entityCodeNamespace must match a local id of a supportedNamespace in the corresponding mappings section. If omitted, the URI of the defaultCodingScheme will be used as the URI of the entity code.

leafOnly: If true and referenceAssociation is supplied and referenceAssociation is defined as transitive, include all entity codes that are "leaves" in transitive closure of referenceAssociation as applied to entity code. Default: false

referenceAssociation: The local identifier of an association that appears in the native relations collection in the default coding scheme. This association is used to describe a set of entity codes. If absent, only the entityCode itself is included in this definition.

targetToSource: If true and referenceAssociation is supplied, navigate from entityCode as the association target to the corresponding sources. If transitiveClosure is true and the referenceAssociation is transitive, include all the ancestors in the list rather than just the direct "parents" (sources).

transitiveClosure: If true and referenceAssociation is supplied and referenceAssociation is defined as transitive, include all entity codes that belong to transitive closure of referenceAssociation as applied to entity code. Default: false

Definition Operator

The description of how a given definition entry is applied.

Attributes of Definition Operator :

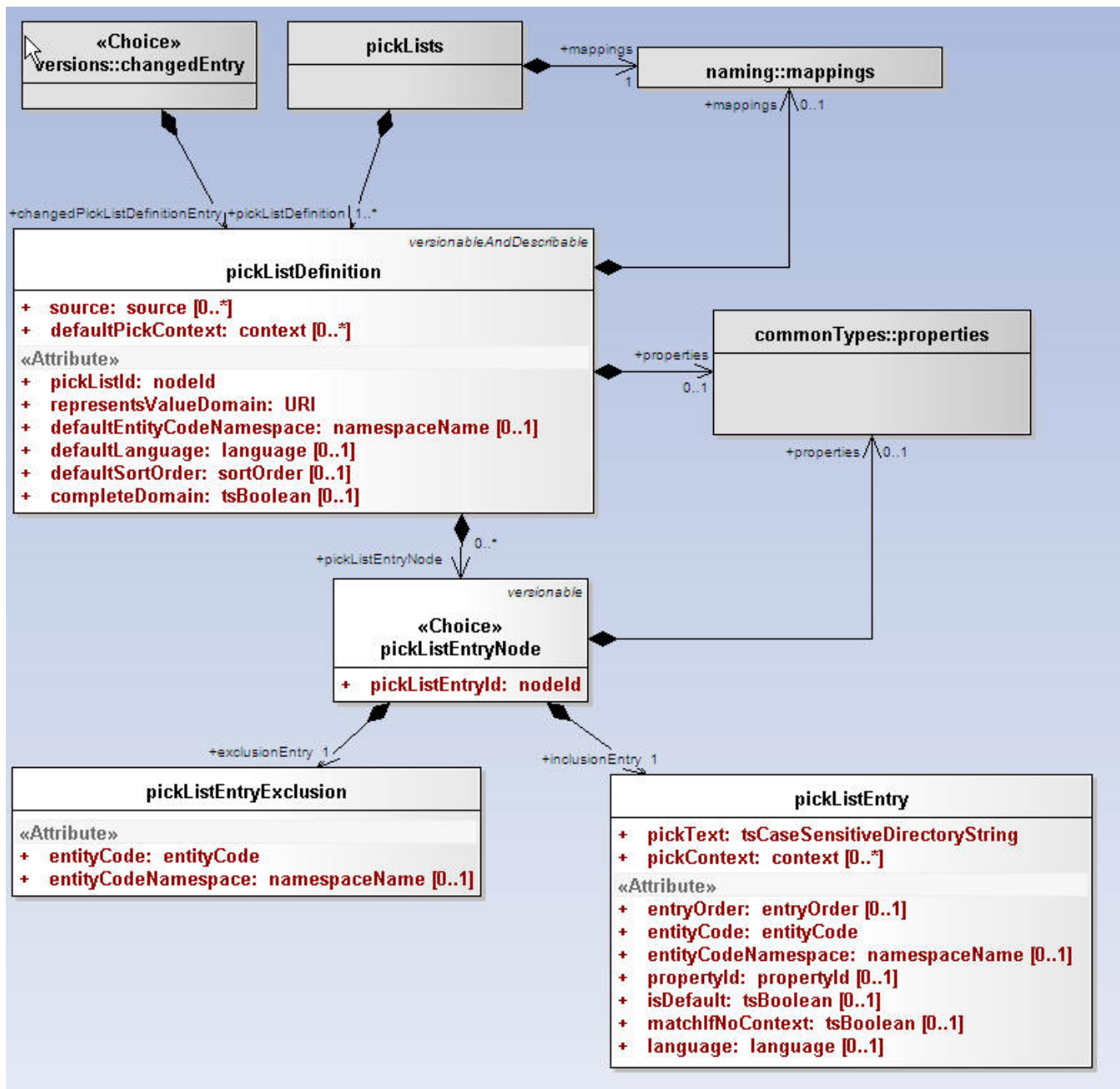
OR: Add the set of entityCodes described by the currentEntity to the value domain. (logical OR)

SUBTRACT: Subtract (remove) the set of entityCodes described by the currentEntity to the value domain. (logical NAND)

AND: Only include the entity codes that are both in the value domain and the definition entry. (logical AND)

LexGrid Pick List Model

Here is a UML representation of Pick List within LexGrid 200901 model:



Pick List Definition

An ordered list of entity codes and corresponding presentations drawn from a value domain.

Attributes of Pick List Definition :

Source: The local identifiers of the source(s) of this pick list definition. Must match a local id of a supportedSource in the corresponding mappings section.

pickListId: An identifier that uniquely names this list within the context of the collection.

representsValueDomain: The URI of the value domain definition that is represented by this pick list

defaultEntityCodeNamespace: Local name of the namespace to which the entry codes in this list belong. defaultEntityCodeNamespace must match a local id of a supportedNamespace in the mappings section.

defaultLanguage: The local identifier of the language that is used to generate the text of this pick list if not otherwise specified. Note that this language does NOT necessarily have any correlation with the language of a pickListEntry itself or the language of the target user. defaultLanguage must match a local id of a supportedLanguage in the mappings section.

defaultSortOrder: The local identifier of a sort order that is used as the default in the definition of the pick list

defaultPickContext: The local identifiers of the context used in the definition of the pick list.

completeDomain: True means that this pick list should represent all of the entries in the domain. Any active entity codes that aren't in the specific pick list entries are added to the end, using the designations identified by the defaultLanguage, defaultSortOrder and defaultPickContext. Default: false

Pick List Entry Node

An inclusion (pickListEntry) or exclusion (pickListEntryExclusion) in a pick list definition

Attributes of Pick List Entry Node :

pickListEntryId: Unique identifier of this node within the list.

Pick List Entry

An entity code and corresponding textual representation.

Attributes of Pick List Entry:

pickText: The text that represents this node in the pick list. Some business rules may require that this string match a presentation associated with the entityCode

pickContext: The local identifiers of the context(s) in which this entry applies. pickContext must match a local id of a supportedContext in the mappings section

entryOrder: Relative order of this entry in the list. pickListEntries without a supplied order follow the all entries with an order, and the order is not defined.

entityCode: Entity code associated with this entry.

entityCodeNamespace: Local identifier of the namespace of the entity code if different than the pickListDefinition defaultEntityCodeNamespace. entityCodeNamespace must match a local id of a supportedNamespace in the mappings section.

propertyId: The property identifier associated with the entityCode and entityCodeNamespace that the pickText was derived from. If absent, the pick text can be anything. Some terminologies may have business rules requiring this attribute to be present.

isDefault: True means that this is the default entry for the supplied language and context.

matchIfNoContext: True means that this entry can be used if no contexts are supplied, even though pickContext is present.

Language: The local name of the language to be used when the application/user supplies a selection language matches. If absent, this matches all languages. language must match a local id of a supportedLanguage in the mappings section.

Pick List Entry Exclusion

An entity code that is explicitly excluded from a pick list.

Attributes of Pick List Entry Exclusion:

entityCode: Entity code associated with this entry.

entityCodeNamespace: Local identifier of the namespace of the entity code if different than the pickListDefinition defaultEntityCodeNamespace. entityCodeNamespace must match a local id of a supportedNamespace in the mappings section.

Value Domain Definitions Possible Forms

- Code system/concept code + relationship + additional rules (leaf only, immediate children, etc)
- Code system - all concept codes in the system
- Code system/concept code - individual code
- Combination of any of the above with OR/AND/SUBTRACT operators

Value Domain Resolution

- A value domain definition has to be made against a specific version of a code system. But it doesn't have to be resolved against the same version.
- Even a simple list (a,b,c,d) needs to be resolved as, at some future date, "c" might be retired.
- Resolution does not create static artifact.

Pick List Definitions Possible Forms

- Value Domain - all concept codes in the value domain
- Code system/concept code - individual code (inclusion and exclusion)

Pick List Resolution

- A picklist definition has to be made against a specific value domain.
- Even a simple list (a,b,c,d) needs to be resolved as, at some future date, "c" might be retired.
- Resolution does not create static artifact.

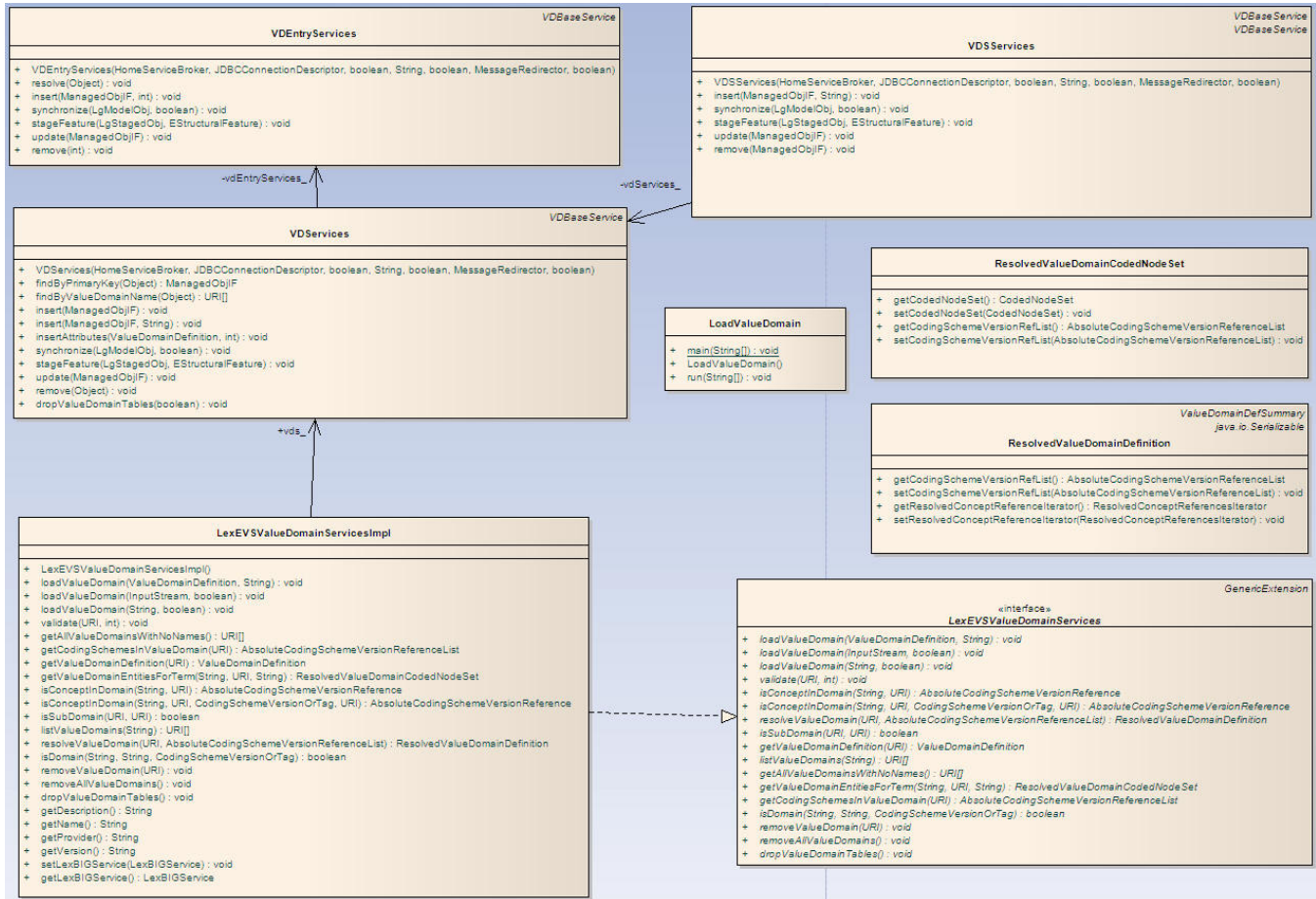
LexEVS Value Domain and Pick List Service Class Diagram

Common Services Class Diagram

These are the classes that are used commonly across Value Domain and Pick List implementation.

Class Name	Description
VDEntryTypeServices	Class to handle Entry Type objects to and fro database .
VDEntryStateServices	Class to handle Entry State objects to and fro database.

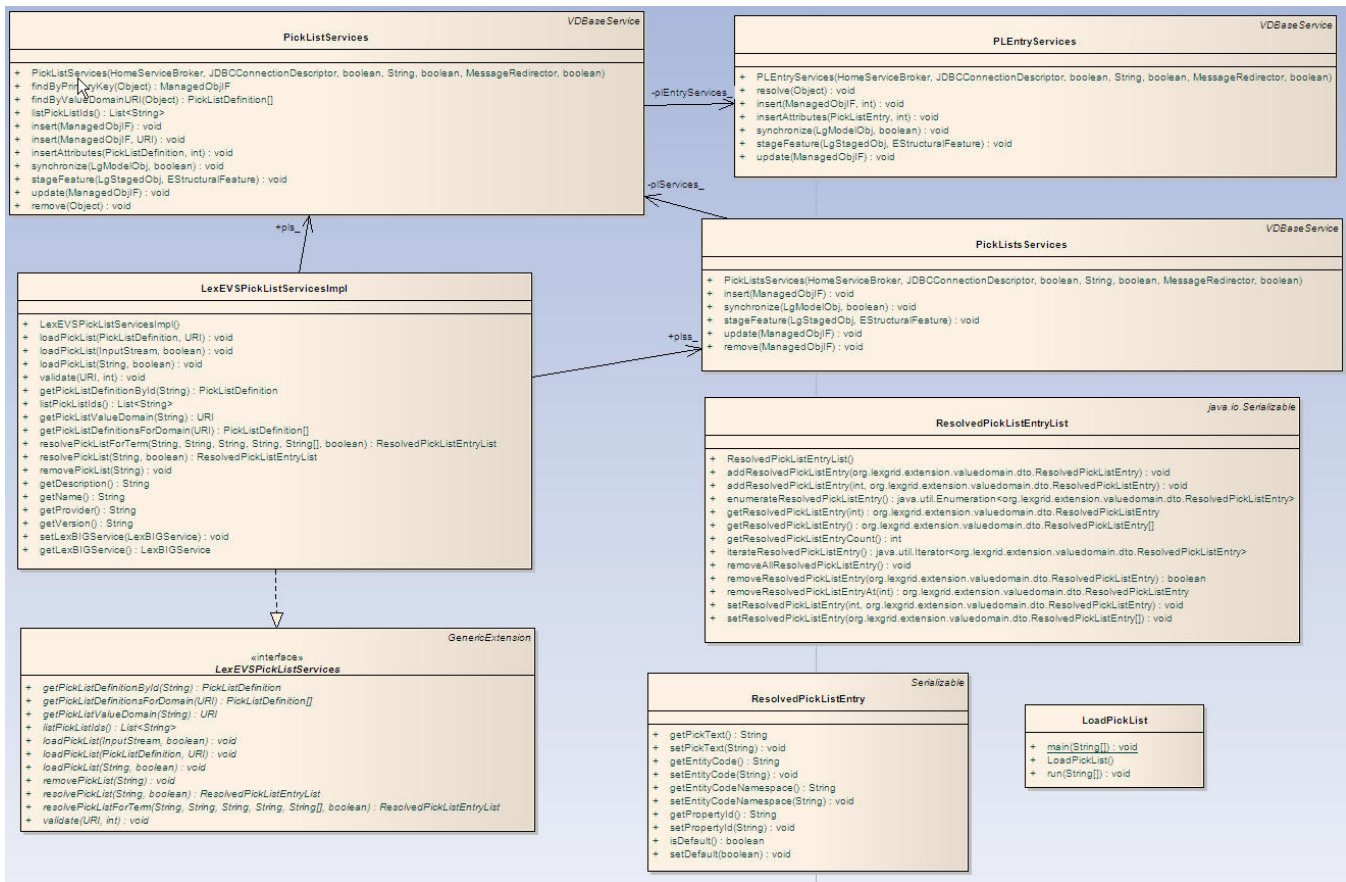
LexEVSVValueDomainServices	Primary interface for LexEVS Value Domain API
LexEVSVValueDomainServicesImpl	Implementation of LexEVSVValueDomainServices which is primary interface for LexEVS Value Domain API.
LoadValueDomain	Imports the value Domain Definitions in the source file, provided in LexGrid canonical format, to the LexBIG repository.
ResolvedValueDomainCodedNodeSet	Contains coding scheme version reference list that was used to resolve the value domain and the coded node set.
ResolvedValueDomainDefinition	A resolved Value Domain definition containing the coding scheme version reference list that was used to resolve the value domain and an iterator for resolved concepts.



Pick List Class Diagram

Classes that implements LexEVS Pick List API

Class Name	Description
PickListsServices	Class to handle list of Pick List Definitions.
PickListServices	Class to handle individual Pick List Definition objects to and fro database.
PEntryServices	Class to handle Pick List Entry objects to and fro database.
LexEVSPickListServices	Primary interface for LexEVS Pick List API.
LexEVSPickListServicesImpl	Implementation of LexEVSPickListServices which is primary interface for LexEVS Pick List API.
LoadPickList	Imports the Pick List Definitions in the source file, provided in LexGrid canonical format, to the LexBIG repository.
ResolvedPickListEntyList	Class to hold list of resolved pick list entries.
ResolvedPickListEntry	Bean for resolved pick list entries.



LexBIG Services Class Diagram

An interface to LexEVS Value Domain and Pick List Services could be obtained using an instance of LexBigService.

Method Name	Description
getValueDomainService()	Returns an interface to LexEVS Value Domain API
getPickListService()	Returns an interface to LexEVS Pick List API.

Serializable

«interface»
LexBIGService

```
+ getCodingSchemeConcepts(String, CodingSchemeVersionOrTag) : CodedNodeSet
+ getCodingSchemeConcepts(String, CodingSchemeVersionOrTag, boolean) : CodedNodeSet
+ getFilter(String) : Filter
+ getFilterExtensions() : ExtensionDescriptionList
+ getGenericExtension(String) : GenericExtension
+ getGenericExtensions() : ExtensionDescriptionList
+ getHistoryService(String) : HistoryService
+ getValueDomainService() : LexEVSVValueDomainServices
+ getPickListService() : LexEVSPickListServices
+ getLastUpdateTime() : Date
+ getMatchAlgorithms() : ModuleDescriptionList
+ getNodeGraph(String, CodingSchemeVersionOrTag, String) : CodedNodeGraph
+ getNodeSet(String, CodingSchemeVersionOrTag, LocalNameList) : CodedNodeSet
+ getServiceManager(Object) : LexBIGServiceManager
+ getServiceMetadata() : LexBIGServiceMetadata
+ getSortAlgorithm(String) : Sort
+ getSortAlgorithms(SortContext) : SortDescriptionList
+ getSupportedCodingSchemes() : CodingSchemeRenderingList
+ resolveCodingScheme(String, CodingSchemeVersionOrTag) : CodingScheme
+ resolveCodingSchemeCopyright(String, CodingSchemeVersionOrTag) : String
```

Main Service API

LexBIG API

An interface to LexEVS Value Domain and Pick List Services could be obtained using an instance of LexBigService.

Information	getValueDomainService()
Description:	Returns an interface to LexEVS Value Domain API.
Input:	<i>none</i>
Output:	<i>org.lexgrid.valuedomain.LexEVSVValueDomainServices</i>
Exception:	<i>LBException</i>
Implementation Details:	Implementation: Step 1: Call this method on the associated LexBIG Service instance. Sample Call: Using <i>LexBIGService</i> instance : org.lexgrid.valuedomain.LexEVSVValueDomainServices vds = lbs.getValueDomainService();

Information	getPickListService()
Description:	Returns an interface to LexEVS Pick List API.
Input:	<i>none</i>
Output:	<i>org.lexgrid.valuedomain.LexEVSPickListServices</i>
Exception:	<i>LBException</i>

Implementation Details:	Implementation: Step 1: Call this method on the associated LexBIG Service instance. Sample Call: Using LexBIGService instance : <pre>__org.lexgrid.valuedomain.LexEVSPickListServices vds = lbs.getPickListService();</pre>
--------------------------------	---

LexEVS Value Domain Service API - Loading Value Domain

There are three methods that could be used to load Value Domain Definitions :

Information	loadValueDomain(ValueDomainDefinition vddef, String systemReleaseURI)
Description:	Loads supplied valueDomainDefinition object
Input:	org.LexGrid.emf.valueDomains.ValueDomainDefinition, String
Output:	none
Exception:	LBException
Implementation Details:	Implementation: Step 1: Call this method on the associated LexEVS Value Domain Service instance to load a Value Domain Definition object and the System Release URI this definition belongs to. Sample Call: Step 1 : Using LexBIGService instance, get the LexEVSVValueDomainServices interface : <pre>org.lexgrid.valuedomain.LexEVSVValueDomainServices vds = lbs.getValueDomainService();</pre> Step 2 : Create and populate the ValueDomainDefinition object ValueDomainDefinition can be created using : <pre>org.LexGrid.emf.valueDomains.ValueDomainDefinition valueDomain = org.LexGrid.emf.valueDomains.ValuedomainsFactory.eINSTANCE.createValueDomainDefinition();</pre> And data for valueDomain object can be populated by using set methods : <pre>valueDomain.setValueDomainURI(uri); valueDomain.setValueDomainName(name); valueDomain.setDefaultCodingScheme(cs); valueDomain.setEntityDescription(ed);</pre> Similarly, DefinitionEntry, Property, Mapping objects can be created and assign to the valueDomain object. <pre>valueDomain.getDefinitionEntry.add(vdEntry); valueDomain.setProperties(propertisObject); valueDomain.setMappings(mappingsObject);</pre> Step 3 : call the load method by passing the Value Domain Definition object and the System Release URI : <pre>vds.loadvalueDomain(valueDomain,"Release 2009");</pre>

Information	loadValueDomain(InputStream inputStream,boolean failOnAllErrors))
Description:	Loads valueDomainDefinitions found in inputStream
Input:	java.io.InputStream boolean
Output:	none
Exception:	Exception
Implementation Details:	Implementation: Step 1: Call this method on the associated LexEVS Value Domain Service instance to load all Value Domain Definitions from the inputStream. Sample Call: Step 1 : Using LexBIGService instance, get the LexEVSVValueDomainServices interface org.lexgrid.valuedomain.LexEVSVValueDomainServices vds = lbs.getValueDomainService(); Step 2 :Call load method by passing the inputSteam and boolean flag whether to stop on load errors. : <pre>vds.loadvalueDomain(inputStream, true);</pre>

Information	loadValueDomain(String xmlFileLocation, boolean failOnAllErrors)
Description:	Loads valueDomainDefinitions found in input xml file
Input:	java.lang.String boolean
Output:	none

Exception:	<i>Exception</i>
Implementation Details:	Implementation: Step 1 : Call this method on the associated LexEVS Value Domain Service instance to load all Value Domain Definitions found in an XML file that is in LexGrid format. Sample Call: Step 1 : Using LexBIGService instance, get the LexEVSVValueDomainServices interface org.lexgrid.valuedomain.LexEVSVValueDomainServices vds = lbs.getValueDomainService(); Step 2 :Call load method by passing the inputfile location and boolean flag whether to stop on load errors. : vds.loadvalueDomain(inputFile, true);

Validate XML resources

Information	validate(URL uri, int valicationLevel) throws LBParameterException
Description:	Perform validation of the candidate resource without loading data.
Input:	java.net.URI int
Output:	none
Exception:	Org.LexGrid.LexBIG.Exceptions.LBParameterException
Implementation Details:	Implementation: Step 1: Call this method on the associated LexEVS Value Domain Service instance to validate the XML file that is in LexGrid format. This call will not load the data in XML file. Sample Call: Step 1 : Using LexBIGService instance, get the LexEVSVValueDomainServices interface org.lexgrid.valuedomain.LexEVSVValueDomainServices vds = lbs.getValueDomainService(); Step 2 :Call validate method for validation by supplying URI of the XML file and validation level: Supported validationLevels includes : 0 = Verify document is well-formed 1 = Verify document is valid vds.validaten(uriOfXMLFile, true);

Query Value Domain

Information	isConceptInDomain(String entityCode, URI valueDomainURI)
Description:	Determine if the supplied entity code is a valid result for the supplied value domain and, if it is, return the particular codingSchemeVersion that was used.
Input:	java.lang.String, java.net.URI
Output:	org.LexGrid.LexBIG.DataModel.Core. AbsoluteCodingSchemeVersionReference
Exception:	org.LexGrid.LexBIG.Exceptions.LBException
Implementation Details:	Implementation: Step 1: Call this method on the associated LexEVS Value Domain Service instance to determine if the supplied entity code is a valid in the supplied Value Domain URI. If it is, return Coding Scheme URI and the Version that was used to resolve, other wise, returns null. Sample Call: Step 1 : Using LexBIGService instance, get the LexEVSVValueDomainServices interface org.lexgrid.valuedomain.LexEVSVValueDomainServices vds = lbs.getValueDomainService(); Step 2 :Call isConceptInComdin method: AbsoluteCodingSchemeVersionReference acsvr = vds.isConceptInDomain("conceptA","valueDomainURI");
Information	isConceptInDomain(String entityCode, URI entityCodeNamespace, CodingSchemeVersionOrTag csvt, URI valueDomainURI)
Description:	Similar to previous method, this method determine if the supplied entity code and entity namespace is a valid result for the supplied value domain when resolved against supplied Coding Scheme Version or Tag.
Input:	java.lang.String, java.net.URI, org.LexGrid.LexBIG.DataModel.Core. CodingSchemeVersionOrTag java.net.URI

Output:	<code>org.LexGrid.LexBIG.DataModel.Core. AbsoluteCodingSchemeVersionReference</code>
Exception:	<code>org.LexGrid.LexBIG.Exceptions.LBException</code>
Implementation Details:	Implementation: Step 1: Call this method on the associated LexEVS Value Domain Service instance to determine if the supplied entity code and its entity namespace is a valid in the supplied Value Domain URI when resolved against supplied Coding Scheme Version or Tag. If it is, return Coding Scheme URI and the Version that was used to resolve, other wise, returns null. Sample Call: Step 1 : Using LexBIGService instance, get the LexEVSVValueDomainServices interface <code>org.lexgrid.valuedomain. LexEVSVValueDomainServices vds = lbs.getValueDomainService();</code> Step 2 :Call <code>isConceptInDomain</code> method: <code>AbsoluteCodingSchemeVersionReference acsvr = vds.isConceptInDomain("conceptA","conceptAEntityNamespace", "codingSchemeVersion","valueDomainURI");</code>

Information	<code>resolveValueDomain(Uri valueDomainUri, AbsoluteCodingSchemeVersionReferenceList csVersionList)</code>
Description:	Resolve a value domain using the supplied set of coding scheme versions.
Input:	<code>java.net.URI,</code> <code>org.LexGrid.LexBIG.DataModel.Core. AbsoluteCodingSchemeVersionReferenceList</code>
Output:	<code>org.lexgrid.valuedomain.dto.ResolvedValueDomainDefinition</code>
Exception:	<code>org.LexGrid.LexBIG.Exceptions.LBException</code>
Implementation Details:	Implementation: Step 1: Call this method on the associated LexEVS Value Domain Service instance to resolve supplied value domain against list of Coding Scheme versions if supplied. If Coding Scheme Versions list is not supplied, API will resolve against any version of the loaded Coding Scheme that is referenced by the Value Domain. Returns, object <code>ResolvedValueDomainDefinition</code> which contains <code>AbsoluteCodingSchemeVersionReferenceList</code> which tells which all Coding Scheme and the versions were used for this resolve, plus the <code>ResolvedConceptReferencesIterator</code>, which is an iterator for all the valid concepts that belongs to the Value Domain and Value Domain Definition itself. Sample Call: Step 1 : Using LexBIGService instance, get the LexEVSVValueDomainServices interface <code>org.lexgrid.valuedomain. LexEVSVValueDomainServices vds = lbs.getValueDomainService();</code> Step 2 :Create <code>AbsoluteCodingSchemeVersionReferenceList</code>: <code>AbsoluteCodingSchemeVersionReferenceList csvList = new AbsoluteCodingSchemeVersionReferenceList();</code> <code>csvList.addAbsoluteCodingSchemeVersionReference(Constructors. createAbsoluteCodingSchemeVersionReference("Automobiles", "2.0"));</code> <code>csvList.addAbsoluteCodingSchemeVersionReference(Constructors. createAbsoluteCodingSchemeVersionReference("AutomobilesParts", "2.0"));</code> Step 3 :Call <code>resolveValueDomain</code> method: <code>ResolvedValueDomainDefinition rvdDef = vds.resolveValueDomain ("valueDomainURI",csvList);</code>

Information	<code>isSubDomain(Uri childValueDomainUri, Uri parentValueDomainUri)</code>
Description:	Check whether childValueDomainUri is a child of parentValueDomainUri.
Input:	<code>java.net.URI,</code> <code>java.net.URI</code>
Output:	<code>boolean</code>
Exception:	<code>org.LexGrid.LexBIG.Exceptions.LBException</code>
Implementation Details:	Implementation: Step 1: Call this method on the associated LexEVS Value Domain Service instance to determine if all the concepts that gets resolved from supplied Child Value Domain URI is children of concepts that gets resolved from Parent Value Domain URI. Return true, if it is otherwise return false. Sample Call: Step 1 : Using LexBIGService instance, get the LexEVSVValueDomainServices interface <code>org.lexgrid.valuedomain. LexEVSVValueDomainServices vds = lbs.getValueDomainService();</code> Step 2 :Call <code>isSubDomain</code> method: <code>boolean isSubDomain = vds.isSubDomain (childValueDomainUri, parentValueDomainUri);</code>

Information	<code>getValueDomainDefinition(Uri valueDomainUri)</code>
Description:	Returns value domain definition for supplied value domain URI.
Input:	<code>java.net.URI</code>

Output:	<code>org.LexGrid.emf.valueDomains.ValueDomainDefinition</code>
Exception:	<code>org.LexGrid.LexBIG.Exceptions.LBException</code>
Implementation Details:	Implementation: Step 1: Call this method on the associated LexEVS Value Domain Service instance to get the Value Domain Definition of supplied Value Domain URI. Sample Call: Step 1 : Using LexBIGService instance, get the LexEVSVValueDomainServices interface <code>org.lexgrid.valuedomain.LexEVSVValueDomainServices</code> <code>vds = lbs.getValueDomainService();</code> Step 2 :Call <code>getValueDomainDefinition</code> method: ValueDomainDefinition <code>vdDef = vds.getValueDomainDefinition (valueDomainURI);</code>

Information	<code>listValueDomains(String valueDomainName)</code>
Description:	Return the URI's for the value domain definition(s) for the supplied domain name. If the name is null, returns everything. If the name is not null, returns the value domain(s) that have the assigned name. Note: plural because there is no guarantee of valueDomain uniqueness. If the name is the empty string "", returns all unnamed valueDomains.
Input:	<code>java.lang.String</code>
Output:	<code>java.net.URI[]</code>
Exception:	<code>org.LexGrid.LexBIG.Exceptions.LBException</code>
Implementation Details:	Implementation: Step 1: Call this method on the associated LexEVS Value Domain Service instance to get the list of Value Domain URI that matches the supplied name. Sample Call: Step 1 : Using LexBIGService instance, get the LexEVSVValueDomainServices interface <code>org.lexgrid.valuedomain.LexEVSVValueDomainServices</code> <code>vds = lbs.getValueDomainService();</code> Step 2 :Call <code>listValueDomains</code> method: URI[] <code>uris = vds.listValueDomains(" someValueDomainName");</code>

Information	<code>getAllValueDomainsWithNoNames()</code>
Description:	Return the URI's of all unnamed value domain definition(s).
Input:	<code>none</code>
Output:	<code>java.net.URI[]</code>
Exception:	<code>org.LexGrid.LexBIG.Exceptions.LBException</code>
Implementation Details:	Implementation: Step 1: Call this method on the associated LexEVS Value Domain Service instance to get the list of Value Domain URI that have no names. Sample Call: Step 1 : Using LexBIGService instance, get the LexEVSVValueDomainServices interface <code>org.lexgrid.valuedomain.LexEVSVValueDomainServices</code> <code>vds = lbs.getValueDomainService();</code> Step 2 :Call <code>getAllValueDomainsWithNoNames</code> method: URI[] <code>uris = vds.getAllValueDomainsWithNoNames();</code>

Information	<code>getValueDomainEntitiesForTerm(String term, URI valueDomainURI, String matchAlgorithm)</code>
Description:	Resolves the value domain supplied and restricts to the term and matchAlgorithm supplied. Return object ResolvedValueDomainCodedNodeSet contains the codingScheme URI and Version that was used to resolve and the CodedNodeSet. Note : the CodedNodeSet is unresolved
Input:	<code>java.lang.String,</code> <code>java.net.URI,</code> <code>java.lang.String</code>
Output:	<code>org.lexgrid.valuedomain.dto.ResolvedValueDomainCodedNodeSet</code>
Exception:	<code>org.LexGrid.LexBIG.Exceptions.LBException</code>

Implementation Details:	Implementation: Step 1: Call this method on the associated LexEVS Value Domain Service instance to get the resolved Value Domain Entries as CodedNodeSet that is restricted to supplied term and the match algorithm. Returned CodedNodeSet is not resolved. Sample Call: Step 1 : Using LexBIGService instance, get the LexEVSVValueDomainServices interface <code>org.lexgrid.valuedomain.LexEVSVValueDomainServices vds = lbs.getValueDomainService();</code> Step 2 :Call <code>getValueDomainEntriesForTerm</code> method: <code>ResolvedValueDomainCodedNodeSet vdvns = vds.getValueDomainEntriesForTerm ("General Motors", new URI("AUTO: AllDomesticANDGM"), MatchAlgorithms.exactMatch.name());</code>
--------------------------------	--

Information	getCodingSchemesInValueDomain(<i>URI valueDomainURI</i>)
Description:	Returns list of coding scheme summary that is referenced by the supplied value domain.
Input:	<code>java.net.URI</code>
Output:	<code>org.LexGrid.LexBIG.DataModel.Collections.AbsoluteCodingSchemeVersionReferenceList</code>
Exception:	<code>org.LexGrid.LexBIG.Exceptions.LBException</code>
Implementation Details:	Implementation: Step 1: Call this method on the associated LexEVS Value Domain Service instance to get List of all Coding Scheme URI and Versions the supplied Value Domain URI references. Sample Call: Step 1 : Using LexBIGService instance, get the LexEVSVValueDomainServices interface <code>org.lexgrid.valuedomain.LexEVSVValueDomainServices vds = lbs.getValueDomainService();</code> Step 2 :Call <code>getCodingSchemesInValueDomain</code> method: <code>AbsoluteCodingSchemeVersionReferenceList csvList = vds.getCodingSchemesInValueDomain(new URI("AUTO: AllDomesticANDGM"));</code>

Information	isDomain(<i>String entityCode, String codingSchemeName, CodingSchemeVersionOrTag csvt</i>)
Description:	Determine if the supplied entity code is of type valueDomain in supplied coding scheme and, if it is, return the true, otherwise return false.
Input:	<code>java.lang.String,</code> <code>java.lang.String</code> <code>org.LexGrid.LexBIG.DataModel.Core.CodingSchemeVersionOrTag</code>
Output:	<code>boolean</code>
Exception:	<code>org.LexGrid.LexBIG.Exceptions.LBException</code>
Implementation Details:	Implementation: Step 1: Call this method on the associated LexEVS Value Domain Service instance to check if supplied entity code is of type valueDomain in supplied Coding Scheme. Sample Call: Step 1 : Using LexBIGService instance, get the LexEVSVValueDomainServices interface <code>org.lexgrid.valuedomain.LexEVSVValueDomainServices vds = lbs.getValueDomainService();</code> Step 2 :Call <code>isDomain</code> method: <code>boolean isDomain = vds.isDomain ("VD005", "Automobiles", Constructors.createCodingSchemeVersionOrTag(null, "2.0"));</code>

Remove Value Domain Definition

Information	removeValueDomain(<i>URI valueDomainURI</i>)
Description:	Removes supplied value domain definition from the system.
Input:	<code>java.net.URI</code>
Output:	<code>none</code>
Exception:	<code>org.LexGrid.LexBIG.Exceptions.LBException,</code> <code>org.LexGrid.managedobj.RemoveException</code>

Implementation Details:	Implementation: Step 1 : Call this method on the associated LexEVS Value Domain Service instance to remove Value Domain Definition from the system that matches the supplied URI. Sample Call: Step 1 : Using LexBIGService instance, get the LexEVSValueDomainServices interface org.lexgrid.valuedomain. LexEVSValueDomainServices vds = lbs.getValueDomainService(); Step 2 :Call removeValueDomain method: vds.removeValueDomain (new URI("AUTO:AllDomesticANDGM"));
Information	removeAllValueDomains()
Description:	Removes all value domain definitions from the system.
Input:	none
Output:	none
Exception:	org.LexGrid.LexBIG.Exceptions.LBException, org.LexGrid.managedobj.RemoveException
Implementation Details:	Implementation: Step 1 : Call this method on the associated LexEVS Value Domain Service instance to remove all the Value Domain Definitions that are loaded in the system. Sample Call: Step 1 : Using LexBIGService instance, get the LexEVSValueDomainServices interface org.lexgrid.valuedomain. LexEVSValueDomainServices vds = lbs.getValueDomainService(); Step 2 :Call removeAllValueDomains method: vds.removeAllValueDomains();

Drop Value Domain tables

Information	dropValueDomainTables()
Description:	Drops value domain tables only if there are no value domain and pick list entries.
Input:	none
Output:	none
Exception:	org.LexGrid.LexBIG.Exceptions.LBException, org.LexGrid.managedobj.RemoveException
Implementation Details:	Implementation: Step 1 : Call this method on the associated LexEVS Value Domain Service instance to drop both Value Domain and Pick List tables. The tables will be dropped only if there are no entries in both Value Domain and Pick List tables. Sample Call: Step 1 : Using LexBIGService instance, get the LexEVSValueDomainServices interface org.lexgrid.valuedomain. LexEVSValueDomainServices vds = lbs.getValueDomainService(); Step 2 :Call dropValueDomainTables method: vds.dropValueDomainTables();

LexEVS Pick List Service API

Loading Pick List

There are three methods that could be used to load Pick List Definitions :

Information	loadPickList(PickListDefinition pldef, String systemReleaseURI)
Description:	Loads supplied Pick List Definition object
Input:	org.LexGrid.emf.valueDomains.PickListDefinition, String
Output:	none
Exception:	LBException

Implementation Details:	Implementation: Step 1: Call this method on the associated LexEVS Pick List Service instance to load Pick List Definition object and the System Release URI this definition belongs to. Sample Call: Step 1 : Using LexBIGService instance, get the LexEVSPickListServices interface : org.lexgrid.valuedomain.LexEVSPickListServices pls = lbs.getPickListService(); Step 2 : Create and populate the PickListDefinition object PickListDefinition can be created using : org.LexGrid.emf.valueDomains.PickListDefinition pickList = org.LexGrid.emf.valueDomains.ValuedomainsFactory.eINSTANCE.createPickListDefinition(); And data for pickList object can be populated by using set methods : pickList.setPickListId(pickListId); pickList.setRepresentsValueDomain(vdURI); pickList.setCompleteDomain(true); pickList.setDefaultEntityCodeNamespace(ecns); pickList.setDefaultLanguage("en"); pickList.setDefaultSortOrder("asc"); pickList.setIsActive(true); pickList.setEntityDescription(ed); Similarly, PickListEntryNode, Property, Mapping objects can be created and assign to the pickList object. pickList.getPickListEntryNode.add(pickListEntry); pickList.setProperties(propertiesObject); pickList.setMappings(mappingsObject); Step 3 : call the load method by passing the Pick List Definition object and the System Release URI : pls.loadPickList(pickList,"Release 2009");
--------------------------------	--

Information	loadPickList(InputStream inputStream, boolean failOnAllErrors)
Description:	Loads Pick List Definitions found in inputStream
Input:	java.io.InputStream boolean
Output:	none
Exception:	Exception
Implementation Details:	Implementation: Step 1: Call this method on the associated LexEVS Pick List Service instance to load all Pick List Definitions from the inputStream. Sample Call: Step 1 : Using LexBIGService instance, get the LexEVS PickList Services interface org.lexgrid.valuedomain.LexEVSPickListServices pls = lbs.getPickListService(); Step 2 :Call load method by passing the inputSteam and boolean flag whether to stop on load errors. : pls.loadPickList(inputStream, true);

Information	loadPickList (String xmlFileLocation, boolean failOnAllErrors)
Description:	Loads Pick List Definitions found in input xml file
Input:	java.lang.String boolean
Output:	none
Exception:	Exception
Implementation Details:	Implementation: Step 1: Call this method on the associated LexEVS Pick List Service instance to load all Pick List Definitions found in an XML file that is in LexGrid format. Sample Call: Step 1 : Using LexBIGService instance, get the LexEVSPickListServices interface org.lexgrid.valuedomain.LexEVSPickListServices pls = lbs.getPickListService(); Step 2 :Call load method by passing the inputfile location and boolean flag whether to stop on load errors. : pls.loadPickList(inputXMLFile, true);

Validate XML resources

Information	validate(Uri uri, int valicationLevel) throws LBParameterException
Description:	Perform validation of the candidate resource without loading data.

Input:	<code>java.net.URI</code> <code>int</code>
Output:	<code>none</code>
Exception:	<code>Org.LexGrid.LexBIG.Exceptions.LBParameterException</code>
Implementation Details:	Implementation: Step 1: Call this method on the associated LexEVS Pick List Service instance to validate the XML file that is in LexGrid format. This call will not load the data in XML file. Sample Call: Step 1 : Using LexBIGService instance, get the LexEVSPickListServices interface <code>org.lexgrid.valuedomain.LexEVSPickListServices pls = lbs.getPickListService();</code> Step 2 :Call validate method for validation by supplying URI of the XML file and validation level: Supported validationLevels includes : 0 = Verify document is well-formed 1 = Verify document is valid <code>pls.validate(uriOfXMLFile, true);</code>

Query Pick List

Information	getPickListDefinitionByld(String pickListId)
Description:	Returns pickList definition for supplied pickListId.
Input:	<code>java.lang.String</code>
Output:	<code>org.LexGrid.emf.valueDomains.PickListDefinition</code>
Exception:	<code>org.LexGrid.LexBIG.Exceptions.LBException</code>
Implementation Details:	Implementation: Step 1: Call this method on the associated LexEVS Pick List Service instance to get Pick List Definition for supplied pickListId. Sample Call: Step 1 : Using LexBIGService instance, get the LexEVSPickListServices interface <code>org.lexgrid.valuedomain.LexEVSPickListServices pls = lbs.getPickListService();</code> Step 2 :Call getPickListDefinitionByld method: <code>PickListDefinition plDef = pls.getPickListDefinitionByld("AUTO:DomesticAutoMakers");</code>

Information	getPickListDefinitionsForDomain(URI valueDomainURI)
Description:	Returns all the pickList definitions that represents supplied valueDomain URI.
Input:	<code>java.net.URI</code>
Output:	<code>org.LexGrid.emf.valueDomains.PickListDefinition[]</code>
Exception:	<code>org.LexGrid.LexBIG.Exceptions.LBException</code>
Implementation Details:	Implementation: Step 1: Call this method on the associated LexEVS Pick List Service instance to get all the Pick List Definitions that are represented by supplied Value Domain URI. Sample Call: Step 1 : Using LexBIGService instance, get the LexEVSPickListServices interface <code>org.lexgrid.valuedomain.LexEVSPickListServices pls = lbs.getPickListService();</code> Step 2 :Call getPickListDefinitionsForDomain method: <code>PickListDefinition[] plDefs = pls.getPickListDefinitionsForDomain(valueDomainURI);</code>

Information	getPickListValueDomain(String pickListId)
Description:	Returns an URI of the represented valueDomain of the pickList.
Input:	<code>java.lang.String</code>
Output:	<code>java.net.URI</code>
Exception:	<code>org.LexGrid.LexBIG.Exceptions.LBException</code>

Implementation Details:	Implementation: Step 1: Call this method on the associated LexEVS Pick List Service instance to get an Value Domain URI represented by supplied pickListId. Sample Call: Step 1 : Using LexBIGService instance, get the LexEVSPickListServices interface org.lexgrid.valuedomain. LexEVSPickListServices pls = lbs.getPickListService(); Step 2 :Call getPickListValueDomain method: URI vdURI = pls.getPickListValueDomain ("AUTO:DomesticAutoMakers");
Information	listPickListIds()
Description:	Returns a list of pickListIds that are available in the system.
Input:	none
Output:	java.util.List<java.lang.String>
Exception:	org.LexGrid.LexBIG.Exceptions.LBException
Implementation Details:	Implementation: Step 1: Call this method on the associated LexEVS Pick List Service instance to get all the PickListIds that are loaded in the system. Sample Call: Step 1 : Using LexBIGService instance, get the LexEVSPickListServices interface org.lexgrid.valuedomain. LexEVSPickListServices pls = lbs.getPickListService(); Step 2 :Call listPickListIds method: List<String> plList = pls.listPickListIds();
Information	resolvePickList(String pickListId, boolean sortByText)
Description:	Resolves pickList definition for supplied pickListId.
Input:	java.lang.String, boolean
Output:	org.lexgrid.valuedomain.dto.ResolvedPickListEntryList
Exception:	org.LexGrid.LexBIG.Exceptions.LBException
Implementation Details:	Implementation: Step 1: Call this method on the associated LexEVS Pick List Service instance to get the resolved Pick List Entries for the supplied pickListId. Optionally, if sortByTests is true, sort the pickText in the list. Sample Call: Step 1 : Using LexBIGService instance, get the LexEVSPickListServices interface org.lexgrid.valuedomain. LexEVSPickListServices pls = lbs.getPickListService(); Step 2 :Call resolvePickList method: ResolvedPickListEntryList pleList = pls.resolvePickList ("AUTO:DomesticAutoMakers", true);
Information	resolvePickListForTerm(String pickListId, String term, String matchAlgorithm, String language, String[] context, boolean sortByText)
Description:	Resolves pickList definition by applying supplied arguments.
Input:	java.lang.String, java.lang.String, java.lang.String, java.lang.String, java.lang.String[], boolean
Output:	org.lexgrid.valuedomain.dto.ResolvedPickListEntryList
Exception:	org.LexGrid.LexBIG.Exceptions.LBException

Implementation Details:	Implementation: Step 1: Call this method on the associated LexEVS Pick List Service instance to get list of Pick List Entries that matches the term supplied and meets other supplied restrictions. Sample Call: Step 1 : Using LexBIGService instance, get the LexEVSPickListServices interface org.lexgrid.valuedomain.LexEVSPickListServices pls = lbs.getPickListService(); Step 2 :Call resolvePickListForTerm method: ResolvedPickListEntryList pleList = pls.resolvePickListForTerm (" AUTO:DomesticAutoMakers","Jaguar", MatchAlgorithms.exactMatch.name(), "en", null, true);
--------------------------------	---

Remove Pick List Definition

Information	removePickList(String pickListId)
Description:	Removes supplied Pick List Definition from the system.
Input:	java.lang.String
Output:	none
Exception:	org.LexGrid.LexBIG.Exceptions.LBException, org.LexGrid.managedobj.RemoveException
Implementation Details:	Implementation: Step 1: Call this method on the associated LexEVS Pick List Service instance to remove Pick List Definition from the system that matches the supplied pickListId. Sample Call: Step 1 : Using LexBIGService instance, get the LexEVSPickListServices interface org.lexgrid.valuedomain.LexEVSPickListServices pls = lbs.getPickListService(); Step 2 :Call removePickList method: vds.removePickList ("AUTO:AllDomesticANDGM");

Resolved Value Domain Objects

ResolvedValueDomainCodedNodeSet

Contains Coding Scheme Version reference list that was used to resolve the Value Domain and the CodedNodeSet. The CodedNodeSet is not resolved.

ResolvedValueDomainDefinition

A resolved Value Domain Definition containing the Coding Scheme Version reference list that was used to resolve the Value Domain and an iterator for resolved concepts.

Resolved Pick List Objects

ResolvedPickListEntry

Contains resolved Pick List Entry Nodes

ResolvedPickListEntryList

Contains the list of resolved Pick List Entries. Also provides helpful features to add, remove, enumerate Pick List Entries.

Error Handling

Both LexEVS Value Domain and Pick List services uses org.LexGrid.LexBIG.Impl.loaders.MessageDirector to direct all fatal, error, warning, info messages with appropriate messages to the LexBIG log files in the 'log' folder of LexEVS install directory.

Along with MessageDirector, the services will also make use of org.LexGrid.LexBIG.exception.LBException to throw any fatal and error messages to the log file as well as to console.

Load Scripts

Scripts to load Value Domain and Pick List Definitions into LexEVS system will be located under 'Admin' folder of LexEVS install directory. This loader scripts will only load data in XML file that is in LexGrid format.

Value Domain Loader

LoadValueDomain.bat for Windows environment and LoadValueDomain.sh for Unix environment.
Both these scripts take in following parameters :

Parameter	Function
-in	Input <uri> URI or path specifying location of the source file.
-v	Validate <int> Perform validation of the candidate resource without loading data. Supported levels of validation includes : 0 = Verify document is well-formed 1 = Verify document is valid

Example:

```
sh LoadValueDomain.sh \-in "file:///path/to/file.xml"
```

Pick List Loader

LoadPickList.bat for Windows environment and LoadPickList.sh for Unix environment.
Both these scripts take in following parameters:

Parameter	Function
-in	Input <uri> URI or path specifying location of the source file.
-v	Validate <int> Perform validation of the candidate resource without loading data. Supported levels of validation includes : 0 = Verify document is well-formed 1 = Verify document is valid

Example:

```
sh LoadPickList.sh \-in "file:///path/to/file.xml"
```

Sample XML files

Value Domain Definitions

Below is a sample XML file containing Value Domain Definitions in LexGrid format that can be loaded using LexEVS Value Domain Service.

```
<systemRelease xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://LexGrid.org/schema/2009/01/LexGrid/versions http://LexGrid.org/schema/2009/01/
LexGrid/versions.xsd"
  xmlns="http://LexGrid.org/schema/2009/01/LexGrid/versions" xmlns:lgVer="http://LexGrid.org/schema/2009/01/
LexGrid/versions"
  xmlns:lgCommon="http://LexGrid.org/schema/2009/01/LexGrid/commonTypes" xmlns:data="data"
  xmlns:lgVD="http://LexGrid.org/schema/2009/01/LexGrid/valueDomains" xmlns:lgNaming="http://LexGrid.org
/schema/2009/01/LexGrid/naming"
  releaseURI="http://testRelease/04" releaseDate="2008-11-07T14:55:51.615-06:00">
  <lgCommon:entityDescription>Sample value domains</lgCommon:entityDescription>
  <lgVer:valueDomains>
    <lgVD:mappings>
      <lgNaming:supportedAssociation localId="hasSubtype" uri="urn:oid:1.3.6.1.4.1.2114.108.1.8.1"
>hasSubtype</lgNaming:supportedAssociation>
      <lgNaming:supportedCodingScheme localId="Automobiles" uri="urn:oid:11.11.0.1">Automobiles</lgNaming:
supportedCodingScheme>
      <lgNaming:supportedDataType localId="testhtml">test/html</lgNaming:supportedDataType>
      <lgNaming:supportedDataType localId="textplain">text/plain</lgNaming:supportedDataType>
      <lgNaming:supportedHierarchy localId="is_a" associationNames="hasSubtype" isForwardNavigable="true"
rootCode="@">hasSubtype</lgNaming:supportedHierarchy>
      <lgNaming:supportedLanguage localId="en" uri="www.en.org/orsomething">en</lgNaming:
supportedLanguage>
      <lgNaming:supportedNamespace localId="Automobiles" uri="urn:oid:11.11.0.1" equivalentCodingScheme="
Automobiles">Automobiles</lgNaming:supportedNamespace>
      <lgNaming:supportedProperty localId="textualPresentation">textualPresentation</lgNaming:
supportedProperty>
```

```

        <lgNaming:supportedSource localId="lexgrid.org">lexgrid.org</lgNaming:supportedSource>
        <lgNaming:supportedSource localId="_111101">11.11.0.1</lgNaming:supportedSource>
    </lgVD:mappings>
    <lgVD:valueDomainDefinition valueDomainURI="SRITEST:AUTO:DomesticAutoMakers" valueDomainName="Domestic
Auto Makers" defaultCodingScheme="Automobiles" effectiveDate="2009-01-01T11:00:00Z" isActive="true" status="
ACTIVE">
        <lgVD:properties>
            <lgCommon:property propertyName="textualPresentation">
                <lgCommon:value> Domestic Auto Makers</lgCommon:value>
            </lgCommon:property>
        </lgVD:properties>
        <lgVD:definitionEntry ruleOrder="1" operator="OR">
            <lgVD:entityReference entityCode="005" referenceAssociation="hasSubtype" transitiveClosure="
true" targetToSource="false" leafOnly="false"/>
        </lgVD:definitionEntry>
    </lgVD:valueDomainDefinition>
    <lgVD:valueDomainDefinition valueDomainURI="SRITEST:AUTO:AllDomesticButGM" valueDomainName="All Domestic
Autos But GM" defaultCodingScheme="Automobiles" effectiveDate="2009-01-01T11:00:00Z" isActive="true" status="
ACTIVE">
        <lgVD:properties>
            <lgCommon:property propertyName="textualPresentation">
                <lgCommon:value> Domestic Auto Makers</lgCommon:value>
            </lgCommon:property>
        </lgVD:properties>
        <lgVD:definitionEntry ruleOrder="1" operator="OR">
            <lgVD:entityReference entityCode="005" referenceAssociation="hasSubtype" transitiveClosure="
true" targetToSource="false" leafOnly="false"/>
        </lgVD:definitionEntry>
        <lgVD:definitionEntry ruleOrder="2" operator="SUBTRACT">
            <lgVD:entityReference entityCode="GM" referenceAssociation="hasSubtype" transitiveClosure="
true" targetToSource="false" leafOnly="false"/>
        </lgVD:definitionEntry>
    </lgVD:valueDomainDefinition>
    <lgVD:valueDomainDefinition valueDomainURI="SRITEST:AUTO:AllDomesticANDGM" valueDomainName="All
Domestic Autos AND GM" defaultCodingScheme="Automobiles" effectiveDate="2009-01-01T11:00:00Z" isActive="true"
status="ACTIVE">
        <lgVD:properties>
            <lgCommon:property propertyName="textualPresentation">
                <lgCommon:value> Domestic Auto Makers AND GM</lgCommon:value>
            </lgCommon:property>
        </lgVD:properties>
        <lgVD:definitionEntry ruleOrder="1" operator="OR">
            <lgVD:entityReference entityCode="005" referenceAssociation="hasSubtype" transitiveClosure="
true" targetToSource="false" leafOnly="false"/>
        </lgVD:definitionEntry>
        <lgVD:definitionEntry ruleOrder="2" operator="AND">
            <lgVD:entityReference entityCode="GM" referenceAssociation="hasSubtype" transitiveClosure="
true" targetToSource="false" leafOnly="false"/>
        </lgVD:definitionEntry>
    </lgVD:valueDomainDefinition>
    <lgVD:valueDomainDefinition valueDomainURI="SRITEST:AUTO:DomesticLeafOnly" valueDomainName="Domestic
Leaf Only" defaultCodingScheme="Automobiles" effectiveDate="2009-01-01T11:00:00Z" isActive="true" status="
ACTIVE">
        <lgVD:properties>
            <lgCommon:property propertyName="textualPresentation">
                <lgCommon:value>Domestic Leaf Only</lgCommon:value>
            </lgCommon:property>
        </lgVD:properties>
        <lgVD:definitionEntry ruleOrder="1" operator="OR">
            <lgVD:entityReference entityCode="005" referenceAssociation="hasSubtype" transitiveClosure="
true" targetToSource="false" leafOnly="true"/>
        </lgVD:definitionEntry>
    </lgVD:valueDomainDefinition>
</lgVer:valueDomains>
</systemRelease>

```

Pick List Definitions

Below is a sample XML file containing Pick List Definitions in LexGrid format that can be loaded using LexEVS Pick List Service.

```

<systemRelease xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://LexGrid.org/schema/2009/01/LexGrid/versions http://LexGrid.org/schema/2009/01/
LexGrid/versions.xsd"
  xmlns="http://LexGrid.org/schema/2009/01/LexGrid/versions" xmlns:lgVer="http://LexGrid.org/schema/2009/01/
LexGrid/versions"
  xmlns:lgCommon="http://LexGrid.org/schema/2009/01/LexGrid/commonTypes" xmlns:data="data"
  xmlns:lgVD="http://LexGrid.org/schema/2009/01/LexGrid/valueDomains" xmlns:lgNaming="http://LexGrid.org
/schema/2009/01/LexGrid/naming"
  releaseURI="http://testRelease/04" releaseDate="2008-11-07T14:55:51.615-06:00">
  <lgCommon:entityDescription>Sample value domains</lgCommon:entityDescription>
  <pickLists>
    <lgVD:mappings>
      <lgNaming:supportedCodingScheme localId="Automobiles" uri="urn:oid:11.11.0.1">Automobiles</lgNaming:
supportedCodingScheme>
      <lgNaming:supportedLanguage localId="en" uri="www.en.org/orsomething">en</lgNaming:
supportedLanguage>
      <lgNaming:supportedNamespace localId="Automobiles" uri="urn:oid:11.11.0.1" equivalentCodingScheme="
Automobiles">Automobiles</lgNaming:supportedNamespace>
      <lgNaming:supportedProperty localId="textualPresentation">textualPresentation</lgNaming:
supportedProperty>
      <lgNaming:supportedSource localId="lexgrid.org">lexgrid.org</lgNaming:supportedSource>
      <lgNaming:supportedSource localId="_111101">11.11.0.1</lgNaming:supportedSource>
    </lgVD:mappings>
    <lgVD:pickListDefinition pickListId="SRITEST:AUTO:DomesticAutoMakers" representsValueDomain="SRITEST:
AUTO:DomesticAutoMakers" isActive="true" defaultEntityCodeNamespace="Automobiles" defaultLanguage="en"
completedDomain="false">
      <lgCommon:owner>Owner for Domestic Auto Makers</lgCommon:owner>
      <lgCommon:entityDescription>DomesticAutoMakers</lgCommon:entityDescription>
      <lgVD:mappings>
        <lgNaming:supportedCodingScheme localId="Automobiles" uri="urn:oid:11.11.0.1">Automobiles<
/ lgNaming:supportedCodingScheme>
        <lgNaming:supportedDataType localId="texthtml">text/html</lgNaming:supportedDataType>
        <lgNaming:supportedDataType localId="textplain">text/plain</lgNaming:supportedDataType>
        <lgNaming:supportedLanguage localId="en" uri="www.en.org/orsomething">en</lgNaming:
supportedLanguage>
        <lgNaming:supportedNamespace localId="Automobiles" uri="urn:oid:11.11.0.1"
equivalentCodingScheme="Automobiles">Automobiles</lgNaming:supportedNamespace>
        <lgNaming:supportedProperty localId="textualPresentation">textualPresentation</lgNaming:
supportedProperty>
        <lgNaming:supportedSource assemblyRule="rule1" uri="http://informatics.mayo.edu" localId="
lexgrid.org">lexgrid.org</lgNaming:supportedSource>
        <lgNaming:supportedSource localId="_111101">11.11.0.1</lgNaming:supportedSource>
      </lgVD:mappings>
      <lgVD:pickListEntryNode pickListEntryId="PLGmp1" isActive="true">
        <lgCommon:owner>Owner for PLGmp1</lgCommon:owner>
        <lgCommon:entryState containingRevision="R001" relativeOrder="1" changeType="NEW" prevRevision="
R00A"/>
        <lgVD:inclusionEntry entityCode="GM" entityCodeNamespace="Automobiles" propertyId="p1">
          <lgVD:pickText>General Motors</lgVD:pickText>
        </lgVD:inclusionEntry>
        <lgVD:properties>
          <lgCommon:property propertyName="textualPresentation" isActive="true" language="en"
propertyId="p1" propertyType="presentation" status="active" effectiveDate="2001-12-17T09:30:47Z"
expirationDate="2011-12-17T09:30:47Z">
            <lgCommon:owner role="role" subRef="subref">General Motors</lgCommon:owner>
            <lgCommon:entryState containingRevision="R001" relativeOrder="1" changeType="NEW"
prevRevision="R00A"/>
            <lgCommon:source subRef="subref1" role="role1">General Motors</lgCommon:source>
            <lgCommon:value dataType="textplain">Property for General Motors</lgCommon:value>
          </lgCommon:property>
        </lgVD:properties>
      </lgVD:pickListEntryNode>
      <lgVD:pickListEntryNode pickListEntryId="PLGmp2" isActive="true">
        <lgCommon:owner>Owner for PLGmp2</lgCommon:owner>
        <lgCommon:entryState containingRevision="R001" relativeOrder="1" changeType="NEW" prevRevision="
R00A"/>
        <lgVD:inclusionEntry entityCode="GM" entityCodeNamespace="Automobiles" propertyId="p2">
          <lgVD:pickText>GM</lgVD:pickText>
        </lgVD:inclusionEntry>
      </lgVD:pickListEntryNode>
      <lgVD:pickListEntryNode pickListEntryId="PLJaguarp1" isActive="true">

```

```

        <lgCommon:owner>Owner for PLJaguarp1</lgCommon:owner>
        <lgCommon:entryState containingRevision="R001" relativeOrder="1" changeType="NEW" prevRevision="
R00A"/>
        <lgVD:inclusionEntry entityCode="Jaguar" entityCodeNamespace="Automobiles" propertyId="p1">
            <lgVD:pickText>Jaguar</lgVD:pickText>
        </lgVD:inclusionEntry>
    </lgVD:pickListEntryNode>
    <lgVD:pickListEntryNode pickListEntryId="PLChevroletp1" isActive="true">
        <lgCommon:owner>Owner for PLChevroletp1</lgCommon:owner>
        <lgCommon:entryState containingRevision="R001" relativeOrder="1" changeType="NEW" prevRevision="
R00A"/>
        <lgVD:inclusionEntry entityCode="Chevy" entityCodeNamespace="Automobiles" propertyId="p1">
            <lgVD:pickText>Chevrolet</lgVD:pickText>
        </lgVD:inclusionEntry>
    </lgVD:pickListEntryNode>
    </lgVD:pickListDefinition>
    <lgVD:pickListDefinition pickListId="SRITEST:AUTO:DomesticLeafOnly" representsValueDomain="SRITEST:AUTO:
DomesticLeafOnly" completeDomain="true" defaultEntityCodeNamespace="Automobiles" defaultLanguage="en" isActive="
true">
        <lgCommon:entityDescription>Leaf Only Nodes of Domestic AutoMakers</lgCommon:entityDescription>
    </lgVD:pickListDefinition>
</pickLists>
</systemRelease>

```

Database structure

Value Domain Tables

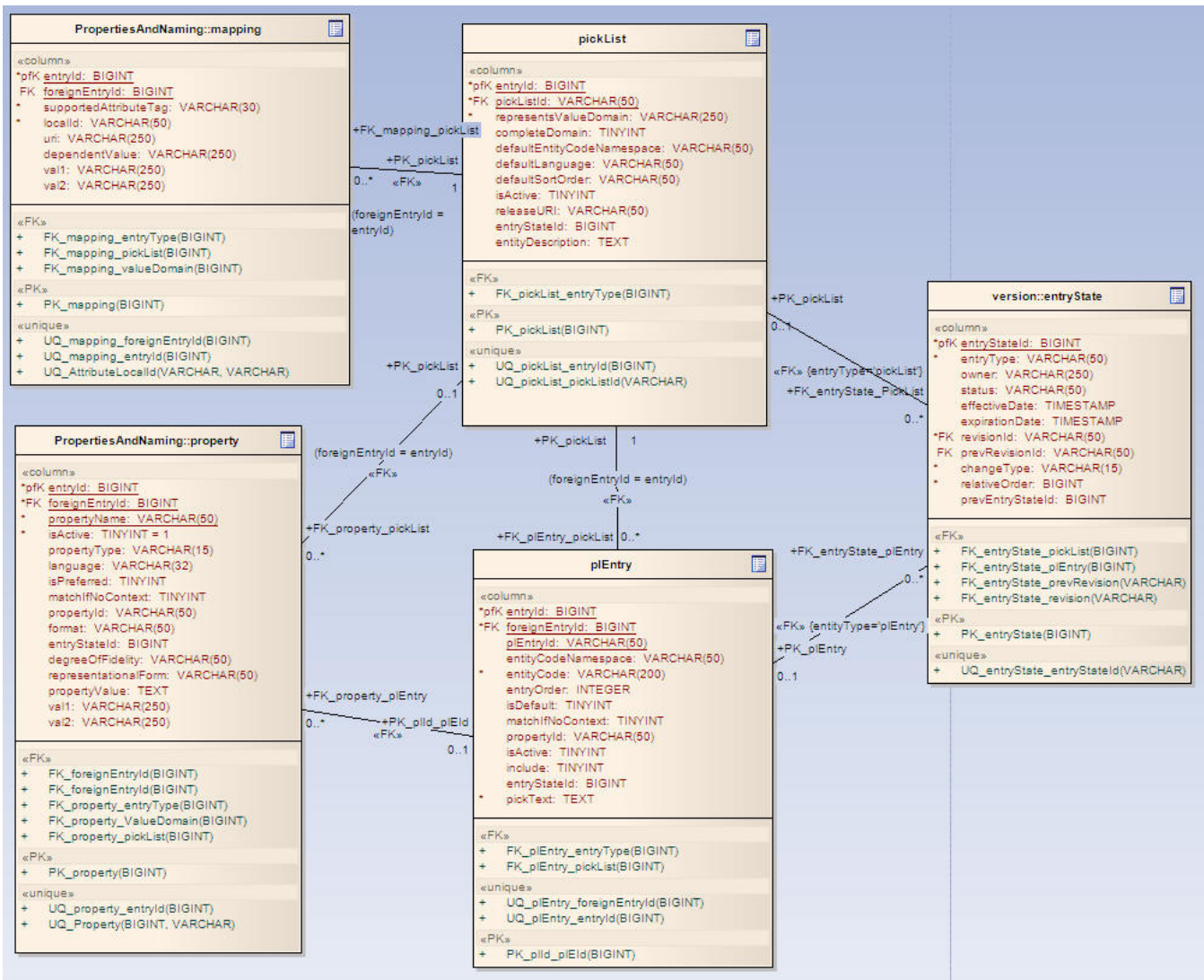


Table Name	Description
pickList	Will contain Pick List Definition information
plEntry	Contains Pick List Entry Nodes information
entryState	Contains entry state details of every entry
mappings	Contains supported mapping information for a Pick List Definition
property	Contains Property informations for Pick List Definition and its Nodes

Installation / Packaging

Both LexEVS Value Domain and Pick List services are integrated part of core LexEVS API and will be packaged and installed with other LexEVS services.

System Testing

Value Domain Service

The System test case for the LexEVS Value Domain service is performed using the JUnit test suite:

org.LexGrid.LexBIG.Impl.testUtility.VDAITests

This test suite will be run as part of regular LexEVS test suites AllTestsAllConfigs and AllTestsNormalConfigs.

Pick List Service

The System test case for the LexEVS Value Domain service is performed using the JUnit test suite:

org.LexGrid.LexBIG.Impl.testUtility.PickListAllTests

This test suite will be run as part of regular LexEVS test suites AllTestsAllConfigs and AllTestsNormalConfigs.

Improved Loader Framework Detailed Design

Document Purpose

This document provides the detailed design and implementation of LexBIG Enterprise Vocabulary Service (LexEVS) Loader Framework Extension. It is also the goal of this document to provide enough information to allow those persons wishing to create their own loaders can do so. This document will also assume the reader is already familiar with the LexEVS software.

Implementation Overview

Description

The LexEVS software already provides a set of loaders within an existing legacy framework which served LexEVS developers well over many years. But as LexEVS has gained users, and requests for new loaders has grown, it was decided that a new loader framework should be developed that would: (1) be easier to extend (2) provide improved performance (3) dynamic loading of new loaders (4) take advantage of proven open source components such as Spring Batch and Hibernate.

Specifically, this development work addresses "TASK 6 - IMPROVE LEXEVS LOADING FRAMEWORK" in the National Cancer Institute (NCI) Statement of Work (SOW) document (reference ?????).

Also, this Framework is completely independent of the current loader code so there is no impact to current loaders.

Scope

The LexEVS Loader Framework will provide a way for LexEVS developers to write new loaders and have them recognized dynamically by the LexEVS code. Also the framework will provide help to loader developers in the form of utility classes and interfaces.

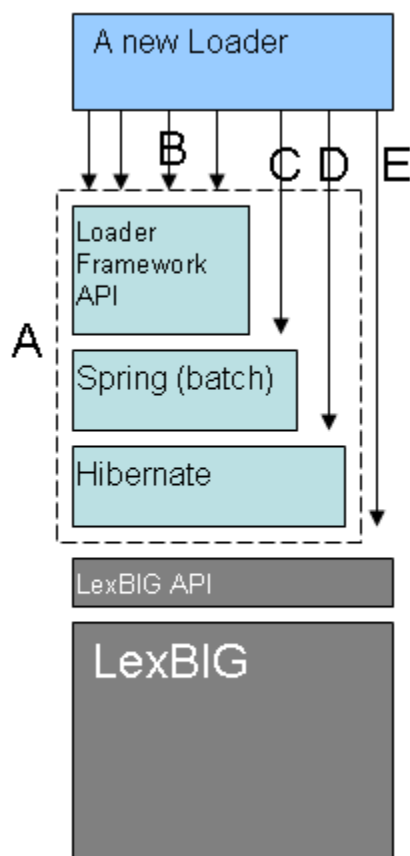
Architecture

The LexEVS Loaders Framework extend the functionality of LexBIG 5.0. For more information on LexBIG, refer to [LexEVS 5.0](#).

High Level Overview

The following figure shows the major components of the Loader Framework (A) in relation to a hypothetical new loader and what expected API usage would be. Ideally, the new loader can find make most if its API calls through the utilities provided by the Loader Framework API (B). Some work will need to be done with Spring (C) such as configuration of a Spring config file. Also it may or may not be necessary for a loader to use Hibernate (D) or use the LexBIG API (E). However, again, the hope is that much of the work a new loader may need to do can be accomplished by the Loader Framework API.

The Loader Framework utilizes Spring Batch for managing its Java objects to improve performance and Hibernate provides the mapping to the LexGrid database.



Assumptions

- None

Dependencies

- This Loader Framework requires LexEVS release 5.0 or above.
- Development system are required to install the Sun Java Development Kit (SDK) or Java Runtime Environment (JRE) version 1.5.0_11 or above.
- Maven 2.1 or greater.
- For software and hardware dependencies for the system hosting the LexEVS runtime, refer to the [LexEVS 5.0 Documentation](#).

Issues

- None

Third Party Tools

- Spring: A lightweight open-source application framework.
 - Spring see <http://www.springsource.com/>
 - Spring Batch see <http://static.springsource.org/spring-batch/>
 - Sprint Batch Reference see <http://static.springsource.org/spring-batch/reference/html/index.html>
- Hibernate: An open source Java persistence framework. See <https://www.hibernate.org/>
- Maven: Apache build manager for Java projects. See <http://maven.apache.org/>
- Eclipse: An Open Source IDE. See <http://www.eclipse.org/>

Implementation Contents

Development and Build Environment

The Loader Framework code is available in the NCI Subversion (SVN) repository. It is comprised of three Framework projects. Also at the time of this writing there are three projects in the repository that utilize the Loader Framework. These projects utilize Maven for build and dependency management.

Loader Framework Projects

- PersistenceLayer: is a Hibernate connector to the LexBIG database.
- Loader-framework: sets up build information for Maven.

- Loader-framework-core: contains all the interfaces and utilities. Also contains an extendable class "AbstractSpringBatchLoader" that all new Loaders should extend.

Loader Projects Using the New Framework

- abstract-rrf-loader: is a holder for common rrf-based loader code
- meta-loader: new loader to read the NCI MetaThesaurus
- umls-loader: a loader for reading Unified Medical Language System (UMLS) content.

Maven

The above projects are built and managed by Maven.
Maven plugin for Eclipse: <http://m2eclipse.codehaus.org/>

How to Use the Loader Framework: A Roadmap

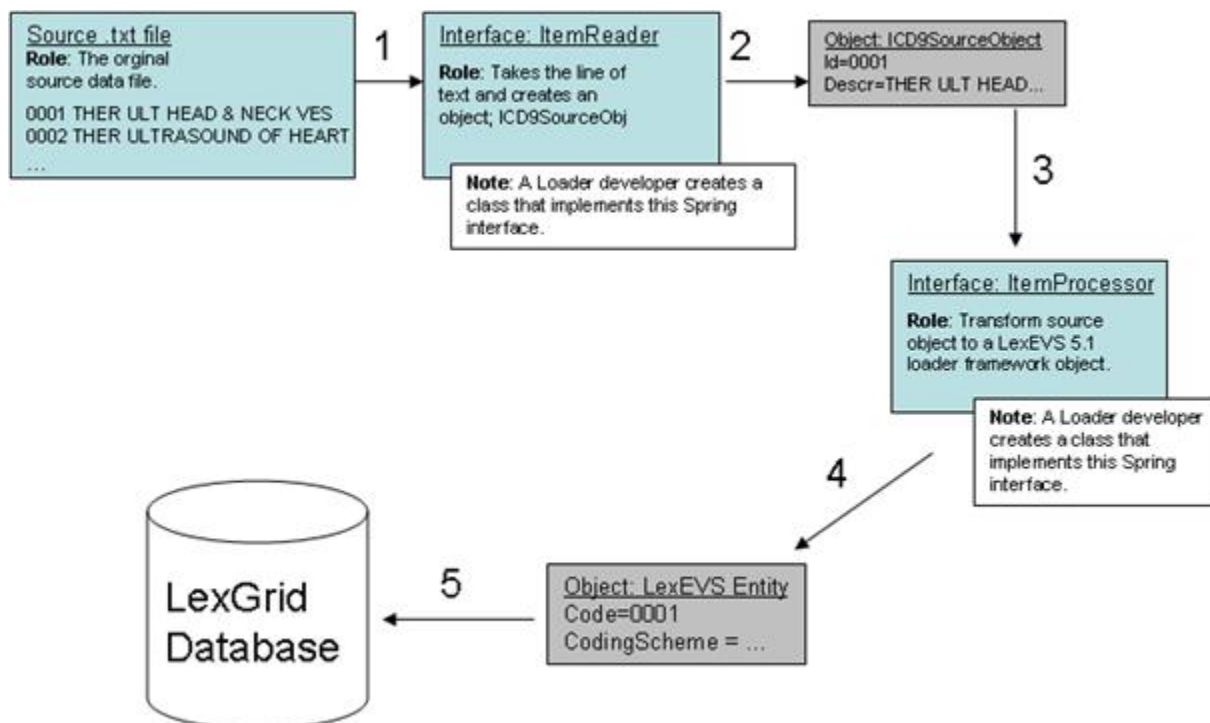
So you want to write a loader and use the Loader Framework. What are the key considerations?
In general the process can be described as:

1. Reading the raw data from the file into intermediate data structures such as a user defined ICD9SourceObject object.
2. Process the user defined objects into LexGrid model object.
3. Write the data in the LexGrid objects to the database.

An example may help in understanding the Framework. Our discussion will refer to the following figure. Lets say we are writing a loader to load the ICD-9-CM codes and their description which are contained in a text file. We know we'll need a data structure to hold the data after we've read it so we have a class:

```
ICD9SourceObject {
String id;
String descr;
String getId() { return id; }
}
```

Enter Spring. The Loader Framework uses Spring Batch to manage the reading, processing and writing of data. Spring provides classes and interfaces to help do this work and the Loader Framework also provides utilities to help loader developers. In our example, we will write a class that will use the Spring ItemReader interface. It will take a line of text and return an ICD9SourceObject (1 and 2). Next we'll want to process that data into a LexEVS object such as an Entity object. So we'll write class that implements Spring's ItemProcessor interface. It will take our ICD9SourceObject and output a LexEVS Entity object (3,4). Finally, we'll want to write the data to the database (5). Note that the LexEVS model objects provided in the Loader Framework are generated by Hibernate and utilize Hibernate to write the data to the database. This will free us from having to write SQL.



Spring

You will need to configure Spring to be aware of your objects and how to manage them. This is done via a XML configuration file. More details on the Spring config file below.

ItemReader/ItemProcessor

You will either write a class implementing this interface or use one of the Spring helper classes that already implement this interface. If you use one of the Spring classes you may need to provide one of your own helper classes to construct your internal data structure object, such as ICD9SourceObject. You would provide it to the Spring object via a setProperty call configured in the Spring config file.

Maven Set up

The projects containing the Loader Framework (**PersistenceLayer** , **loader-framework** , and **loader-framework-core**) use Maven for dependency management and build. You will still use Eclipse as your IDE and code repository but you will need to install a Maven plugin for Eclipse which can be found at: <http://m2eclipse.sonatype.org/>

After the plugin is installed you'll need to provide a URL and userid/password to a Maven repository on a server (which manages your dependencies or dependent jar files). Ours here at Mayo is:
<http://bmidev4:8282/nexus-webapp-1.3.3/index.html>

Once Maven is configured you can import the Loader Framework classes from SVN. Upon doing that you will most likely see build errors about missing jars. Resolve those by right clicking on the project with errors, select '**Maven**', and '**Resolve Dependencies**'. This will pull the dependant jars from the Maven repository into your local environment.

To build a Maven project, right click on the project, select '**Maven**', then select '**assembly:assembly**'.

Eclipse Project Set up

When loader developers create a new loader project in Eclipse it is recommended they follow the Maven directory structure. By following this convention Maven can build the project and find the test cases.

From the Maven documentation:

Under this directory you will notice the following **standard project structure**.

```
my-app
|-- pom.xml
'-- src
    |-- main
    |   |-- java
    |       |-- com
    |           |-- mycompany
    |               |-- app
    |                   |-- App.java
    |-- test
    |   |-- java
    |       |-- com
    |           |-- mycompany
    |               |-- app
    |                   |-- AppTest.java
```

The src/main/java directory contains the project source code, the src/test/java directory contains the test source, and the pom.xml is the project's Project Object Model, or POM.

For more information on the Maven project refer to the [documentation on apache.org](http://maven.apache.org/).

Configure your Spring Config (myLoader.xml)

Spring is a lightweight bean management container and among other things it contains a batch function which is utilized by the Loader Framework. A loader using the framework will need to work closely with Spring Batch and the way it does that is through Spring's configuration file where you configure beans (your loader code) and how the loader code should be utilized by Spring Batch (by configuring a Job, Step and other Spring Batch stuff in the spring config file). What follows is a brief overview of those tags related to the LoaderFramework. For more detail refer to the [Spring documentation](http://spring.io/docs/core/3.1.0.RELEASE/spring-batch/).

```

<job id="ioSampleJob">
  <step name="step1">
    <tasklet>
      <chunk reader="fooReader" processor="fooProcessor" writer="compositeItemWriter" commit-interval="100">
      </chunk>
    </tasklet>
  </step>
</job>

<bean id="compositeItemWriter" class="...CompositeItemWriter">
  <property name="delegate" ref="barWriter" />
</bean>

<bean id="barWriter" class="...BarWriter" />

```

Beans

The 'beans:beans' tag is the all-encompassing tag. You define all your other tags in here. You can also define an import within this tag to import an external Spring config file. Not shown in figure 3.

Bean

Use these tags, 'beans:bean', to define the beans to be managed by the Spring container by specifying the packaged qualified class name. You can also specify initialization values and set bean properties within these tags.

```

<beans:bean id="umlsCuiPropertyProcessor" parent="umlsDefaultPropertyProcessor" class="org.lexgrid.loader.
processor.EntityPropertyProcessor">
  <beans:property name="propertyResolver" ref="umlsCuiPropertyResolver" />
</beans:bean>

```

Job

The 'job' tag is the main unit of work. The job is comprised of one or more steps that define the work to be done. Other advanced and interesting things can be done within the Job such as using 'split' and 'flow' tags to indicate work that can be done in parallel steps to improve performance.

```

<job id="umlsJob" restartable="true">
  <step id="populateStagingTable" next="loadHardcodedValues" parent="stagingTablePopulatorStepFactory"/>
  ...

```

Step

One or more step tags make up a job and can vary from simple to complex in content. Among other things, you can specify which step should be executed next.

Tasklet

You can do anything you want within a Tasklet such as sending an email or a LexBIG function such as indexing. You're not limited to just database operations. The Spring documentation also has this to say about Tasklets:

 The Tasklet is a simple interface that has one method, execute, which will be called repeatedly by the TaskletStep until it either returns RepeatStatus.FINISHED or throws an exception to signal a failure. Each call to the Tasklet is wrapped in a transaction

Chunk

Spring documentation says it best:

 Spring Batch uses a 'Chunk Oriented' processing style within its most common implementation. Chunk oriented processing refers to reading the data one at a time, and creating 'chunks' that will be written out, within a transaction boundary. One item is read in from an ItemReader, handed to an ItemWriter, and aggregated. Once the number of items read equals the commit interval, the entire chunk is written out via the ItemWriter, and then the transaction is committed.

Reader

An attribute of the chunk tag. Here is the class that you defined implementing the Spring ItemReader interface to read data from your data file and create domain-specific objects.

Processor

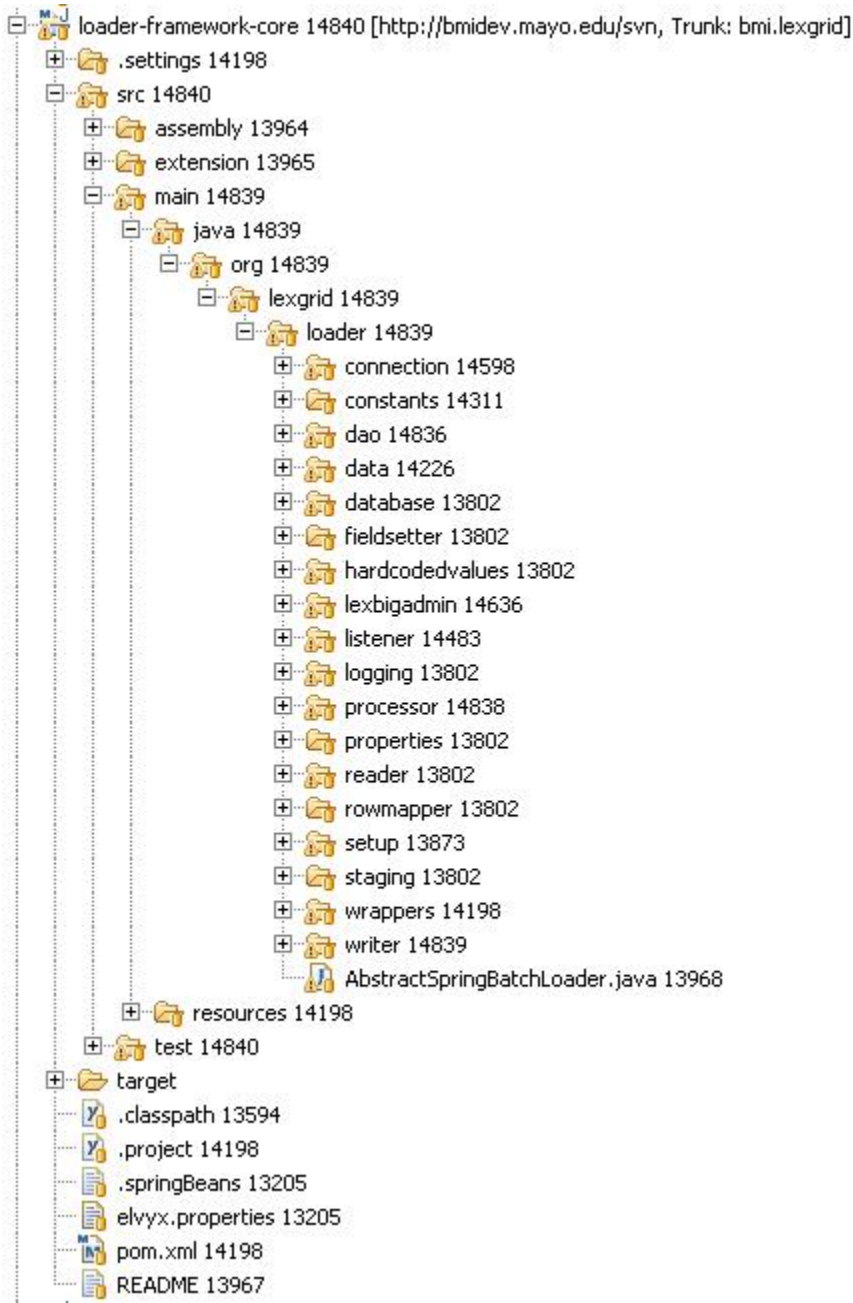
Another attribute of the chunk tag. This is the class that implements the ItemProcessor interface where other manipulations of the domain objects take place. In the case of the Loader Framework we create LexGrid model objects from the domain objects so that they can be written to the database via Hibernate. Note that this is not a required attribute. In theory if you had a data source you could read from such that you could create LexBIG object immediately you would not have need of a processor. In practice this is most likely not be the case. You will need to work with the data to get it into LexBIG objects.

Writer

Attribute of the chunk tag. This class will implement the Spring interface ItemWriter. In the case of the Loader Framework these classes have been written for you. They are the LexGrid model objects that use Hibernate to write to the database.

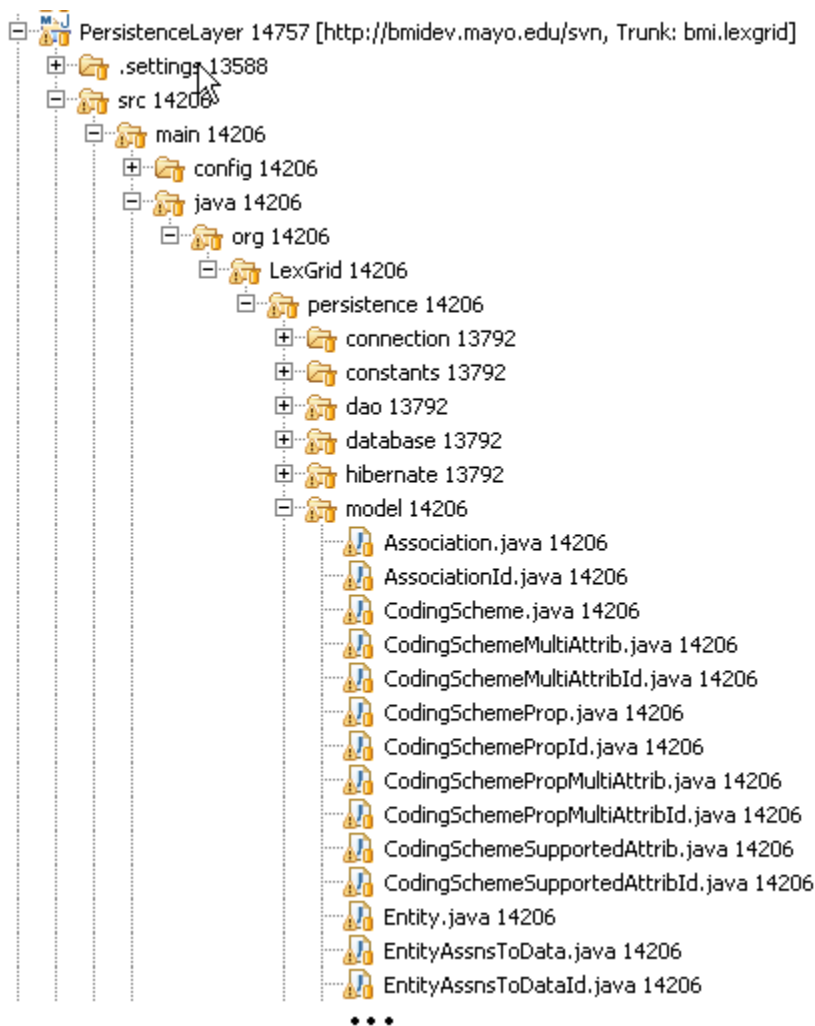
Key Directories

Below is an image of the loader-framework-core project in Eclipse which shows the key directories of the Loader Framework. The following is a summary of the contents of those directories.



Directory	Summary
connection	Connect to LexBIG and do LexBIG tasks such as register and activate.

constants	Assorted constants.
dao	Access to the LexBIG database.
data	Directly related to data going into the LexBIG database tables.
database	Database specific tasks not related to data, such as finding out the database type (MySQL, Oracle)
fieldsetter	Spring related classes for helping to write to the database.
lexbigadmin	Common tasks you want LexBIG to do for you such as indexing.
listener	You can attach listeners to a load so that the code will execute and certain points in the load such as a cleanup listener that runs when the load is finished or a setup listener etc...
logging	Gives you access to the LexBIG logger.
processor	Important directory. Contains classes that you can pass your domain specific object to and will return a LexBIG object.
properties	Code used internally by the Loader Framework.
reader	Readers and reader-related tools for loader developers.
rowmapper	Classes for reading from a database. Currently experimental code.
setup	Classes such as JobRepositoryManager that help Spring do its work. As Spring hums along it keeps tables of its internal workings. Loader developers should not need to dive into this directory.
staging	If your loader needs to load data to the database temporarily you can find helper classes in this directory.
wrappers	Helper classes and data structures such as a Code/CodingScheme class.
writer	Miscellaneous classes that write to the database. These are not the same ones you'd use in your loader, i.e the LexBIG model objects that use Hibernate. Those classes are contained in the PersistenceLayer project (next figure). It is by using those classes in the PersistenceLayer that you let the Loader Framework do some of the heavy lifting for you.



Algorithms

None

Batch Processes

None

Error Handling

Spring Batch gives the Loader Framework some degree of recovery from errors. Like the other features of Spring it is something the Loader developer would need to configure in their Spring config file. Basically, Spring will keep track of the steps it has executed and make note of any step that has failed. Those failed steps can be re-run at a later time. The Spring documentation provides additional information on this function. Refer to the [configure job documentation](#) and [configure step documentation](#).

Database Changes

None

Client

Currently, the LexBIG GUI does not provide a framework to dynamically load extendable GUI components. While not impossible to extend the GUI functionality especially for those working closely with the LexBIG code, loaders written to use the new framework should expect that their loader will be called via the command line or script.

JSP/HTML

None

Servlet

None.

Security Issues

None.

Performance

Spring can accommodate parallel processing to enhance performance. The Spring documentation provides a good discussion of this topic. Refer to [scalability on springsource.org](#).

Internationalization

Not internationalized.

Installation / Packaging

The Loader Framework is packaged as a LexBIG extension and thus is not included in the LexBIG jar.

Migration

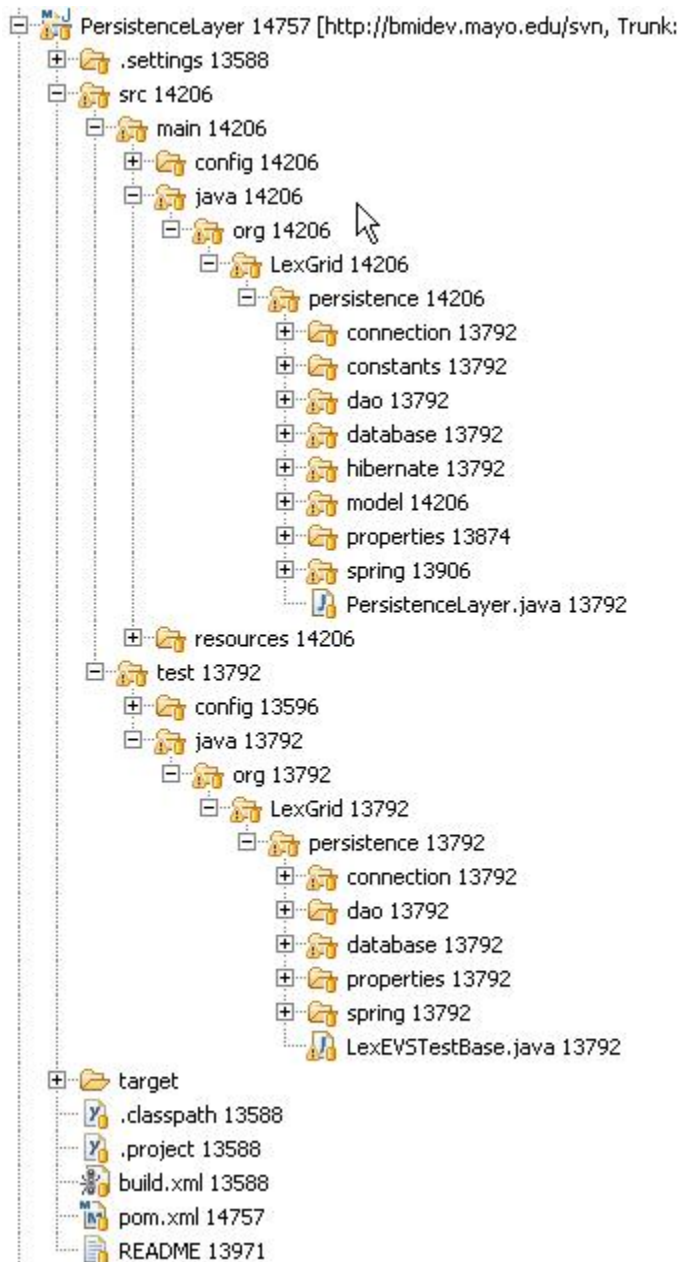
None.

Documentation Considerations

The Loader Framework will also be described in the Knowledge Center.

Testing

Automated tests are run via Maven. As mentioned earlier the projects containing the Loader Framework code are configured to work with Maven. The following figure shows the PersistenceLayer project and its standard Maven layout. Notice the structure of the test code mirrors the structure of the application code. To run the automated test in our Eclipse environment we select the project, right click, select 'Run As' and select 'Maven test'. Maven will do the rest.



Test Guidelines

The test cases are also integrated into the LexBIG 5.1 build environment and are run with each build.

Test Cases

See System Testing.

Test Results

See System Testing.

Custom Loader Feasibility Report and Recommendation

Persistence Layer Feasibility

The Persistence Layer enables LexEVS to have a single access point to the underlying database. This has several advantages:

1. The DAO is implemented as an Interface, not a concrete class. We can implement this interface with Hibernate, JDBC, Ibatis, or any other Persistence tool or framework.
2. All loaders can now share a single entry point to the database, and are not limited by memory constraints as some of the EMF persistence was.

3. Connection Pooling and management is abstracted from the code and pluggable. Data source implementations may be switched and Connection Pooling may be configured without recompiling code.
4. Transactions may be defined programmatically via AOP interceptors.

As LexEVS moves forward, the Persistence Layer is also flexible enough to play a part in the runtime Query API. With this, the runtime and loader code would be able to share a common Data Access Layer - we would then have a true DAO Layer.

Loader Framework Feasibility

The Loader Framework has been implemented for two loaders, the UMLS single ontology loader and the NCI Metathesaurus loader. These loaders that implement the Loader Framework simple must define the READ and TRANSFORMATION mechanisms for the load, as well as load order and flow. All common details of Loading to LexEVS will be dealt with by the Loader Framework and will not have to be implemented. Tools exist for:

1. Lucene Indexing
2. Registering CodingSchemes
3. Changing CodingScheme status (to ACTIVE, INACTIVE, etc)
4. Building the Transitivity Closure table
5. Adding Supported Attributes
6. Detecting Database type
7. Staging temporary data to the database
8. Restarting failed loads
9. Integrating with LexEVS logging
10. Detecting and handling Root Nodes
11. ...and more common LexEVS load related tasks

Also, to aid in Transformation, basic building blocks have been created that users may extend, such as:

1. Processors for all of LexEVS Model Objects
2. Various List Processors
3. Grouping Processors
4. Auto-Supported Attribute adding Processors
5. Several basic Resolvers to extract LexEVS Specific data from the source
6. ... various other Processors for specialized tasks

Several Utilities are also available for Reading and Writing, such as:

1. Group Readers
2. Group Writers
3. Writers configurable to skip certain records
4. Partitionable readers to break up large source files
5. Error checking Readers and Writers
6. A Validating framework for inspecting content before it is inserted into the database.
7. etc.

BDA Support Detailed Design

LexEVS uses the BDA (Build and Deployment Automation) system to build and deploy artifacts. This build script that produces these artifacts and deploys them is kicked off via a build server (an instance of Anthill pro).

Currently LexEVS has builds for a local java application, a web enabled java application, a data service, an analytical grid service and a data grid service.

Each of these is produced as a separate artifact via a single build call. The local Java application is built and integrated into the web application. The data service is also integrated into the web application. Separate analytical and data grid services are also created.

Once artifacts are successfully compiled and packaged in to jars, wars and zip files the build script makes contact with the remote server tier it is about to install to, and begins pushing artifacts out to application containers.

In this case two separate JBoss server instances are required. In one instance the web enabled application (lexevsapi) is copied to the remote server as a war file, it is accompanied by a grid data service zip file that is unzipped into a typical war file directory structure. In the other instance the LexEVS grid analytical service is deployed. In every case the server is stopped, configured as necessary, receives the new war file or directory and is started again. The BDA process is considered to be successful when all project builds are complete, all artifacts are deployed, and each public interface of these services is verified.

Implementation Plan

This will include the technical environment (HW,OS, Middleware), external dependencies, teams/locations performing development and procedures for development (e.g. lifecycle model,CM), and a detailed schedule.

Technical environment

No new environment requirements exist for the the LexEVS 5.1, with the exception of additional storage to accommodate larger content loads.

Software (Technology Stack)

Operating System

- Linux (though no operating system dependencies currently exist)

Application Server

- JBoss 4.0.5

Database Server

- MySQL 5.0.45

Other Software Components

- caGrid 1.3 / Globus 4.0.3

Server Hardware

Server

- NCI standard hardware.

Minimum Processor Speed

- Minimum required by JBoss.

Minimum Memory

- Minimum required by JBoss.

Storage

Expected file server disk storage (in MB)

- 200GB (May increase due to additional RRF content load)

Expected database storage (in MB)

- 100GB (May increase due to additional RRF content load)

Networking

Application specific port assignments

- Standard port required by JBoss to externalize LexEVS grid service. May be assigned any suitably available port #.

JBoss Container Considerations

There are specific requirements for JBoss containers for LexEVS 5.1.

In order to support multiple versions of LexEVS (for example 5.0 and 5.1), there are JBoss considerations.

- Both lexevsapi 5.0 and 5.1 in the same container. This means that both <http://lexevsapi.nci.nih.gov/lexevsapi50> and <http://lexevsapi.nci.nih.gov/lexevsapi51> can exist in the same Jboss container.
- Grid services can NOT be in the same JBoss container because of naming - they both need to be named /wsrf in the container. This results in the use of 1 JBoss container for LexEVS 5.0 and 1 JBoss container for LexEVS 5.1.

External dependencies

N/A

Team/Location performing development

- Traci St. Martin / Mayo Clinic
- Craig Stancl / Mayo Clinic
- Scott Bauer / Mayo Clinic
- Kevin Peterson / Mayo Clinic
- Sridhar Dwarkanath / Mayo Clinic
- Michael Turk / Mayo Clinic

Procedures for Development

Development will follow procedures as defined by NCI.

Detailed schedule

The LexEVS 5.1 project plan is located on [the Gforge archive in MS Project format](#) and [PDF format](#)

The LexEVS 5.1 BDA Project plan is located at [LexEVS 5.1 BDA Project Plan](#) (login required).

Training and documentation requirements

Documentation for LexEVS 5.1 will be provided by the Vocabulary Knowledge Center: [LexEVS Guides and Training](#).

Download center changes

Downloads for LexEVS 5.1 will be located on the [LexEVS](#) wiki.