LexBIG 1.0.2 Developer's Technical Guide

Version 1.0.2

Contents of this Page

LexBIG Software User Agreement

October 30, 2006

Usage of Content
THIS PRODUCTS MAKES AVAILABLE SOFTWARE, DOCUMENTATION, INFORMATION AND/OR OTHER MATERIALS (COLLECTIVELY "CONTENT"). USE OF THE CONTENT IS GOVERNED BY THE TERMS AND CONDITIONS OF THIS AGREEMENT AND/OR THE TERMS AND CONDITIONS OF LICENSE AGREEMENTS OR NOTICES INDICATED OR REFERENCED BELOW. BY USING THE CONTENT, YOU AGREE THAT YOUR USE OF THE CONTENT IS GOVERNED BY THIS AGREEMENT AND/OR THE TERMS AND CONDITIONS OF ANY APPLICABLE LICENSE AGREEMENTS OR NOTICES INDICATED OR REFERENCED BELOW. IF YOU DO NOT AGREE TO THE TERMS AND CONDITIONS OF THIS AGREEMENT AND THE TERMS AND CONDITIONS OF ANY APPLICABLE LICENSE AGREEMENTS OR NOTICES INDICATED OR REFERENCED BELOW, THEN YOU MAY NOT USE THE CONTENT.

Applicable Licenses
Unless otherwise noted, content is provided to you under terms and conditions of the following agreement:

```
Copyright: (c) 2004-2006 Mayo Foundation for Medical Education and
Research (MFMER).  All rights reserved.  MAYO, MAYO CLINIC, and the
triple-shield Mayo logo are trademarks and service marks of MFMER.

Except as contained in the copyright notice above, the trade names,
trademarks, service marks, or product names of the copyright holder shall
not be used in advertising, promotion or otherwise in connection with
this Software without prior written authorization of the copyright holder.
Licensed under the Eclipse Public License, Version 1.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

            http://www.eclipse.org/legal/epl-v10.html

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.
```

> (i) **Note**
>
> Content includes redistribution of additional 3rd party software modules, placed in the /runtime/extlib folder of the product root directory. Each module is accompanied by a license file that identifies specific terms of use and redistribution that may differ from the terms listed above. Modules include, but are not limited to, those listed in the appendix of this document.

IT IS YOUR OBLIGATION TO READ AND ACCEPT ALL SUCH TERMS AND CONDITIONS PRIOR TO USE OF THE CONTENT.

Contacts and Support

Type	Contact
Training contact	VKC VocabKC@mayo.edu
Developer	Division of Biomedical Informatics Mayo Clinic 200 1st ST SW Rochester, MN 55905 informatics@mayo.edu

Facilities Pertinent to Software Teams

Resource	URL	Description
Bug Tracking and feature requests	Bug Reports and Feature Requests To Include	JIRA
Discussion Forums and Project Wiki	This wiki	Vocabulary Knowledge Center

Architecture and Requirements specifications	`https://gforge.nci.nih.gov/scm/?group_id=491`	LexEVS SVN repository

Overview

This manual provides an introduction to the architecture, software components, and application programming interfaces (APIs) of the LexGrid Vocabulary Services for caBIG® project (hereafter referred to as the 'LexBIG' project).

Organization

Section Name	Section Contents
Overview of the Software	Provides an overview of the deployed software packages and describes a typical development/deployment scenario.
Systems and Architecture	Describes the functional pieces of the LexBIG runtime and how they interact with each other and calling programs.
Information Models	Describes formal models related to LexBIG content, including XML schema and UML representations.
LexBIG APIs	Describes the application programming interfaces provided by the LexBIG runtime.
Appendix A - References	References to formal publications and additional online resources.
Appendix B - Included Materials	Provides detailed information of additional software components packaged with the LexBIG distribution and their usage.

Getting Started

Recommendations for approaching this guide:

- Review the introduction to learn about the manual structure
- Review Overview of the Software for a brief overview of the software
- Review the source code distributed as examples and automated tests, and correlate to the LexBIG APIs as described in this guide.
- Based on these techniques, try to write example programs of your own.

Document Text Conventions

The following table shows various typefaces to differentiate between regular text and menu commands, keyboard keys, and text that you type. This illustrates how conventions are represented in this guide.

Convention	Description	Example
Bold & Capitalized Command	Indicates a Menu command	
Capitalized command>Capitalized command	Indicates Sequential Menu commands	**Admin > Refresh**
`Special typestyle`	Used for filenames, directory names, commands, file listings, source code examples and anything that would appear in a Java program, such as methods, variables, and classes.	`URL_definition ::= url_string`
Boldface type	Options that you select in dialog boxes or drop-down menus. Buttons or icons that you click.	Select the file and click the **Open** button.
Italics	Used to reference other documents, sections, figures, and tables.	*caCORE Software Development Kit 1.0 Programmer's Guide*
Italic boldface type	Text that you type	In the New Subset text box, enter ***P roprietary Proteins.***
(i)	Highlights a concept of particular interest	(i) **Note** This concept is used throughout the installation manual.

	Highlights information of which you should be particularly aware.

> ⊕ **Warning**
>
> Deleting an object will permanently delete it from the database.

Overview of the Software

This section assumes that the LexBIG software has been installed, per instructions provided by the *LexBIG Administrator's Guide*.

What's Inside

This section describes the location and organization of installed materials. Following installation, many of the following hierarchy of files and directories will be available (some features are optionally installable):

<located in the LexBIG installation root directory>

Directory	Description of Content
`admin`	Installed by default. This directory provides a centralized point for command line scripts that can be executed to perform administrative functions such as the loading, activation/deactivation, and removal of vocabulary resources. ⓘ **Note** Programmers may be interested in the code used to execute these functions. In that light, it is perhaps noteworthy to mention that this directory does not contain any source or binary code, only the scripts used to launch the admin functions. Object code used to carry out these functions is included directly in the LexBIG runtime components. Source code is included in the /source directory in the `lbAdmin-src.jar` (described below).
`doc`	Optionally installed. This directory provides documentation related to LexBIG services, configuration, and execution. This guide is distributed in the /doc top-level directory.
`/doc /javadoc`	*Of special interest to programmers.* This directory provides the generated javadoc for model classes and public interfaces available to LexBIG programmers. Also included with each object representation is a UML-based model diagram that shows the object, its attributes and operations, and immediately linked objects. The diagrams work to provide clickable navigation through the javadoc materials.
`/examples`	Optionally installed. This directory provides a small number of example programs. Refer to the README.txt file in this directory for instructions used to configure and run the example programs. The examples are intended to provide a limited interactive demonstration of LexBIG capabilities. Source and object code for the example programs is provided under the /examples/org subdirectory. Source materials are also centrally archived under the /source directory in the file `lbExamples-src.jar`.
`/examples /resources`	Contains sample vocabulary content for reference by the example programs; use the /examples/LoadSampleData command-line script to load.
`/gui`	Optionally installed. This folder contains programs and supporting files to launch the LexBIG Graphical User Interface (GUI). The GUI provides convenient centralized access to administrative functions as well as support to test and exercise most of the LexBIG API. The GUI is launched using a platform-specific script file in the /gui directory. The name of the platform (e.g. Windows, OSX, etc) is included in the file name. Program source and related materials are centrally archived under the /source directory in the file `lbGUI-src.jar`.
`/logs`	Default location for log files, which can be modified by the `LOG_FILE_LOCATION` entry in the `config.props` file (see next section).
`/resources`	Installed by default. This directory contains resources referenced and written directly by the LexBIG runtime. It should, in general, be considered off-limits to modify or remove the content of this directory without specific guidance and reason to do so. Files typically stored to this location include the vocabulary registry (tracking certain metadata for installed content) and indexes used to facilitate query over the installed content. One file of particular interest in this directory is the `/resources/config/config.props` file. This file controls access to the database repository and other settings used to tune the LexBIG runtime behavior. Contents of this file should be set according to instructions provided by the *LexBIG Administrator's Guide*.

/runtime	Installed by default. This directory contains a Java archive (.jar) file containing the combined object code of the LexBIG runtime, LexBIG administrative interfaces, and any additional code they are dependent on. All required code for execution of LexBIG administrative and runtime services is installed to this directory. ⓘ **Note** Java programmers writing to the LexBIG runtime interfaces should always include the following files in their java classpath (listed in order of inclusion): • `/runtime/lbPatch.jar` - In the course of the product lifecycle, it is possible that smaller fixes will be introduced as a patch to the initially distributed runtime. Including this file in the classpath ensures automatic accessibility to the calling program without requiring adjustment. All patches are cumulative (there is at most one patch file introduced per release; all patch-level fixes are cumulative). • `/runtime/lbRuntime.jar` - This is the standard runtime file, including all LexBIG and dependency code required for program execution except for SQL drivers (see next).
/runtime /sqldriv ers	The JDBC drivers used to connect to database repositories are not included in the `lbRuntime.jar`. Instead, the runtime scans this directory for the drivers to include. This can be overridden by path settings in the `config.props` file. ⓘ **Note** While the LexBIG software package ships with JDBC drivers to certain open source databases such as mySQL and PostgreSQL, this folder provides a mechanism to introduce updated drivers or to add drivers for additional supported database systems. For example, the Oracle database is supported by the runtime environment. However, the drivers are not redistributed with the LexBIG software. To run against Oracle, an administrator would add a jar with the appropriate JDBC driver to this directory and then reference it in the `config.props` settings.
/runtime - componen ts	Optionally installed. Due to license considerations for additional materials (as described by the `license.pdf` and `license.txt` files in the install directory), the cumulative runtime provided in the `lbRuntime.jar` is not redistributable. This directory contains a finer grain breakdown of object code into logical components and 3rd party inclusions. All components are redistributable under their own license agreements, which are provided along with each archive. The top-level of the `/runtime-components` directory contains all code produced for the LexBIG project in a single lexbig.jar file. ⓘ **Note** These files are included as an alternative to the `lbRuntime.jar` for code execution and redistribution. There is no need to include any of these files in the Java classpath if you are already including the `lbPatch.jar` and `lbRuntime.jar` described above.
/runtime - componen ts /extlib	This subdirectory includes all 3rd party code redistributed with the LexBIG runtime, along with respective license agreements.
/source	Archive source directories and files described in further detail in the next table below.
/test	Optionally installed. This directory provides an automated test bucket that can be used by System Administrators to verify node installation. Note that the `/runtime/config/config.props` file must still be configured for database access prior to invoking the test bucket. Testcases are launched via the `TestRunner` command-line script. Several reporting options are provided and are further described in the _LexBIG Administrator's Guide_. ⓘ **Note** Programmers may be interested in referencing the source code for the test programs. These are provided in the /source directory (described in the next table).
/uninsta ller	Contains an executable jar that can be invoked by an administrator to uninstall files originally introduced by the LexBIG installation.

The following table describes the source subdirectories and the associated content.

`/source` **Subdirectories**	**Description of Content**
lbAdmin-src	Source for LexBIG administrative interfaces.

`lbExamples-src`	Corresponds to programs provided in the `/examples` directory (described above).
`lbGUI-src`	Source for the Graphical User Interface.
`lbImpl-src`	Source for implementation classes fulfilling the LexBIG service interfaces and interacting with models.
`lbInterfaces-src`	Source defining service-level interfaces for the LexBIG runtime.
`lbModel-src`	Source defining LexBIG-specific extensions to the LexGrid information model.
`lbTest-src`	Corresponds to automated test programs provided in the `/test` directory (described in the table above).
`lgConverter-src`	Source containing services to convert data into various formats.
`lgIndexer-src`	Source for the services used in indexing databases.
`lgModel-src`	Source defining the LexGrid Model.
`lgModel.emf-src`	Source for the EMF representation of the LexGrid Model.
`lgRDFConverter-src`	Framework for reading different types of ontology resources.
`lgResourceReader-src`	Source for handling RDF conversions.
`lgUtility-src`	Source for LexBIG utility programs.

A Simple Example

This section describes the basic steps involved in writing and deploying a program that directly invokes the LexBIG Java software components.

Establish the Development Environment

Installation should first be performed to the host server according to procedures described in the *LexBIG Administrator's Guide*. After installation, developers may wish to copy the components required for program development to a local environment. Installed components of interest include the following:

- `/runtime/lbRuntime.jar` - This archive contains the combined code for all executable code, as described earlier in this section. Adding this file to the Java classpath during development will allow compilation of source code.
- `/runtime-components/lexbig.jar` - While this jar does not include all code required for runtime execution, it does provide a sufficient alternative to the *lbRuntime.jar* file for purposes of program compilation.
- `/source/lb*.jar, lg*.jar` - Contain the source code for the LexBIG interfaces. Many development environments (e.g. Eclipse) will provide the ability to link these files to the object code archives. This allows for improved reference materials during program development.

Write and Compile Programs

Example source is provided below for a simple program that lists the available coding schemes (containers for vocabulary concepts and relations) registered to a LexBIG server node:

```
import org.LexGrid.LexBIG.DataModel.Collections.CodingSchemeRenderingList;
import org.LexGrid.LexBIG.DataModel.InterfaceElements.CodingSchemeRendering;
import org.LexGrid.LexBIG.Impl.LexBIGServiceImpl;
import org.LexGrid.LexBIG.LexBIGService.LexBIGService;
import org.LexGrid.LexBIG.Utility.ObjectToString;

public class ListCodeSystems {
    public static void main(String[] args) {
        try {
            LexBIGService lbs = new LexBIGServiceImpl();
            CodingSchemeRenderingList schemes = lbs.getSupportedCodingSchemes();
            for (CodingSchemeRendering csr
                    : schemes.getCodingSchemeRendering() )
                System.out.printIn(

    ObjectToString.toString(csr.getCodingSchemeSummary()));
        } catch (Exception e) {
            e.printStackTrace();
        }
```

> (i) **Note**
>
> Sun Java Development Kit (JDK) level 5.0 or above is required for program development. The code base has not been updated to accommodate Java 6.

Deploy and Run

Continuing the example above, the compiled `ListCodeSystems.class` file should be copied to the LexBIG server system for invocation. Currently LexBIG services are invoked through direct programmatic access, though future direction is to provide additional web or grid-level service invocation in support of multi-tier scenarios. Additional description of system components and deployment scenarios is provided in the section, Systems and Architecture.

As with development, program execution requires the Sun Java JDK (or JRE) version 5.0 or above. Assuming the Java command is accessible in the path and the ListCodeSystems class file is installed to a directory of choice {pgm-dir} and the LexBIG is installed to {lexbig-dir}, the program can be invoked through the following command line interface:

```
java -cp {lexbig-dir}/runtime/lbPatch.jar:{lexbig-dir}/runtime/lbRuntime.jar:{pgm-dir} ListCodeSystems
```

> ⓘ **Note**
>
> The above example uses syntax for a Linux installation; Windows users would use alternate syntax for file separator ('\') and classpath separator (';').

As mentioned previously, it is considered good practice for developers to include the `lbPatch` archive in their class path, as it is designed to introduce smaller fixes without requiring download and redeployment of the complete (and much larger) runtime jar.

System Requirements

Refer to the *LexBIG Administrator's Guide* for minimum recommended requirements for the system hosting the LexBIG runtime. There are no unique requirements for development systems other than those enforced by the development tools being used (see next section).

Software and Hardware Dependencies

Refer to the *LexBIG Administrator's Guide* regarding software and hardware dependencies for the system hosting the LexBIG runtime. Development systems are required to install the Sun Java Development Kit (JDK) or Java Runtime Environment (JRE) version 5.0 or above. In addition, an integrated development environment (IDE) such as Eclipse or NetBeans is recommended for program development.

Security and Security Management

The LexBIG runtime relies on standard operating system and database infrastructure to provide security of data and access to runtime services. Any additional requirements are noted in the *LexBIG Administrator's Guide*. There are no unique security requirements for development systems.

Systems and Architecture

This section describes the functional pieces of the LexBIG runtime and how they interact with each other and calling programs. Topics include a high level description of the LexGrid information model, LexBIG model and service extensions, and current/future deployment scenarios.

The Basics

What is LexGrid?

LexGrid is an initiative of the Mayo Clinic Division of Biomedical Informatics that focuses on the representation, storage, and dissemination of vocabularies. This effort centers on, but is not limited to, the domain of medical vocabularies and nomenclatures. Focal points of the LexGrid project include the development and promotion of standards, tools, and content that:

- Provide flexibility to represent yesterday's, today's and tomorrow's terminological resources using a single information model.
- Provide the ability for these resources to be published online, cross-linked, and indexed.
- Provide standardized building blocks and tools that allow applications and users to take advantage of the content where and when it is needed.
- Provide consistency and standardization required to support large-scale terminology adoption and use.

What is LexBIG?

LexBIG is a more specific project that applies LexGrid vision and technologies to requirements of the caBIG® community. The goal of the project is to build a vocabulary server accessed through a well-structured application programming interface (API) capable of accessing and distributing vocabularies as commodity resources. The server is to be built using standards-based and commodity technologies. Primary objectives for the project include:

- Provide a robust and scalable open source implementation of EVS-compliant vocabulary services. The API specification will be based on but not limited to fulfillment of the caCORE EVS API. The specification will be further refined to accommodate changes and requirements based on prioritized needs of the caBIG® community.
- Provide a flexible implementation for vocabulary storage and persistence, allowing for alternative mechanisms without impacting client applications or end users. Initial development will focus on delivery of open source freely available solutions, though this does not preclude the ability to introduce commercial solutions (e.g. Oracle).
- Provide standard tooling for load and distribution of vocabulary content. This includes but is not limited to support of standardized representations such as UMLS Rich Release Format (RRF), the OWL web ontology language, and Open Biomedical Ontologies (OBO) .

The goal for the initial year of development was to achieve the Bronze level of compatibility with regard to the caBIG® requirements. Silver-level compatibility is being pursued.

Software Overview

LexBIG software architecture and implementation is designed to facilitate flexibility and future expansion. The following diagrams are intended to aid the understanding of LexBIG service integration in context of the larger caBIG® universe and specific deployment scenarios.

The following diagram depicts the LexBIG vision. Individual Cancer Centers will be able to use the existing set of caCORE EVS services. If desired, local instances of vocabularies can be installed.
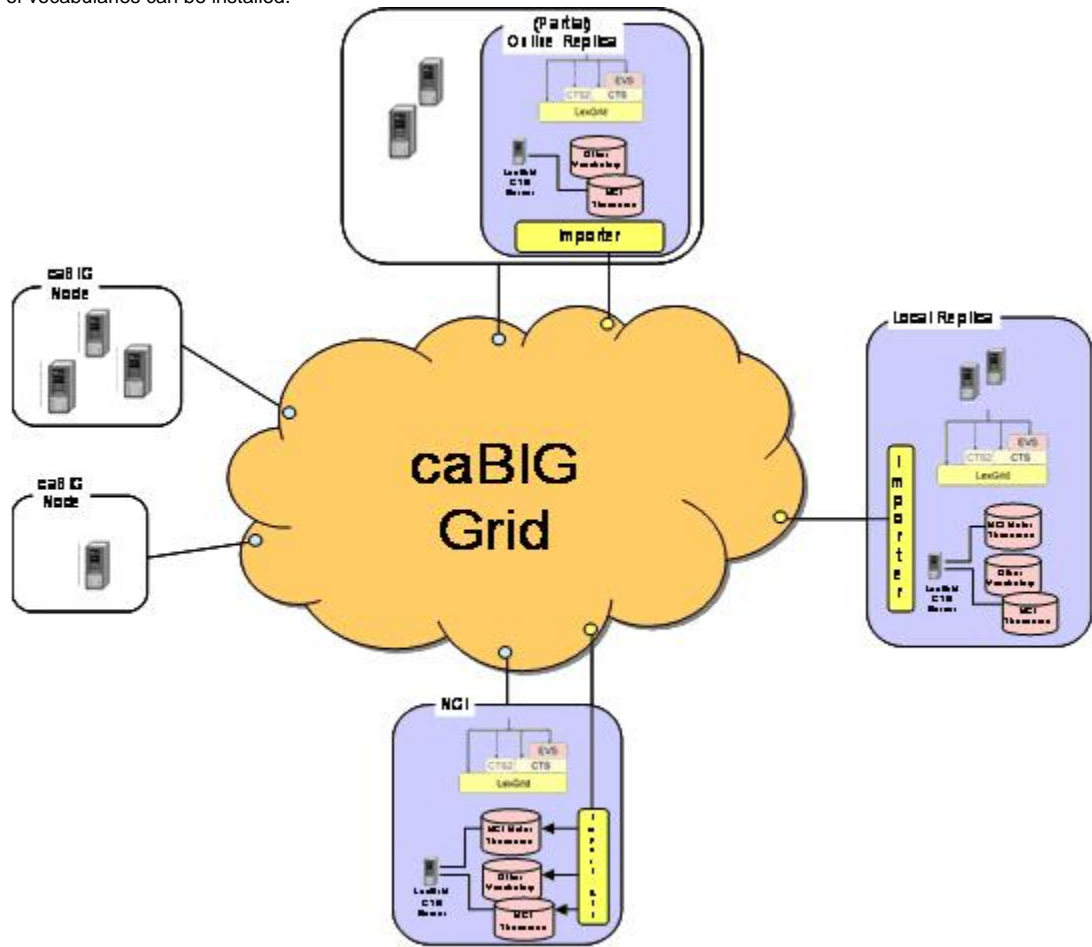
The following diagram depicts direct programmatic access (Java-to-Java access) to LexBIG functions. This is the primary deployment scenario for phase 1.

LexBIG Runtime
Direct Programmatic Access
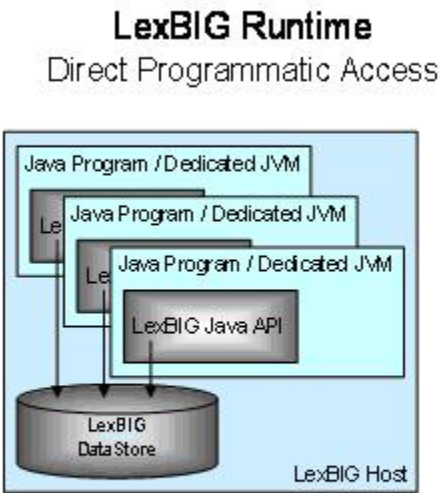
ⓘ **Note**

It is not required that the database be located on the same system as the program runtime.

The following diagram depicts consolidated access through caCORE Enterprise Vocabulary Services (EVS) to a LexBIG vocabulary engine.

LexBIG Runtime
Consolidated Host – EVS Access
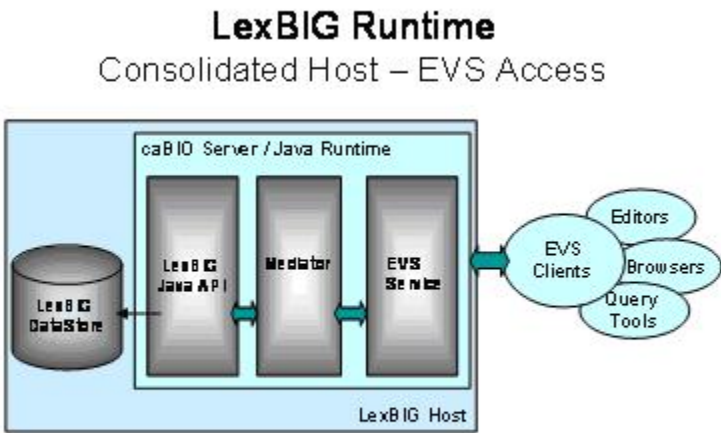
The primary goal is to provide a compatible experience for existing EVS browsers and client applications.

ⓘ **Note**

This diagram shows the possible inclusion of a mediation layer between EVS and the LexBIG runtime.
This would be done to facilitate alternate communications with the LexBIG server (e.g. through web services as described below).

The LexBIG API is designed with Web and grid-level enablement in mind. The following diagram depicts deployments that wrap the current API to allow the runtime to be accessed in a consolidated way through Web or grid services.

LexGrid Model

Overview

The LexGrid Model is Mayo Clinic's proposal for standard storage of controlled vocabularies and ontologies. The LexGrid Model defines how vocabularies should be formatted and represented programmatically, and is intended to be flexible enough to accurately represent a wide variety of vocabularies and other lexically-based resources. The model also defines several different server storage mechanisms and a XML format. This model provides the core representation for all data managed and retrieved through the LexBIG system, and is now rich eno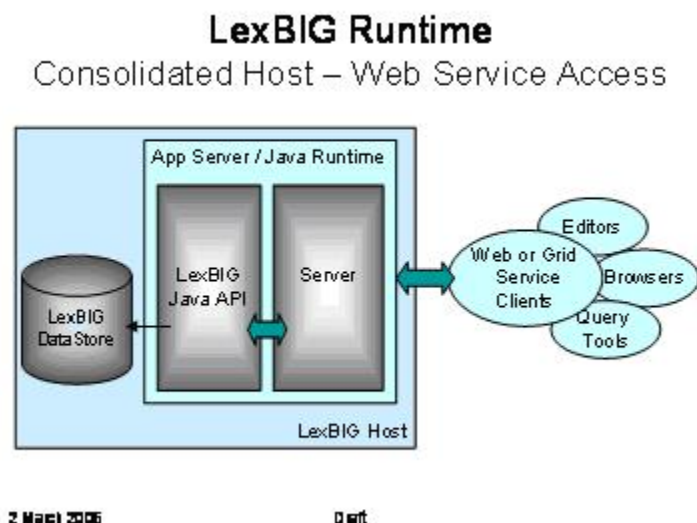ugh to represent vocabularies provided in numerous source formats such as OWL (NCI Thesaurus) and RRF (NCI MetaThesaurus).

Once the vocabulary information is represented in a standardized format, it becomes possible to build common repositories to store vocabulary content and common programming interfaces and tools to access and manipulate that content. The LexBIG API developed for caBIG® is one such interface, and is described in additional detail in LexBIG APIs.

Following are some of the higher-level objects incorporated into the model definition:

Code Systems

Each service defined to the LexGrid model can encapsulate the definition of one or more vocabularies. Each vocabulary is modeled as an individual code system, known as a *codingScheme*. Each scheme tracks information used to uniquely identify the code system, along with relevant metadata. The collection of all code systems defined to a service is encapsulated by a single *codingSchemes* container.

Concepts

A code system may define zero or more coded concepts, encapsulated within a single container. A concept represents a coded entity (identified in the model as a *concept*) within a particular domain of discourse. Each concept is unique within the code system that defines it. To be valid, a concept must be qualified by at least one designation, represented in the model as a *property*. Each property is an attribute, facet, or some other characteristic that may represent or help define the intended meaning of the encapsulating concept. A concept may be the source for and/or the target of zero or more relationships. Relationships are described in more detail in a following section.

Relations

Each code system may define one or more containers to encapsulate relationships between concepts. Each named relationship (e.g. "hasSubtype" or "hasPart") is represented as an *association* within the LexGrid model. Each relations container must define one or more association. The association definition may also further define the nature of the relationship in terms of transitivity, symmetry, reflexivity, forward and inverse names, etc. Multiple instances of each association can be defined, each of which provide a directed relationship between one source and one or more target concepts.

Source and target concepts may be contained in the same code system as the association or another if explicitly identified. By default, all source and target concepts are resolved from the code system defining the association. The code system can be overridden by each specific association, relation source (*associationInstance*), or relation target (*associationTarget*).

LexBIG Model

Overview

The LexBIG vocabulary model extends the LexGrid model to provide unique constructs or granularity required by caBIG® that are not present in the core model. While many extensions exist, this document will focus on some of direct relevance to the high-level architecture.

Concept Resolution

LexBIG allows the service runtime to provide managed resolution of code-based objects that are referenced through LexBIG-specific lists and iterators (mechanism that allow streaming of list content). These lists and iterators are typically returned when requesting sets or graphs of vocabulary terms through the LexBIG API (described in LexBIG APIs). Some model components involved in the resolution process include:

- `ConceptReference` - A globally unique reference to a concept code.
- `ResolvedConceptReference` - A concept reference for which additional information has been resolved, including description and relationship participation.
- `AssociatedConcept` - A concept reference that contains full detail in participation as a source or target of an association, including indications of navigability and qualification.

> ⓘ **Note**
>
> Formal representation of the LexGrid and LexBIG models are discussed in the section, Information Models.

LexBIG Services

This section describes architectural detail for services provided by the LexBIG system. These services are geared toward the administration, management, and serving of vocabularies defined to the LexGrid/LexBIG information model. A system overview is provided, followed by a description of key subsystems and components. Each subsystem is described in terms of its overall structure, formal model, and specification of key public interfaces.
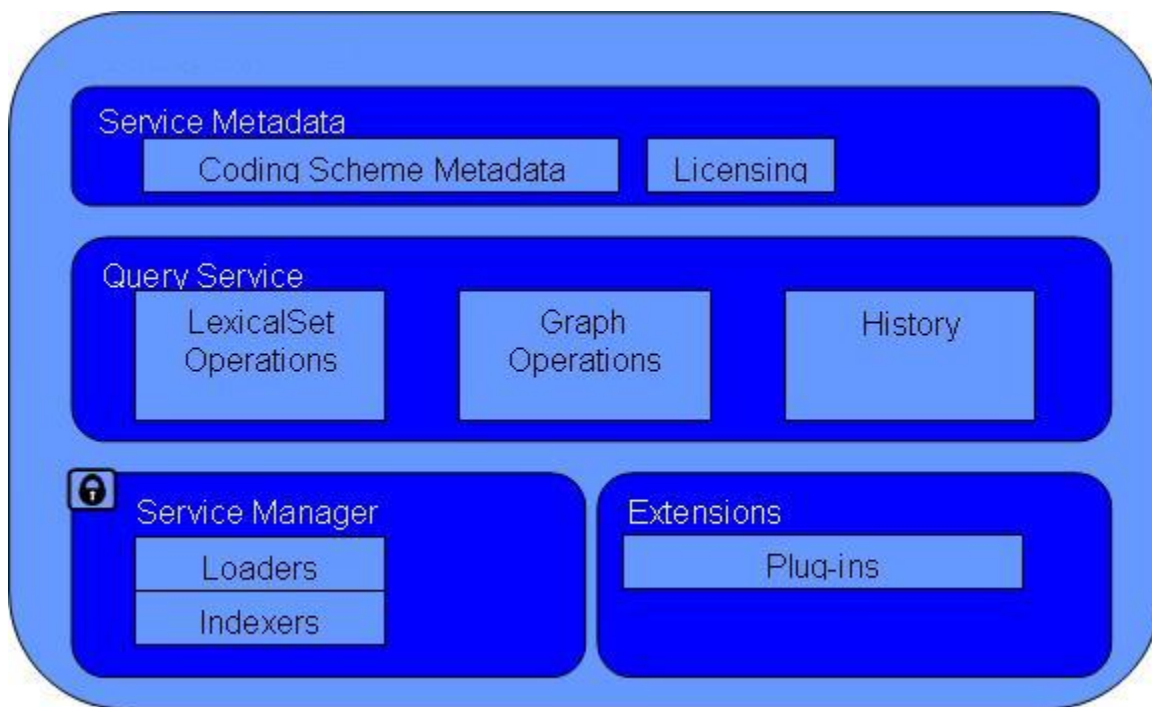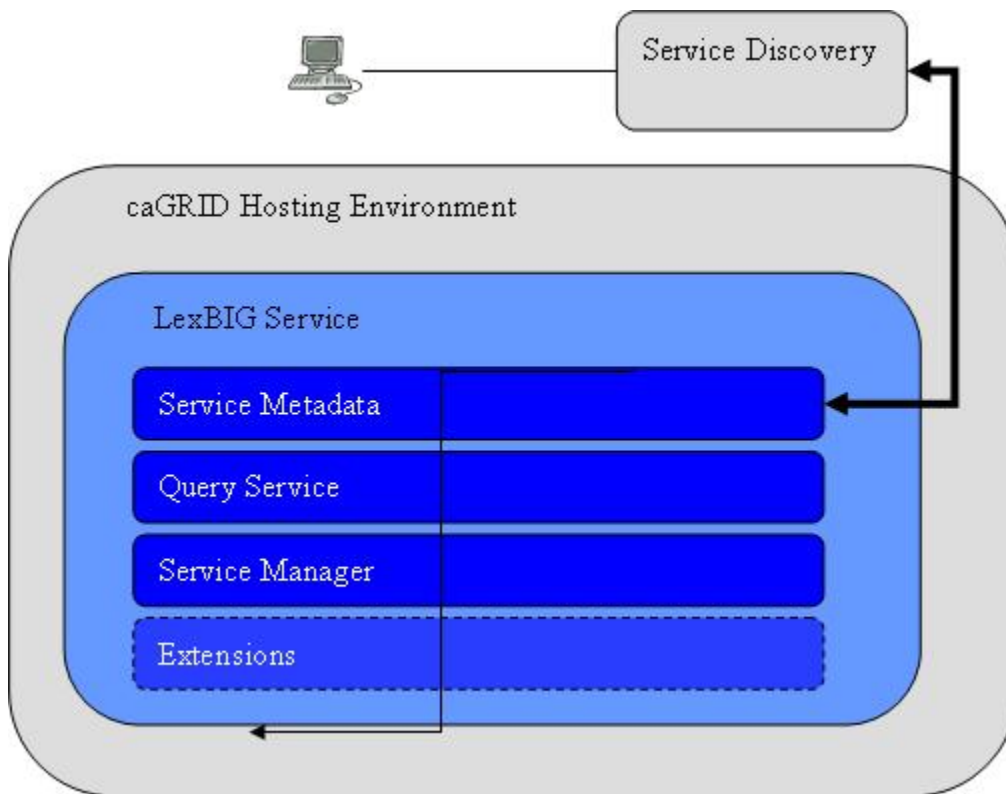

Overview

The LexBIG Service is designed to run standalone or as part of a larger network of services. It is comprised of four primary subsystems: Service Management, Service Metadata, Query Operations, and Extensions. The Service Manager provides administration control for loading a vocabulary and activating a service. The Service Metadata provides external clients with information about the vocabulary content (e.g. NCI Thesaurus) and appropriate licensing information. The Query Operations provide numerous functions for querying and traversing vocabulary content. Finally, the extensions component provides a mechanism to extend the specific service functions, such as Loaders, or re-wrap specific query operations into convenience methods. Primary points of interaction for programming include the following classes:

- `LexBIGService` - This interface provides centralized access to all LexBIG services.
- `LexBIGServiceManager` - The service manager provides a centralized access point for administrative functions, including write and update access for a service's content. For example, the service manager allows new coding schemes to be validated and loaded, existing coding schemes to be retired and removed, and the status of various coding schemes to be updated and changed.

caGRID Hosting

Overview

The LexBIG architecture provides the underpinnings LexBIG services to be made accessible through the caGRID environment in the future, where LexBIG services might optionally be deployed in a caGRID Globus container. caGrid provides a Globus service for service registration and discovery. LexBIG services deployed to the grid would be registered in the NCICB registry and be searchable through the NCICB index service.

Specification

Additional specifications related to the registration and discovery of LexBIG services in the caGRID environment will be included later phases of work in concordance with caGRID 1.0. This is will be coordinated with caBIG® Architecture workspace designees.

Service Management Subsystem
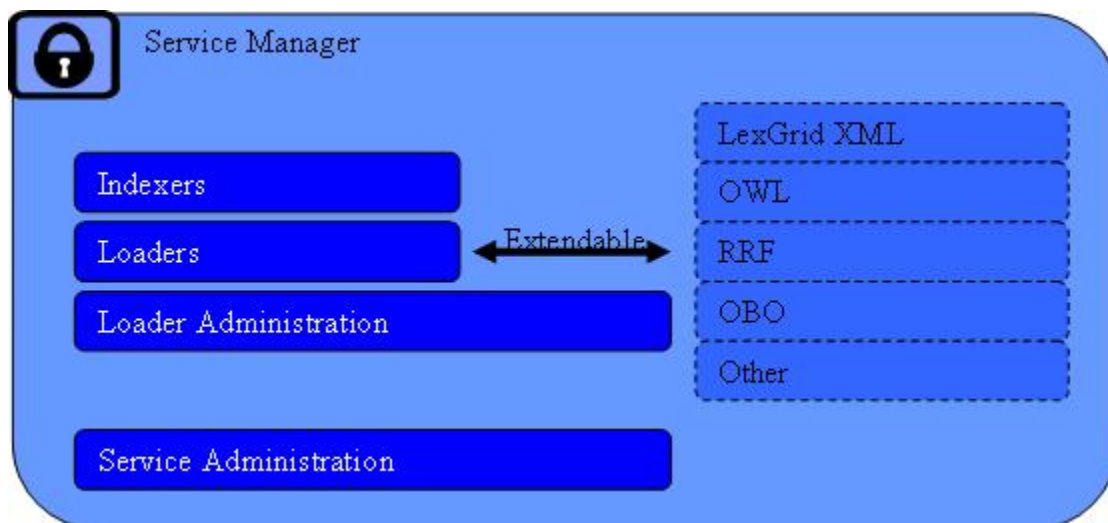
Overview

This subsystem provides administrative access to functions related to management and publication of LexBIG vocabularies. These functions are generally considered to be reserved for LexBIG administrators, with detailed instructions on how to secure and carry out related tasks described by the *LexBIG Administrator's Guide*.

This subsystem is further broken down into the following components:

- **Indexers**- Vocabularies may be indexed to provide enhanced performance or query capabilities. Types of indexes incorporated into the LexBIG system include but are not limited to the following:
    - Lexical Match - for example, "begins-with" and "contains"
    - Phonetic - allows for the ability to query based on "sounds-like" entry of search criteria.
    - Stemming - allows for the ability to find lexical variations of search terms.
      Index creation is typically bundled into the load process. Architecturally speaking, however, this capability is decoupled and extensible.
- **Loaders**- Vocabularies may be imported to the system from a variety of accepted formats, including but not limited to:
    - LexGrid XML (LexBIG canonical format)
    - NCI Thesaurus, provided in Web Ontology Language format (OWL)
    - UMLS Rich Release format (RRF)
    - Open Biomedical Ontologies format (OBO)

As with indexers, the load mechanism is designed to be extensible from an architectural standpoint. Additional loaders can be supported by the introduction of pluggable modules. Each module is implemented in the Java programming language according to a LexBIG-provided interface, and registered to the loader runtime environment.

Metadata and Discovery Subsystem

Overview

This subsystem provides information about accessible vocabularies, related licensing/copyright information, and registration/discovery of LexBIG services.

The ability to locate and resolve vocabulary metadata is fulfilled through the LexBIGService class. Metadata defined by the LexGrid information model is resolved with each CodingScheme instance. Available metadata on each resolved scheme includes, but is not necessarily limited to, the following:

- License or copyright information
- Supported values (e.g. supported concept status, language, property names, etc)
- Mappings from names used locally to globally unique URNs

In addition, each LexBIGService provides a centralized metadata index that allows registration and query of code system metadata without requiring resolution of individual CodingSchemes. This metadata index is optionally populated, typically during the vocabulary load process. The metadata index allows for the metadata of multiple code systems to be cross-indexed and searched as part of the query subsystem.

Finally, the LexBIG architecture provides the underpinnings for LexBIG services to be made accessible through the caGRID environment in the future, where vocabulary services might be deployed and discovered within a caGRID Globus container. However, this portion of the API is preliminary and awaits coordination with caBIG® Architecture WS designees to determine exact recommendations and nature of LexBIG services on the grid.

Query Subsystem
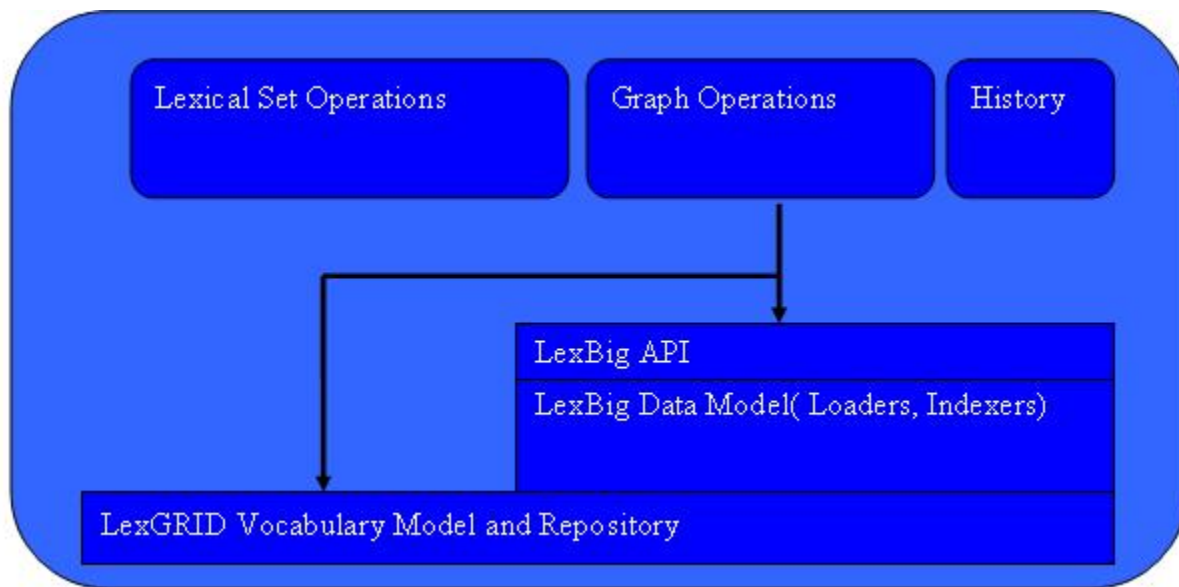
Overview

This subsystem provides the functionality required to fulfill caCORE/EVS and other vocabulary requests. The Query Service is comprised of Lexical Operations, Graph Operations, Metadata, and History Operations.

- **Lexical Set Operations** - Lexical Set Operations provides methods to return a lists or iterators of coded entries. Supported query criteria include the application of match/filter algorithms, sorting algorithms, and property restrictions. Support is also provided to resolve the union, intersection or difference of two node sets.
- **Graph Set Operations** - Graph Operations support the subsetting of concepts according to relationship and distance, identification of relation source and target concepts, and graph traversal. Additional operations include enumeration and traversal of concepts by relation, walking of directed acyclic graphs (DAGs), enumeration of source and target concepts for a relation, and enumeration of relations for a concept.
- **Metadata Operations** - Metadata Operations allows for the query and resolution of registered code system metadata according to specified coding scheme references, property names, or values.
- **History Operations** - History provides vocabulary-specific information about concept insertions, modifications, splits, merges, and retirements when supplied by the content provider.

Information Models

A brief introduction to the information models referenced by the LexBIG runtime are provided in the section, Systems and Architecture. This section will extend on this introduction, providing a brief description of classes included by the base LexGrid information model and LexBIG-specific extensions to that model.

> ⓘ **Note**
>
> The information below is provided for introductory purposes. A full description of all available model components is also available in the javadoc distributed with the LexBIG installation package (see file breakdown in Overview of the System). Since the javadoc is automatically generated and synchronized during the build process, it is recommended as the primary reference for use by LexBIG developers.

LexGrid Model

The LexGrid model is mastered in XML Schema. The LexBIG project currently builds on the 2008 version of the LexGrid schema. A formal representation, showing portions of this structure that are of primary interest to the LexBIG project, is presented below. A complete version of the model is available at Lex Big Model and Schema.

CodingSchemes

The CodingSchemes branch of the model defines high level containers for concepts and relations. Each CodingScheme represents a unique code system or version in the LexBIG service. Components of interest are described in the table below, with a corresponding class diagram following the table.

CodingScheme Component	Description
codingSchemes	Directory of coding schemes contained in the service.
codingScheme	Describes and/or defines a code system, comprising a collection of concept codes and relationships.
concepts	A set of coded entries in a coding scheme.

relations	A collection of relations across a set of concept codes drawn from one or more coding schemes.
versions	A list of past versions of the coding scheme.

> ⓘ **Note**
>
> While listed for completeness, note that this portion of the model is not referenced at this time by the LexBIG API. In the first release, history information is handled as a series of NCIChangeEvents (see LexBIG extensions below) by the LexBIG HistoryService.

cd codingSchemes

codingSchemes

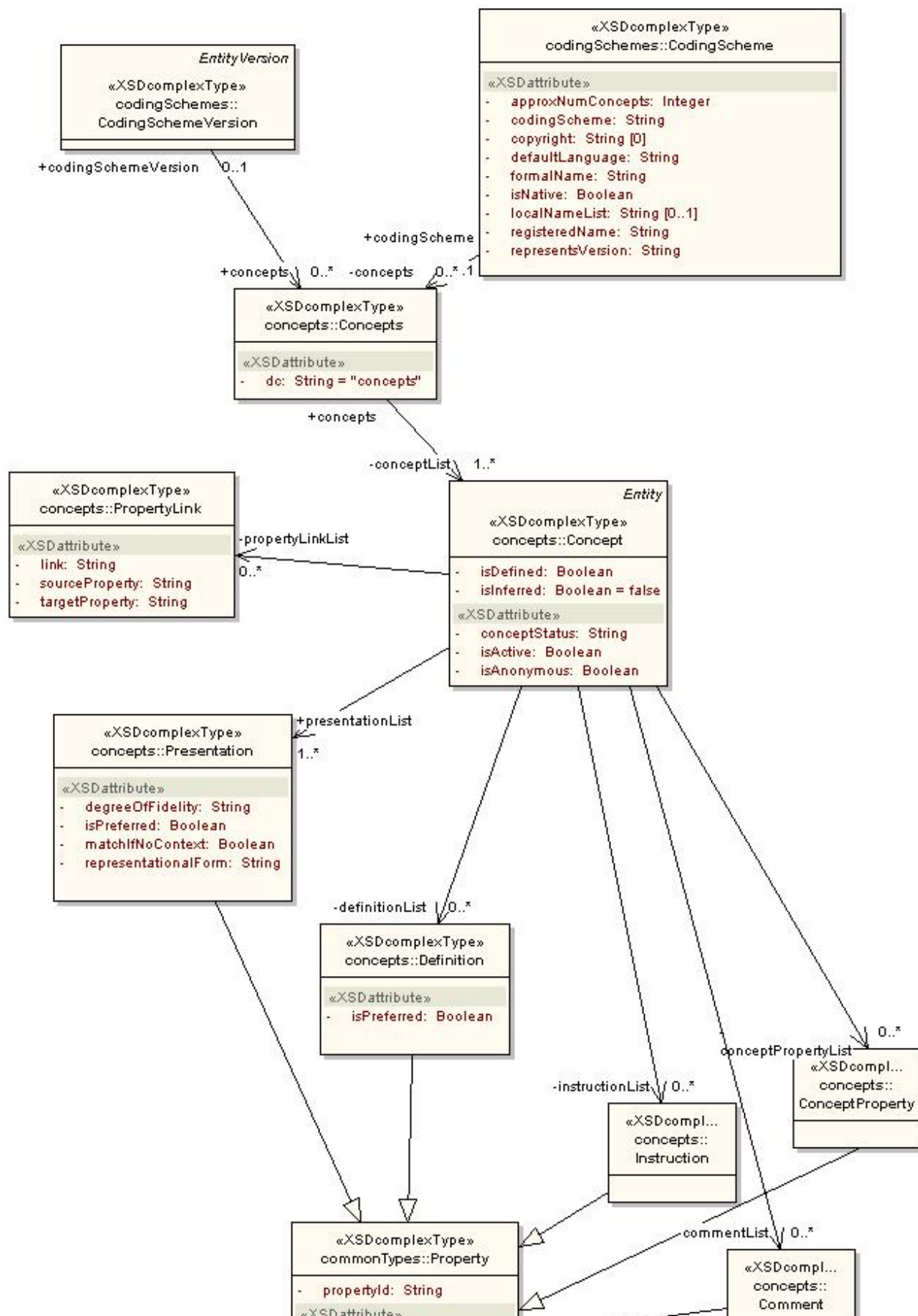+ «XSDattribute» dc: dc [0..1] = codingSchemes

codingScheme *describable*

+codingScheme 1..*

+ «XSDattribute» approxNumConcepts: tsInteger [0..1]
+ «XSDattribute» codingScheme: localName
+ «XSDattribute» defaultLanguage: defaultLanguage
+ «XSDattribute» formalName: tsCaseIgnoreIA5String
+ «XSDattribute» isNative: tsBoolean [0..1]
+ localName: localName [1..-1]
+ «XSDattribute» registeredName: registeredName
+ «XSDattribute» representsVersion: version
+ source: source [0..-1]
+ supportedAssociation: supportedAssociation [0..-1]
+ supportedAssociationQualifier: supportedAssociationQualifier [0..-1]
+ supportedCodingScheme: supportedCodingScheme [0..-1]
+ supportedConceptStatus: supportedConceptStatus [0..-1]
+ supportedContext: supportedContext [0..-1]
+ supportedDataType: supportedDataType [0..-1]
+ supportedFormat: supportedFormat [1..-1]
+ supportedLanguage: supportedLanguage [1..-1]
+ supportedProperty: supportedProperty [1..-1]
+ supportedPropertyLink: supportedPropertyLink [0..-1]
+ supportedRepresentationalForm: supportedRepresentationalForm [0..-1]
+ supportedSource: supportedSource [0..-1]

+relations [0..*]

relations::relations *describable*

+ «XSDattribute» dc: dc
+ «XSDattribute» isNative: tsBoolean [0..1]
+ source: source [0..-1]

+versions [0..1]

versions

+ «XSDattribute» dc: dc [0..1] = version

+concepts [0..1]

concepts::concepts

+ «XSDattribute» dc: dc [0..1] = Concepts

Concepts

Each concept represents a unique entity within the code system, which can be further described by properties and related to other concepts through relations.
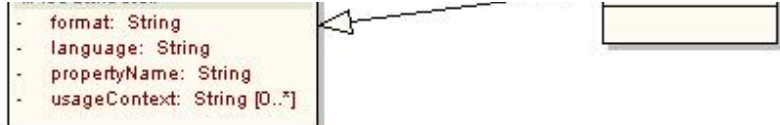
class Diagrams

EntityVersion
«XSDcomplexType»
codingSchemes::
CodingSchemeVersion

«XSDcomplexType»
codingSchemes::CodingScheme

«XSDattribute»
- approxNumConcepts: Integer
- codingScheme: String
- copyright: String [0]
- defaultLanguage: String
- formalName: String
- isNative: Boolean
- localNameList: String [0..1]
- registeredName: String
- representsVersion: String

+codingSchemeVersion   0..1

+codingScheme

+concepts   0..*   -concepts   0..*   1

«XSDcomplexType»
concepts::Concepts

«XSDattribute»
- dc: String = "concepts"

+concepts

-conceptList   1..*

Entity
«XSDcomplexType»
concepts::Concept

- isDefined: Boolean
- isInferred: Boolean = false
«XSDattribute»
- conceptStatus: String
- isActive: Boolean
- isAnonymous: Boolean

«XSDcomplexType»
concepts::PropertyLink

«XSDattribute»
- link: String
- sourceProperty: String
- targetProperty: String

-propertyLinkList

0..*

«XSDcomplexType»
concepts::Presentation

«XSDattribute»
- degreeOfFidelity: String
- isPreferred: Boolean
- matchIfNoContext: Boolean
- representationalForm: String

+presentationList

1..*

-definitionList   0..*

«XSDcomplexType»
concepts::Definition

«XSDattribute»
- isPreferred: Boolean

-instructionList   0..*

«XSDcompl...
concepts::
Instruction

conceptPropertyList

0..*

«XSDcompl...
concepts::
ConceptProperty

«XSDcomplexType»
commonTypes::Property

- propertyId: String
«XSDattribute»

commentList   0..*

«XSDcompl...
concepts::
Comment

Components of interest are described in the table below:

Concept Component	Description
concept	Represents a unique code for a concept within a coding scheme or a coding scheme version, along with an associated description and properties.
comment	A comment or annotation property for a concept.
definition	A definitional property for a concept.
instruction	A formal instruction for the use of a concept.
presentation	A designation for a concept. The presentation identifier must, at bare minimum, uniquely map to a given text string within the context of the containing concept. In some terminologies, every unique text string will have exactly one presentation identifier, which means that the same presentation identifier may occur under more than one concept. In other terminologies, there may be more than one identifier for a given text string, meaning that the presentation identifier uniquely determines the concept. Service software must not assume **either** model. (See: property for additional elements).
property	A description, definition, annotation or other attribute that serves to further define or identify a coded term. Property names must be included in the supportedProperty metadata for the coding scheme.
propertyLink	A link between two properties for a concept.. Examples include acronymFor, abbreviationOf, spellingVariantOf, etc. Link identifiers must be included in the supportedPropertyLink metadata for the coding scheme.

Relations

Relations are used to define and qualify associations between concepts.
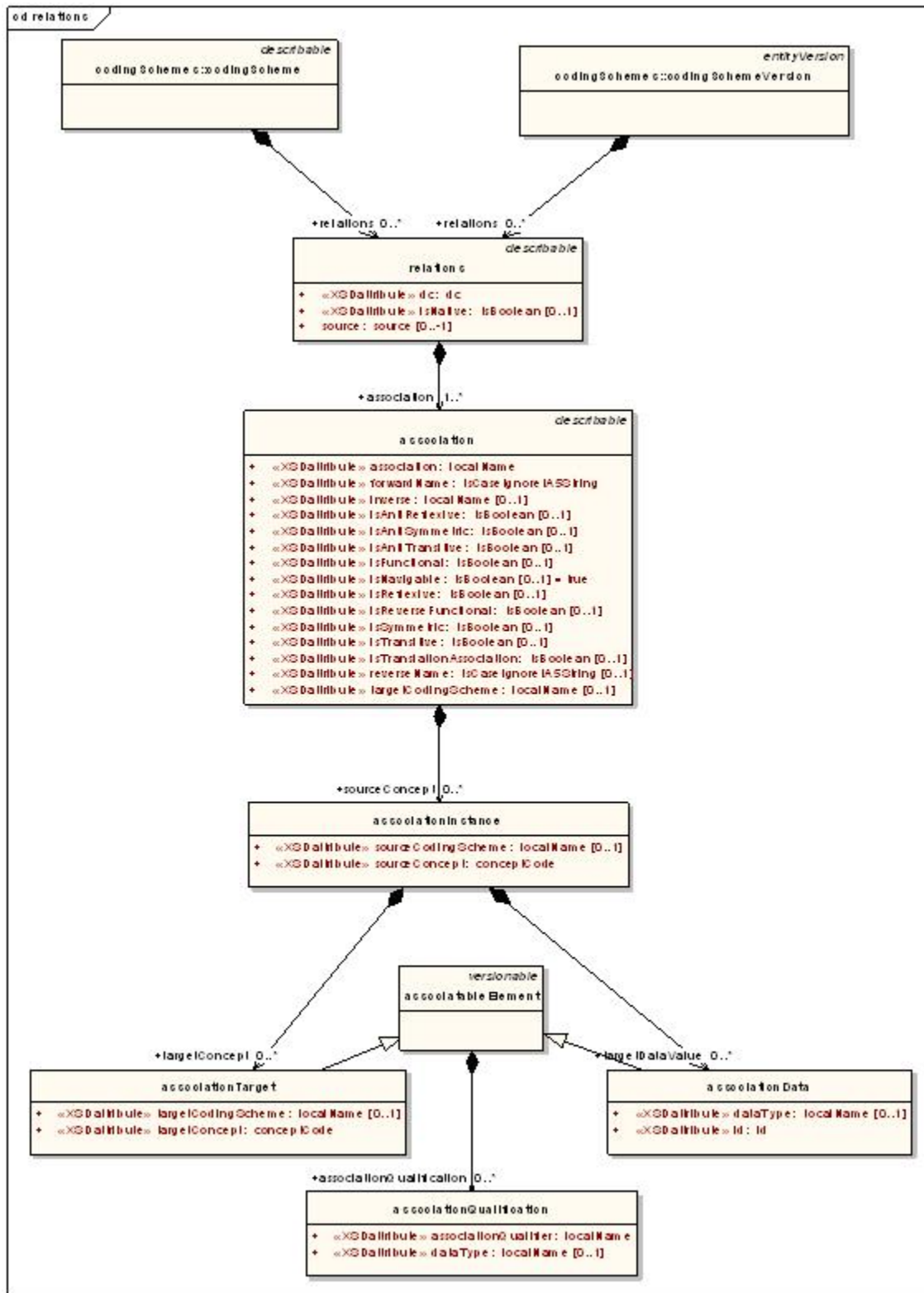
Components of interest are described in the following table:

Relation Component	Description
association	A relation between concept codes or concept codes and data. Association names must be included in the supportedAssociation metadata for the coding scheme.

associationInstance	An instance of a 'source' or left-hand side (LHS) of an association. An association instance references one or more 'targets' or right-hand sides (RHS).
associationTarget	An instance of a target or RHS concept of an association.
associationData	An instance of a target or RHS data value of an association.
associationQualific ation	A modifier that further qualifies an association triple.

Naming

These elements are primarily used to define metadata for a coding scheme, mapping locally used names to global references.
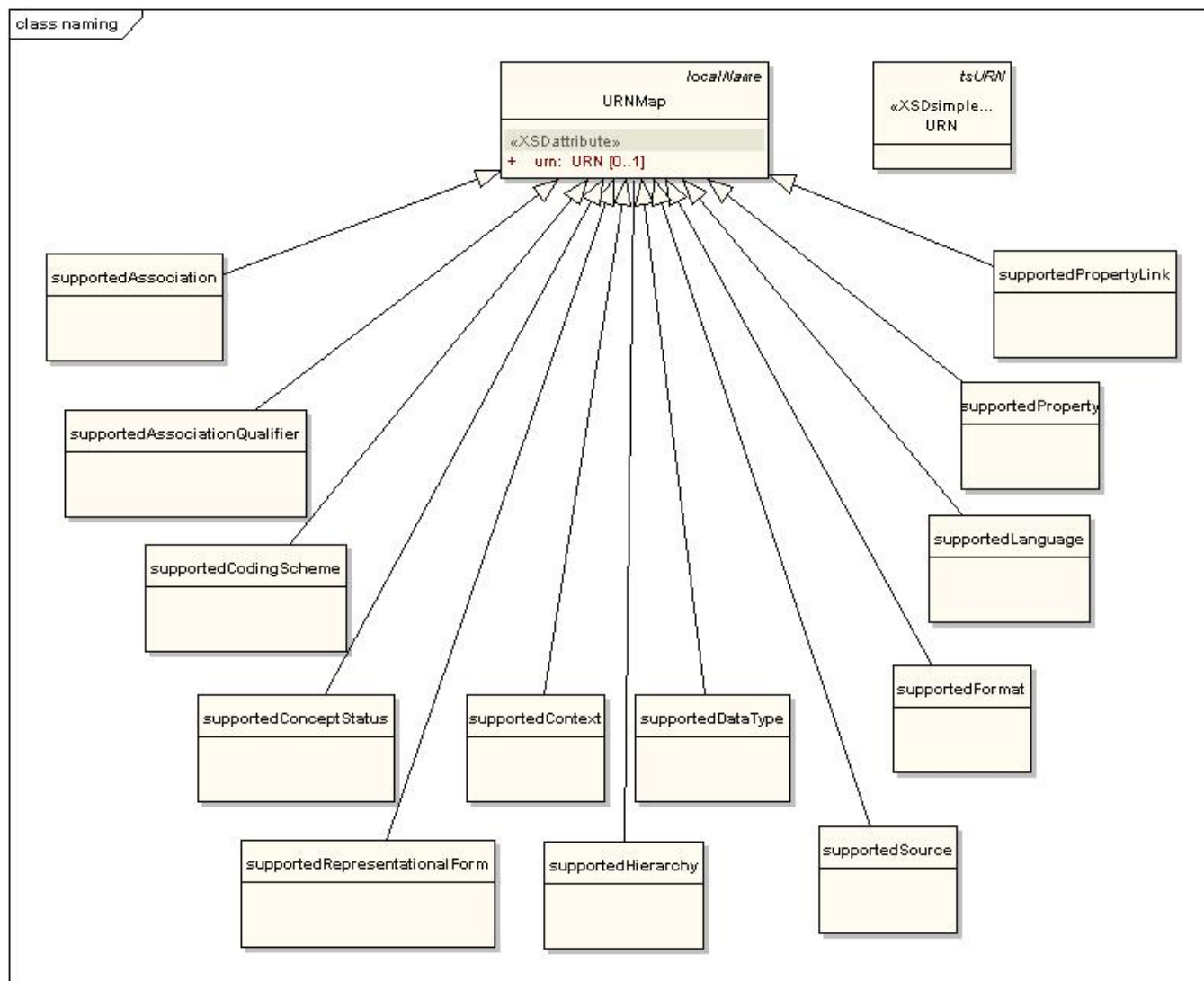
Components of interest are described in the following table:

Naming Component	Description
supportedAssociation	Each entry identifies the URN and local name for an association.
supportedAssociatio nQualifier	Each entry identifies the URN and local name for an association qualifier.
supportedCodingSc heme	Each entry identifies the URN and local name of an external coding scheme. The URN portion maps to the registeredName of the referenced scheme The local name is local to the referencing object.
supportedConceptS tatus	Each entry identifies the URN and local name for a concept status value.

supportedContext	Each entry identifies the URN and local name for a usage context.
supportedDataType	Each entry identifies the URN and local name for a data type (usually based in XML).
supportedFormat	Each entry identifies the URN and local name for a presentation format (usually mime type).
supportedHierarchy	Each entry identifies the URN, id, association, root node, and whether or not the hierarchy is forward navigable.
supportedLanguage	Each entry identifies the URN and local name of a spoken or written language.
supportedProperty	Each entry identifies the URN and local name of a property.
supportedPropertyLink	Each entry identifies the URN and local name for a lexical association between two concept properties (e.g. 'abbreviationFor', 'acronymFor').
supportedRepresentationalForm	Each entry identifies the URN and local name of a representational form (e.g. noun, eponym).
supportedSource	Each entry identifies the URN and local name of an external source reference.
URN	A universal resource name, representing the globally unique name of a resource such as a source, coding scheme, concept code, etc.
URNMap	The declaration of a local name and the URN that it represents. The behavior of an omitted URN is context specific.

LexBIG Model Extensions

The following extensions to the LexGrid model were introduced in support of caBIG® requirements. As with the LexGrid model, this document provides a summary of the most significant elements for consideration by LexBIG programmers. The complete and current version of the model is available online at LexBig Model and Schema.

Core

LexBIG core elements provide enhanced referencing and controlled resolution of LexGrid model objects.
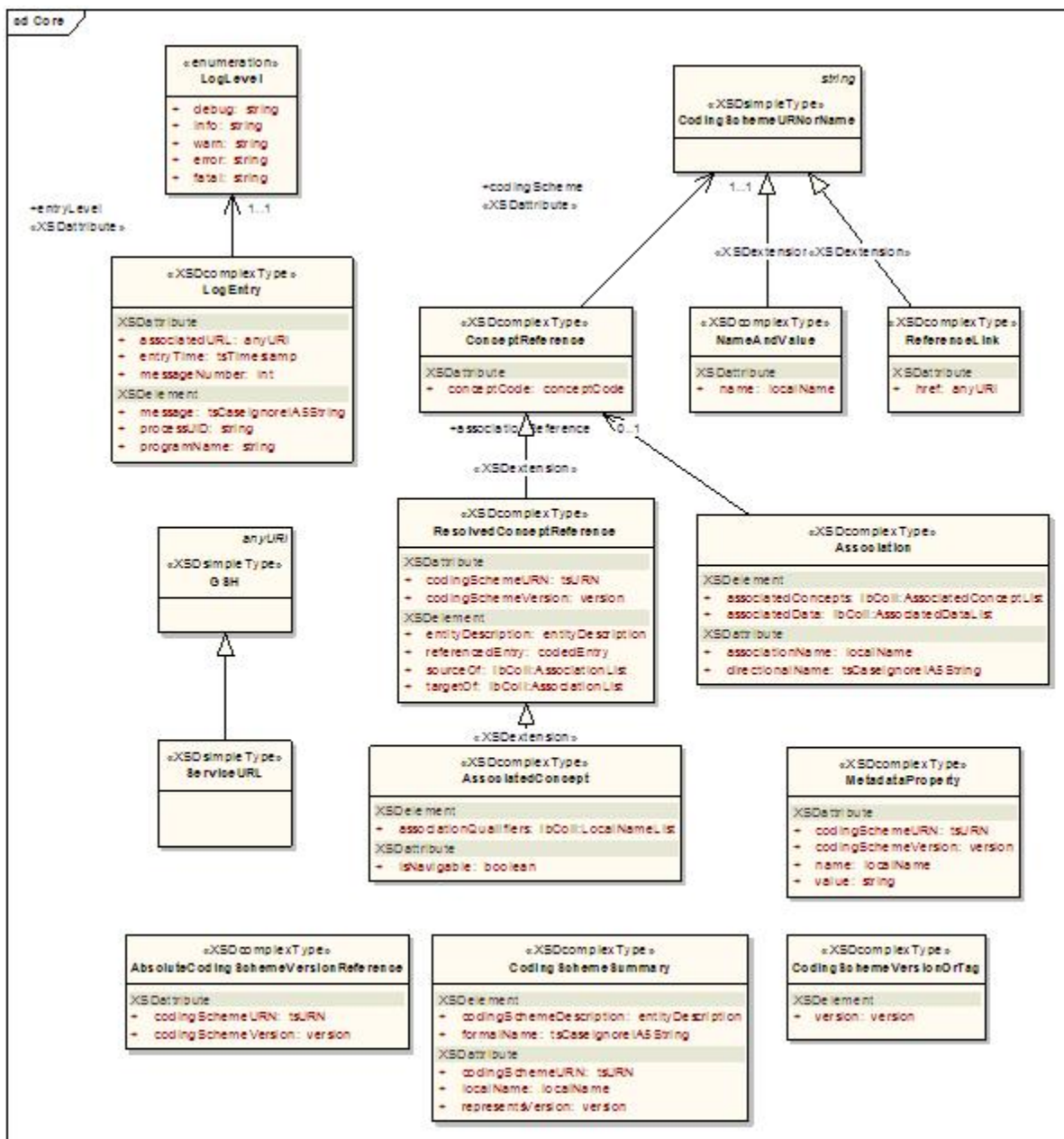
Components of interest are described in the following table:

Core Component	Description
AbsoluteCodingSchemeVersionReference	An absolute reference to a coding scheme. This form of reference is service independent, as it doesn't depend on local coding schemes names or virtual tags.
AssociatedConcept	A concept reference that is the source or target of an association.
Association	The representation of a particular association as it appears in a CodedNode.
CodingSchemeSummary	Abbreviated list of information about a coding scheme.
CodingSchemeURNorName	Either a local name or the URN of a coding scheme. These two are differentiated syntactically - if the entity includes a colon ":" or a hash "#" it is assumed to be a URN. Otherwise it is assumed to be a local name.
CodingSchemeVersionOrTag	A named coding scheme version or a virtual tag (e.g. latest, production, etc). Note that the tagged form of identifier is only applicable in the context of a given service, as one service may identify the scheme as "production" and another as "staging".
ConceptReference	A reference to a coding scheme and a concept code.

LogEntry	A single recorded log entry.
LogLevel	Indicates severity of the log entry.
MetadataProperty	Reference to a property name and value stored in the coding scheme metadata.
NameAndValue	A simple name/value pair.
ReferenceLink	Any reference to another document element. Used by the REST architecture to embed links.
ResolvedConceptReference	A resolvable concept reference.
ServiceURL	References a service in the Globus environment, this will be a global service handle (GSH).

InterfaceElements

Defines metadata related to model objects required by the runtime.

cd InterfaceElements

«XSDsimpleType» ProcessState (string)

«XSDcomplexType» ModuleDescription
XSDelement
+ description: string
XSDattribute
+ name: string
+ version: string

+state «XSDattribute» 1..1

«XSDextension»

«XSDcomplexType» ProcessStatus
XSDattribute
+ endTime: tsTimestamp
+ errorsLogged: tsBoolean
+ message: tsCaseIgnoreIA5String
+ startTime: tsTimestamp
+ warningsLogged: tsBoolean

«XSDcomplexType» ExtensionDescription
XSDelement
+ extensionBaseClass: string
+ extensionClass: string
XSDattribute
+ extensionProvider: sour

«enumeration» SortContext
+ graph: string
+ set: string
+ setIteration: string

+restrictToContext 0..*

«XSDextension»   «XSDextension»   «XSDextension»

«XSDcomplexType» LoadStatus
XSDelement
+ loadSource: anyURI
XSDattribute
+ numConceptsLoaded: int
+ numRelationsLoaded: int

«XSDcomplexType» ExportStatus
XSDelement
+ destination: anyURI

«XSDcomplexType» SortDescription

«XSDcomplexType» SortOption
XSDattribute
+ ascending: boolean
+ extensionName: string

«XSDcomplexType» RenderingDetail
XSDattribute
+ deactivateDate: tsTimestamp
+ lastUpdateTime: tsTimestamp
+ versionStatus: CodingSchemeVersionStatus
XSDelement
+ versionTags: CodingSchemeTagList

«XSDcomplexType» SystemReleaseDetail
XSDelement
+ entityVersions: entityVersion

+renderingDetail 1..1

«XSDcomplexType» CodingSchemeRendering
XSDelement
+ codingSchemeSummary: CodingSchemeSummary
+ referenceLink: ReferenceLink
+ serviceHandle: ReferenceLink

Components of interest are described in the following table:

Component	Description
CodingSchemeRendering	Information about a coding scheme as it appears in a particular service.
ExportStatus	Reports the state of LexBIG export operations.
ExtensionDescription	Describes an add-on module registered to the LexBIG environment.

LoadStatus	Reports the state of LexBIG load operations.
ModuleDescription	Describes a LexBIG integrated software module.
ProcessState	Enumerates possible status reported for LexBIG runtime operations.
ProcessStatus	Reports the state of LexBIG runtime operations.
RenderingDetail	The details of how a coding scheme is rendered in a given service.
SortContext	Describes a LexBIG sort module.
SortDescription	A description of a LexBIG extension module.
SortOption	Represents a pairing of sort algorithm and order.
SystemReleaseDetail	The combination of a system release and all of the entityVersions that accompanied that release.

NCIHistory

Maintains a record of modifications made to a code system.

Components of interest are described in the following table:

Component	Description
changeType	Atomic modification actions. Currently populated from a combination of Concordia, SNOMED-CT list and NCI's action list.
NCIChangeEvent	A change event as documented in ftp://ftp1.nci.nih.gov/pub/cacore/EVS/ReadMe_history.txt. Note that date and time of the change event is recorded in the containing version. All change events for the same/date and time a recorded in the same version.

LexBIG APIs

The section, Information Models describes the general format and organization of information handled by the LexBIG runtime. This section describes the primary application programming interfaces used to take action (e.g. retrieve or administer) against that content.

> ⓘ **Note**
>
> The information below is provided for introductory purposes. A full description of all available classes and methods is also available in the javadoc distributed with the LexBIG installation package (see file breakdown in Overview of the Software). Since the javadoc is automatically generated and synchronized during the build process, it is recommended as the primary reference for use by LexBIG developers.

Overview

Programming interfaces for the system fall into three primary categories:

- **Core Services** - Includes the LexBIGService, LexBIGServiceManager, CodedNodeSet and CodedNodeGraph classes, which provide the initial entry points for programmatic access to all system features and data.

- **Service Extensions** - The extension mechanism provides for pluggable system features. Current extension points allow for the introduction of custom load and indexing mechanisms, unique query sort and filter mechanisms, and generic functional extensions which can be advertised for availability to client programs.

- **Utilities** - Utility classes, such as those implementing iterator support, are provided by the system to provide convenience and optimize the handling of resources accessed through the runtime.

Core Services

Provides central entry points for programmatic access to system features and data.

Components of interest are described in the following table:

Component	Description
CodedNodeGraph	A virtual graph where the edges represent associations and the nodes represent concept codes. A CodedNodeGraph describes a graph that can be combined with other graphs, queried or resolved into an actual graph rendering.
CodedNodeSet	A coded node set represents a flat list of coded entries.
LexBIGService	This interface represents the core interface to a LexBIG service.
LexBIGServiceManager	The service manager provides a single write and update access point for all of a service's content. The service manager allows new coding schemes to be validated and loaded, existing coding schemes to be retired and removed and the status of various coding schemes to be updated and changed.
LexBIGServiceMetadata	Interface to perform system-wide query over optionally loaded metadata for loaded code systems and providers.

Service Extensions

Provides registration and lookup for pluggable system features.

Components of interest are described in the following table:

Component	Description
ExtensionRegistry	Allows registration and lookup of implementers for extensible pieces of the LexBIG architecture.
Extendable	Marks a class as an extension to the LexBIG application programming interface. This allows for centralized registration, lookup, and access to defined functions.

Query Extensions

Query extensions provide the ability to further constrain or manage query results.
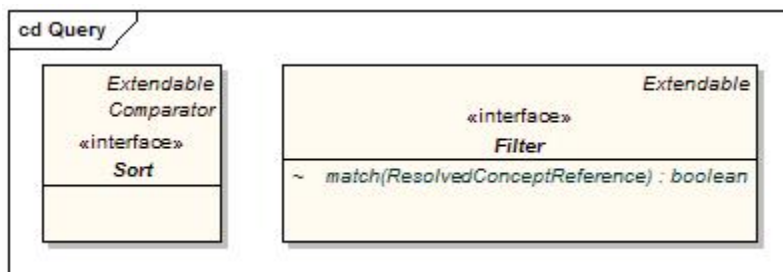
Components of interest are described in the following table:

Component	Description
Filter	Allows for additional filtering of query results.
Sort	Allows for unique sorting of query results. This interface provides a comparator to evaluate order of any two given items from the result set.

Load Extensions

Load extensions are responsible for the validation and import of content to the LexBIG repository. Vocabularies may be imported from a variety of formats including LexGrid canonical XML, NCI Thesaurus (OWL), and NCI MetaThesaurus (UMLS RRF).
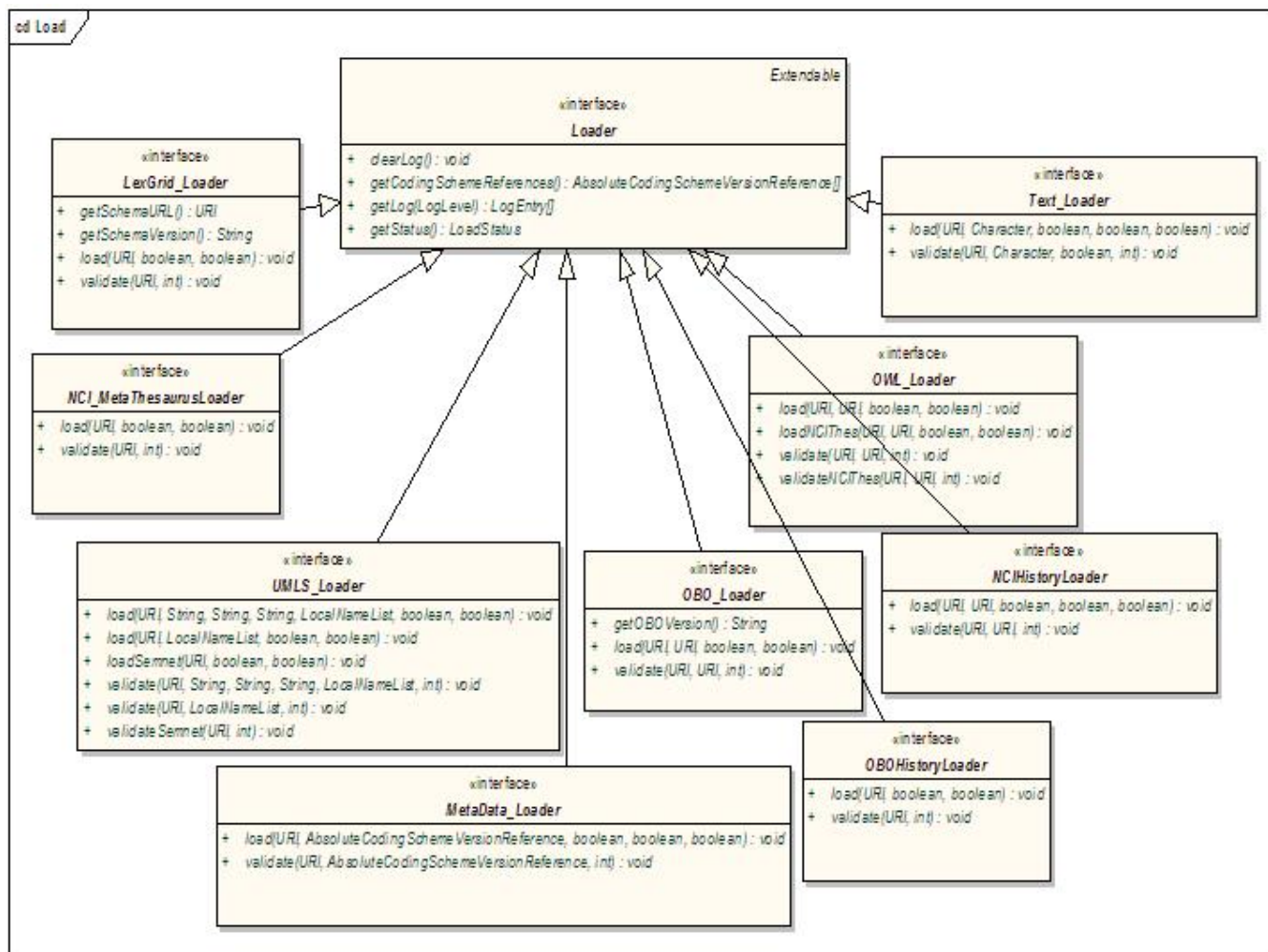
Components of interest are described in the following table:

Component	Description
Loader	The loader interface validates and/or loads content for a service.
LexGrid_Loader	Validates and/or loads content provided in the LexGrid canonical XML format.
NCI_MetaThesaurusLoader	Validates and/or loads the complete NCI MetaThesaurus. Content is supplied in RRF format. Note: To load individual coding schemes, consider using the UMLS_Loader as an alternative.
OBO_Loader	Validates and/or loads content provided in Open Biomedical Ontologies (OBO) text format.
OWL_Loader	Validates and/or loads content provided in Web Ontology Language (OWL) XML format. Note that for LexBIG phase 1 this loader is designed to specifically handle the NCI Thesaurus as provided in OWL format.
Text_Loader	A loader for delimited text type files. Text files come in one of two formats: indented code/designation pair or indented code/designation/description triples.
UMLS_Loader	Load one or more coding schemes from UMLS RRF format stored in a SQL database.
MetaData_Loader	Validates and/or loads content provided in metadata xml format. The only requirement of the xml file is that it be a valid xml file.
NCIHistoryLoader	A loader that takes the delimited NCI history file and applies it to a coding scheme.
OBOHistoryLoader	Load an OBO change history file.

Export Extensions

Export extensions are responsible for the export of content from the LexBIG repository to other representative vocabulary formats.
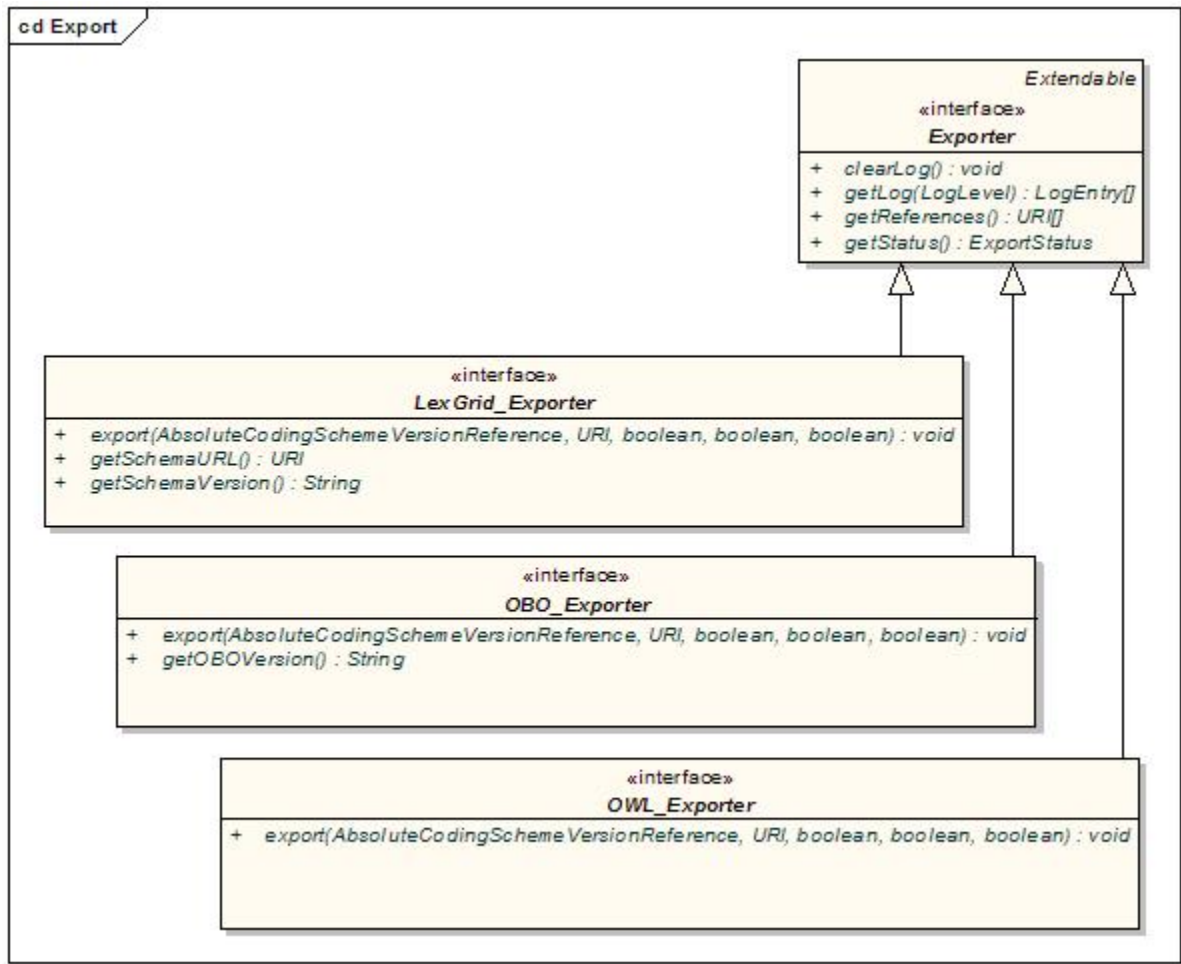
Components of interest are described in the following table:

Component	Description
Exporter	Defines a class of object used to export content from the underlying LexGrid repository to another repository or file format.
LexGrid_Exporter	Exports content to LexGrid canonical XML format.
OBO_Exporter	Exports content to OBO text format.
OWL_Exporter	Exports content to OWL XML format.

Index Extensions

Index extensions are built to optimize the finding, sorting and matching of query results.
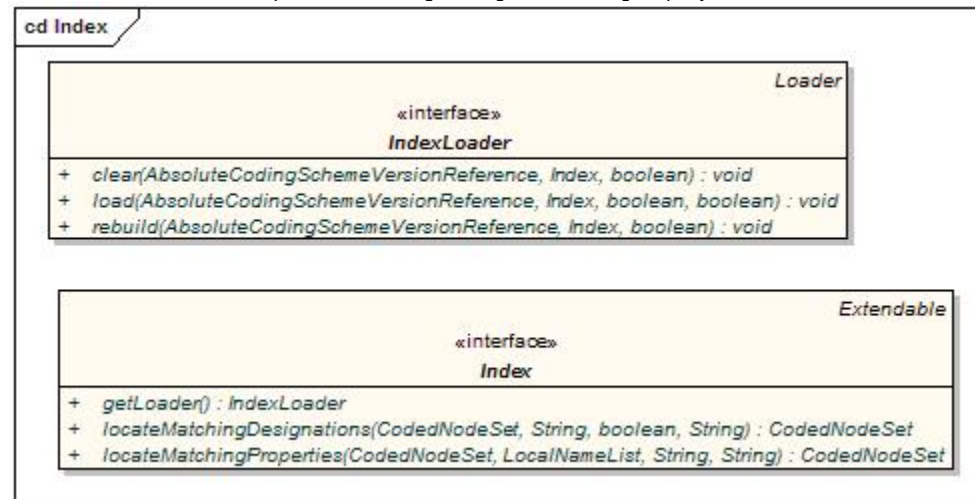
Components of interest are described in the following table:

Component	Description
Index	Identifies expected behavior and an associated loader to build and maintain a named index. Note that a single loader may be used to maintain multiple named indexes.
IndexLoader	Manages registered index extensions. A single loader may be used to create and maintain multiple indexes over one or more coding schemes. It is the responsibility of the loader to properly interpret each index it services by name, version, and provider.

Generic Extensions

Generic extensions provides a mechanism to register application-specific extensions for reference and reuse.
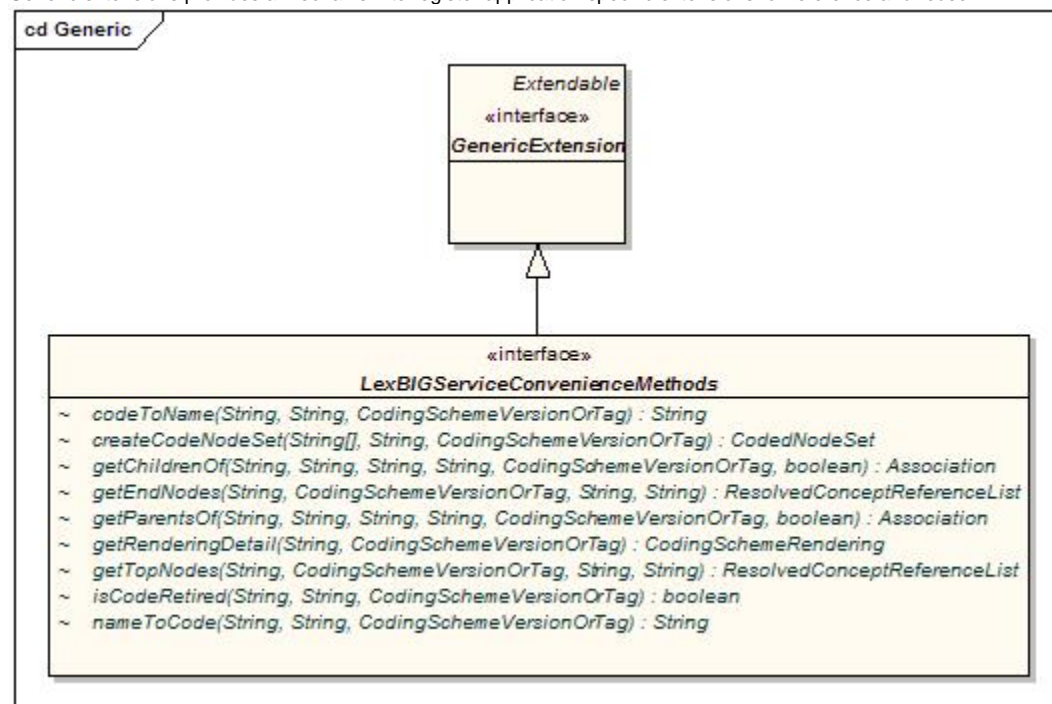
Components of interest are described in the following table:

Component	Description
GenericExtension	The generic extension class. Classes that implement this class are accessible via the LexBIGService interface.
LexBIGServiceConvenienceMethods	Convenience methods to be implemented as a generic extension of the LexBIG API.

Utilities

Defines helper classes externalized by the LexBIG API.

Iterators

Iterators are used to provide controlled resolution of query results.
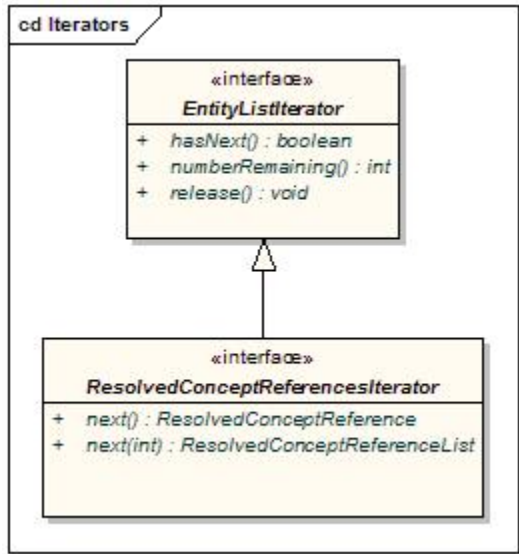
Components of interest include:

Component	Description
EntityListIterator	Generic interface for flexible resolution of LexBIG objects.
ResolvedConceptReferencesIterato	An iterator for retrieving resolved coding scheme references.

Additional Utility Classes

It is highly recommended that all LexBIG programmers familiarize themselves with the classes contained in the org.LexGrid.LexBIG.Utility package. Many useful features are provided in an effort to increase approachability of the API and assist the programmer in common tasks. This package currently contains the following classes:

- Constructors - Helper class to ease creating common objects.
- ConvenienceMethods - One-stop shopping for convenience methods that have been implemented against the LexBIG API.
- LBConstants - Provides constants for use in the LexBIG API.
- ObjectToString - Provides centralized formatting of LexBIG Objects to String representations.

Examples and Recommendations for Use

Concept Resolution

Programmers access coded concepts by acquiring first a node set or graph. After specifying optional restrictions, the nodes in this set or graph can be resolved as a list of `ConceptReference` objects which in turn contain references to one or more Concept objects. The following example provides a simple query of concept codes:

```java
// Create a basic service object for data retrieval
LexBIGService lbSvc = new LexBIGServiceImpl();

// Create a concept reference list appropriate for this coding scheme and
// this concept code where the parameters are a String array consisting of
// a single value and the name of the coding scheme where this concept
// resides.
ConceptReferenceList crefs =
    ConvenienceMethods.createConceptReferenceList(
        new String[] {code}, SAMPLE_SCHEME);

// Initialize a coding scheme version object with a version number for the
// sample scheme.
CodingSchemeVersionOrTag csvt = new CodingSchemeVersionOrTag();
csvt.setVersion(VERSION);

// Initialize A CodedNodeSet Object with all concepts in our sample coding
// scheme (We named the scheme we wanted and by using the Boolean value,
// false, retrieved both active and inactive concepts). This method call
// ignores the version tag using the null parameter. The final
// restrictToCodes(crefs) method call restricts the return to the single
// code in the previously initialized list of one.
CodedNodeSet nodes = lbSvc.getCodingSchemeConcepts(SAMPLE_SCHEME,
    csvt).restrictToCodes(crefs);
// Build a list of references from the current (and already restricted) set
// and restrict them further to the single property of NCI_NAME and
// restrict to a single answer (parameter 1)).
ResolvedConceptReferenceList matches = nodes.resolveToList(null,
    ConvenienceMethods.createLocalNameList("FULL_SYN"), 1);

// Does our list of one contain the single reference were were looking for?
// If so, then initialize a ResolvedConceptReference with the result, and
// initialize a Concept object by calling the getReferencedEntry()
// method. The Concept object is the base information model object and
// contains, among other things, the CONCEPT_NAME value we were seeking. We
// retrieve it with a call to the first element on the properties list,
// getting the text and it's accompanying content.
            if (matches.getResolvedConceptReferenceCount() > 0) {
            ResolvedConceptReference ref =
        (ResolvedConceptReference)matches
            .enumerateResolvedConceptReference().nextElement();
            Concept entry = ref.getReferencedEntry();
            System.out.println("Matching synonym: " +
                    entry.getPresentation(0).getText().getContent()) ;

            } else {
            System.out.println("No match found!");
        }
```

Service Metadata Retrieval

The LexBIG system maintains service metadata which can provide client programs with information about code system content and assigned copyright /licensing information. Below is an brief example showing how to access and print some of this metadata:

```
// We can get a CodingSchemeRenderingList object directly from the
// LexBigService.
LexBIGService lbs = new LexBIGServiceImpl();
CodingSchemeRenderingList schemeList = lbs.getSupportedCodingSchemes();

for (CodingSchemeRendering csr : schemeList.getCodingSchemeRendering()) {
    CodingSchemeSummary css = csr.getCodingSchemeSummary();

    // Print separator, then details from the CodingSchemeSummary
    System.out.println("=========================================");
    System.out.println(ObjectToString.toString(css));

    // Set up a coding scheme reference to resolve Copyright
    String urn = css.getCodingSchemeURN();
    String version = css.getRepresentsVersion();
    CodingSchemeVersionOrTag csVorT =
        Constructors.createCodingSchemeVersionOrTagFromVersion(version);
    CodingScheme cs = lbs.resolveCodingScheme(urn, csVorT);
    System.out.println("Copyright: " + cs.getCopyright().getContent());

    // Get the final details from the RenderingDetail
    RenderingDetail rd = csr.getRenderingDetail();
    System.out.println(ObjectToString.toString(rd));
    System.out.println();
}
```

Combinatorial Queries

One of the most powerful features of the LexBIG architecture is the ability to define multiple search and sort criteria without intermediate retrieval of data from the LexBIG service. Consider the following code snippet:

```
System.out.println("Example double restriction query with additional
  application of sort criteria and restricted return values");
//Declare the service ...
LexBIGService lbs = new LexBIGServiceImpl();

//Start with an unconstrained set of all codes for the vocabulary ...
CodingSchemeVersionOrTag csvt = new CodingSchemeVersionOrTag();
csvt.setVersion(VERSION);
CodedNodeSet cns = lbs.getCodingSchemeConcepts(SAMPLE_SCHEME,csvt);

//Constrain to concepts with designations (assigned text presentations)
//that contain text that sounds like 'heart ventricle'
cns.restrictToMatchingDesignations(
    "hart ventrikle",
    SearchDesignationOption.ALL,
    MatchAlgorithms.DoubleMetaphoneLuceneQuery.toString(),
    null);

//Further restrict the results to concepts with a semantic type of
//'Body Space or Junction'.
cns.restrictToMatchingProperties(
    Constructors.createLocalNameList("Semantic_Type"),
        "Anatomical Structure",
        "exactMatch",
    null);

//Indicate that the resulting list should be sorted,
//with best results first and then sorted by code if there is a tie.
SortOptionList sortCriteria =
    Constructors.createSortOptionList(new String[]{"matchToQuery", "code"});

//Indicate to return only the assigned UMLS_CUI and
//textualPresentation properties.
LocalNameList restrictTo =
    ConvenienceMethods.createLocalNameList(new String[]{"UMLS_CUI",
        "textualPresentation"});
//Still nothing computed yet!
//Perform the query and resolve the sorted/filtered list,
//with a maximum of 6 items returned ...
ResolvedConceptReferenceList list = cns.resolveToList(sortCriteria,
restrictTo,
    null, 6);
//Print the results ...
ResolvedConceptReference[] rcr = list.getResolvedConceptReference();
for(ResolvedConceptReference rc : rcr){
        System.out.println("Resolved Concept: " +
```

This example shows a simple yet powerful query to search a code system based on a 'sounds like' match algorithm (the list of all available match algorithms can be listed using the *'ListExtensions -m'* admin script).

Declaring the target concept space

The coded node set (variable 'cns') is initially declared to query the NCI Thesaurus vocabulary. At this point the concept space included by the set can be thought of as unrestricted, addressing every defined coded entry (the 'false' value on the declaration indicates to also include inactive concepts). However, it important to note that no search is performed by the LexBIG service at this time.

Applying filter criteria

Similarly, no computation is performed (to realize query results) during invocation of the <tt>restrictToMatchingDesignations()</tt> and <tt>restrictToMatchingProperties()</tt> methods. However, these calls effectively narrow the target space even further, indicating that filters should be applied to the information returned by the LexBIG query service.

Using the Lucene Query Syntax and other text matching functions

The text criteria applied in methods such as `restrictToMatchingDesignations()` uses one of a number of powerful text processing applications to provide the user with broad capability for text based searches. Text matches can be simple applications of exactMatch, startsWith or contains algorithms as well as powerful regular expressions and Lucene Query syntax (used in the LuceneQuery function.) As shown above these options are passed into the `restrictToMatchingDesignations()` Method as parameters.

Lucene Queries are well documented and can be very powerful. The uninitiated user may need some background on their use however. The user should start here with the official Lucene Query Parser documentation.

Keep in mind that some LexBIG queries such as "startsWith" and "contains" use wild card searches under the covers, so that use of wild cards in this context can cause errors in searches involving these search types.

Instead it is best to use the flexibility of the Lucene Query searches in the matchingDesignation by using the Lucene Query searches in LexBIG where most searches will work much as described in the query syntax documentation.

Special characters in the Lucene Query search can cause unexpected results. If you are not using special characters as recommended for various Lucene search mechanisms then your searches may not return expected results or may return an error. If the value you are searching upon contains say, parenthesis, you will need to place the value in quotations. The escape characters described in the Lucene Documentation do not work at this time.

Likewise you should not expect to see a Lucene Query narrow down search results as you progressively enter a longer substring more closely matching your term of interest. Instead use the contains method.

Applying sorting criteria

Multiple sort algorithms can be applied to control the order of items returned. In this case, we indicate that results are to be sorted based on primary and secondary criteria. The "matchToQuery" algorithm indicates to sort the result according to *best* match as determined by the search engine. The "code" item indicates to perform a secondary sort based on concept code.

> ⓘ **Note**
>
> The list of all available sort algorithms can be listed using the *'ListExtensions -s'* admin script.

Restricting the information returned for matching items

The LexBIG API also allows the programmer to restrict the values returned for each matching concept. In this example, we chose to return only the UMLS CUI and assigned text presentations.

Retrieving the result

A query is finally performed during the 'resolve' step, with results returned to the declared list. It is at this point that the LexBIG service does the heavy lifting. By declaring the full extent of the request up front (namespace, match criteria, sort criteria, and returned values), the service then has the opportunity to optimize the query path. In addition, in this example we restrict the number of items returned to a maximum of 6. This combined approach has the benefit of reducing server-side processing while minimizing the volume and frequency of traffic between the client program and the LexBIG service.

> ⓘ **Note**
>
> While this section provides one example of combining criteria, this same pattern can be applied to many of the <tt>CodedNodeSet</tt> and <tt>CodedNodeGraph</tt> operations. It is strongly recommended that programmers familiarize themselves with this programming model and its application.

Additional Resources

The examples and automated test programs provided by the LexBIG installation (see file breakdown in the section, Overview of the Software) are available as additional reference materials.

Exercising the API - The LexBIG GUI

The LexBIG Graphical User Interface, or GUI, is an optional component of the LexBIG install which will be in the /gui folder of the base LexBIG installation (see file breakdown in the section, Overview of the Software). The GUI is meant to provide a simple tool to test LexBIG API methods and quickly view the results; almost all public methods defined by the LexBIG API are supported. This guide provides a brief overview of how the GUI can aid programmers in writing code to the LexBIG API.

> ⓘ **Note**
>
> The LexBIG GUI supports both administrative and test functions. Please refer to the *LexBIG Administrator's Guide* for instructions on using the GUI as an administration tool.

Launching the GUI

Depending on the operating systems that you selected at installation time, you should have one or more of the following programs in the `/gui` folder:

```
Linux_64-lbGUI.sh            Linux-lbGUI.sh
OSX-lbGUI.command            Windows-lbGUI.bat
```

Launch the GUI by executing the appropriate script for your platform. You will be presented with an application that looks like this:

Overview

The upper section of the GUI shows all of the code systems currently loaded, along with corresponding metadata. The lower section of the GUI is used to combine, restrict and resolve Code Sets and Code Graphs.

The lower left section is where you can perform Boolean logic on Code Sets and Code Graphs. The lower right section is where you can introduce restrictions on Code Sets and Code Graphs and browse results.

> ⓘ **Note**
>
> The menu options are used primarily for administrative functions, and are covered in detail by the *LexBIG Administrator's Guide*. In addition, all of the disabled buttons in the top half of the application are used for administrative functions, and are also described in the *LexBIG Administrator's Guide*.

Creating New Queries

There are four buttons on the top half that are of interest for creating queries.

- **Refresh** - This button causes the LexBIG GUI to reread the available terminologies and their respective metadata. This can be useful when using the GUI to view a LexBIG environment that is being modified by another process.
- **Get History** - If a terminology with available history data is selected, this button opens a history browser to view it via the NCI history API. This option is currently only applicable when working with the NCI Thesaurus terminology.
- **Get Code Set** - This button causes the selected terminology to be added to the lower left section of the GUI as a code set - which is noted by a 'CS' prefix.
- **Get Code Graph** - This button causes the selected terminology to be added to the lower left section of the GUI as a code graph - which is noted by a 'CG' prefix.

Customizing Queries

After selecting a code system and clicking on **Get Code Set** or **Get Code Graph**, a row will be added to the lower left section of the GUI for each click. There are seven buttons in the lower left section that allow combinatorial logic between the code sets in the lower left.

- **Union** - This button is enabled if two Code Sets or two Code Graphs are selected in the lower left. Clicking the button creates a new virtual Code Set or Code Graph which represents the Boolean union of the two selected items. All restrictions applied to the individual items still apply.
- **Intersection** - This button is enabled if two Code Sets or two Code Graphs are selected in the lower left. Clicking the button creates a new virtual Code Set or Code Graph which represents the Boolean intersection of the two selected items. All restrictions applied to the individual items still apply.
- **Difference** - This button is enabled if two Code Sets or two Code Graphs are selected in the lower left. Clicking the button creates a new virtual Code Set which represents the Boolean difference of the two selected Code Sets. All restrictions applied to the individual items still apply.
- **Restrict to Codes** - This button is enabled if a Code Set and a Code Graph are selected in the lower left. Clicking the button creates a new virtual Code Graph which will be restricted to concept codes occurring in the selected Code Set.
- **Restrict to Source Codes** - This button is enabled if a Code Set and a Code Graph are selected in the lower left. Clicking the button creates a new virtual Code Graph which will have its source codes restricted to codes occurring in the selected Code Set.
- **Restrict to Target Codes** - This button is enabled if a Code Set and a Code Graph are selected in the lower left. Clicking the button creates a new virtual Code Graph which will have its target codes restricted to codes occurring in the selected Code Set.
- **Remove** - This button is enabled if any Code Set or Code Graph (or virtual Code Set or Code Graph) is selected in the lower left. Clicking the button will remove the selected item from the list.

The lower right section of the GUI is used to apply restrictions to Code Sets or Code Graphs, and set the variables that need to be passed into the resolve method.

Working with Code Sets

If a Code Set is selected in the lower left, then the lower right section will look like this:

In the lower right section, there are two halves - the top half and the bottom half. The top half is used to apply restrictions. The bottom half provides query options and resolution.

- **Add** - This button introduces a new restriction to the Coded Node Set. Clicking it will bring up the following dialog box for creating restrictions:
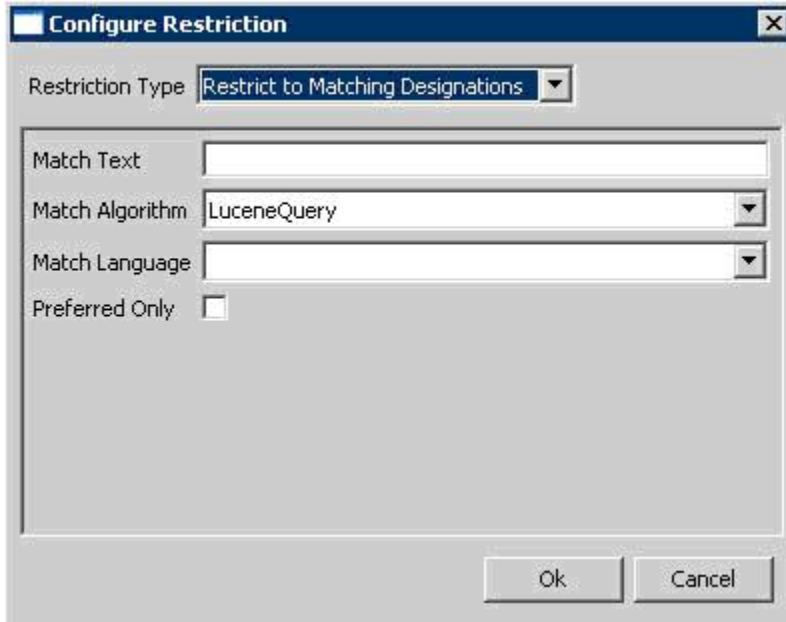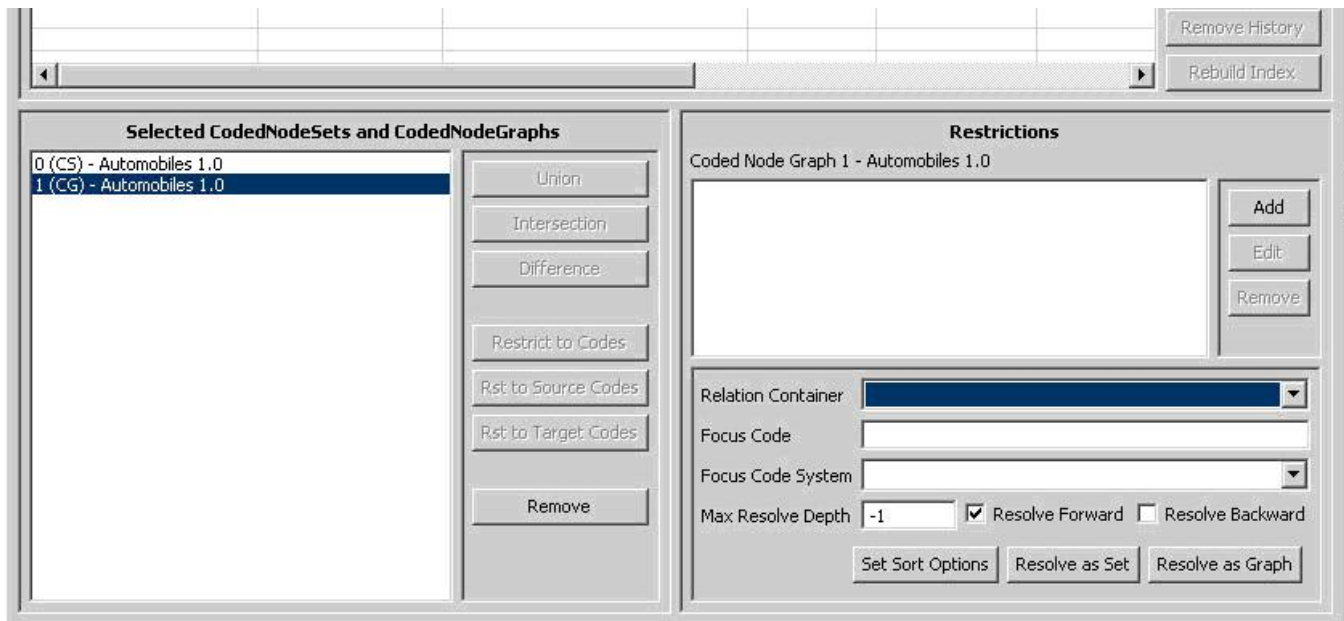
The top drop down list indicates the type of restriction to add. The rest of the dialog box will change depending on the type of restriction selected. All required parameters for the selected restriction type will be presented.
- **Edit** - This button is enabled when a restriction is selected. Clicking it allows revision of an existing restriction.
- **Remove** -This button is enabled when a restriction is selected. Clicking it removes the selected restriction.
- **Only Include Active Codes** - This check box indicates whether or not to include inactive codes when resolving the selected code set.
- **Set Sort Options** - This button will bring up a dialog box to choose the desired sort order of the results.
- **Resolve Code Set** - This button will bring up a result window where the Code Set will be resolved and displayed.

Working with Code Graphs

If you select a Coded Node Graph in the lower left section of the LexBIG GUI, the lower right section will look like this:

Again, there are two halves to the lower right section. The top half allows restrictions to be applied to the selected Code Graph, and it works the same as it does for a Coded Node Set. Please see the section above on applying restrictions to a Coded Node Set.

The lower half provides additional variables applicable when resolving a Coded Node Graph. For further explanation of these options, refer to the LexBIG API documentation.

- **Relation Container** (Optional) - Indicates the CodingScheme Relations container to query. The drop down list is populated with allowable selections.
- **Focus Code** (Optional) - Provides the code used as a starting point when resolving graph relations. This value is required for some queries, depending on the nature of requested associations.
- **Focus Code System** (Optional) - Indicates the code system containing the Focus Code. The drop down list is populated with allowable selections.

- **Max Resolve Depth** - How many levels deep should the graph be resolved? -1 is the default, which does not limit the depth.
- **Resolve Forward** - Populate codes downstream from the focus node (based on directionality defined by each association).
- **Resolve Backward** - Populate codes upstream from the focus node (based on directionality defined by each association).
- **Set Sort Options** - This button will bring up a dialog box to choose the desired sort order of the results.
- **Resolve As Set** - Resolves and displays the graph results as a coded node set.
- **Resolve As Graph** -Resolves and displays the graph results.

Viewing Query Results

Clicking on the Resolve buttons for either a Coded Node Set or a Coded Node Graph will bring up the Result Browser window:
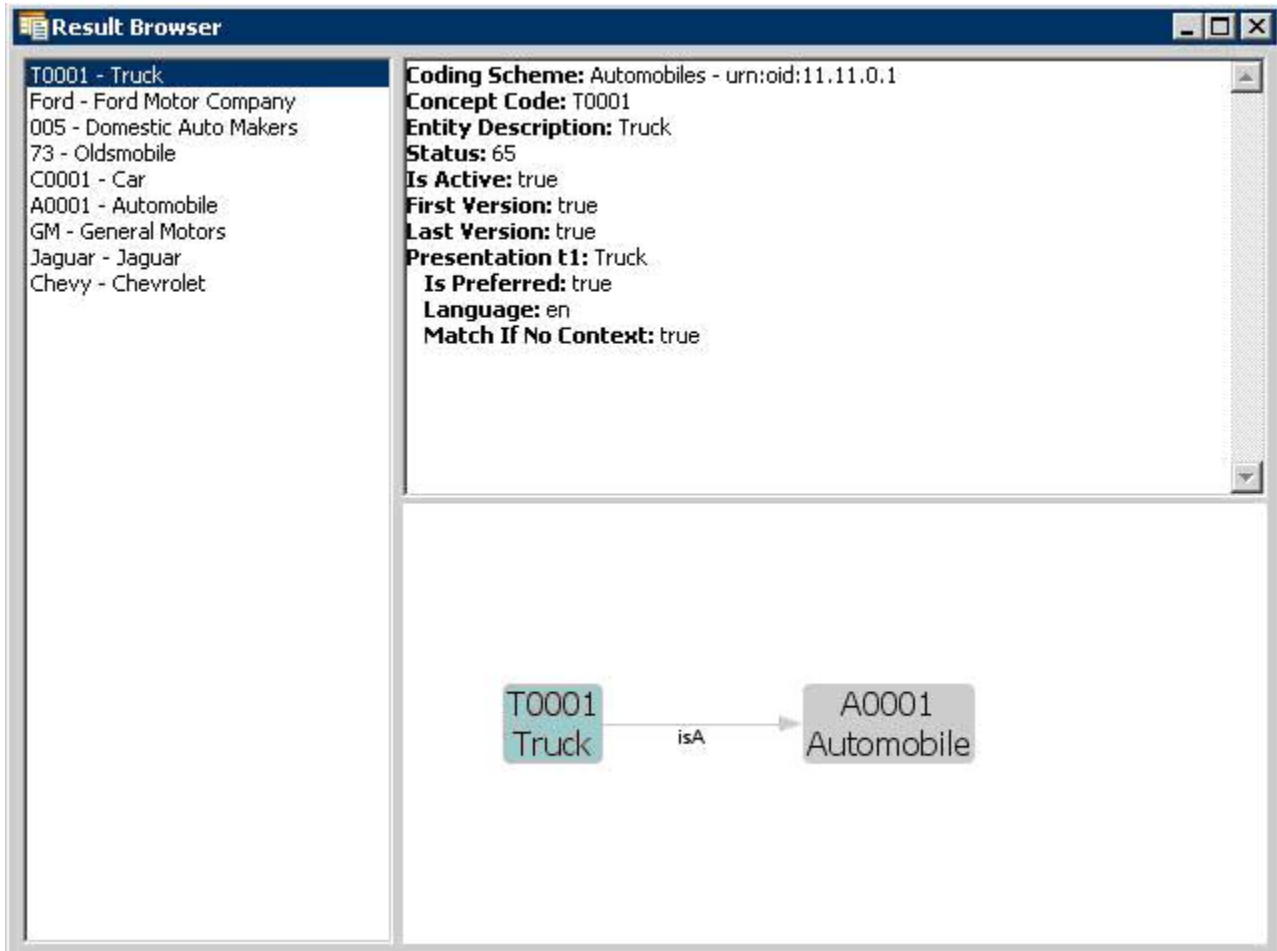
The left side shows a list of all the concept codes returned. When a concept code is selected on the left, the upper right will show a full description of the selected code. The lower right will show a graph view of the neighboring concepts.

When a Coded Node Graph is resolved, the result viewing window will look like this (this is the same Code System as above):
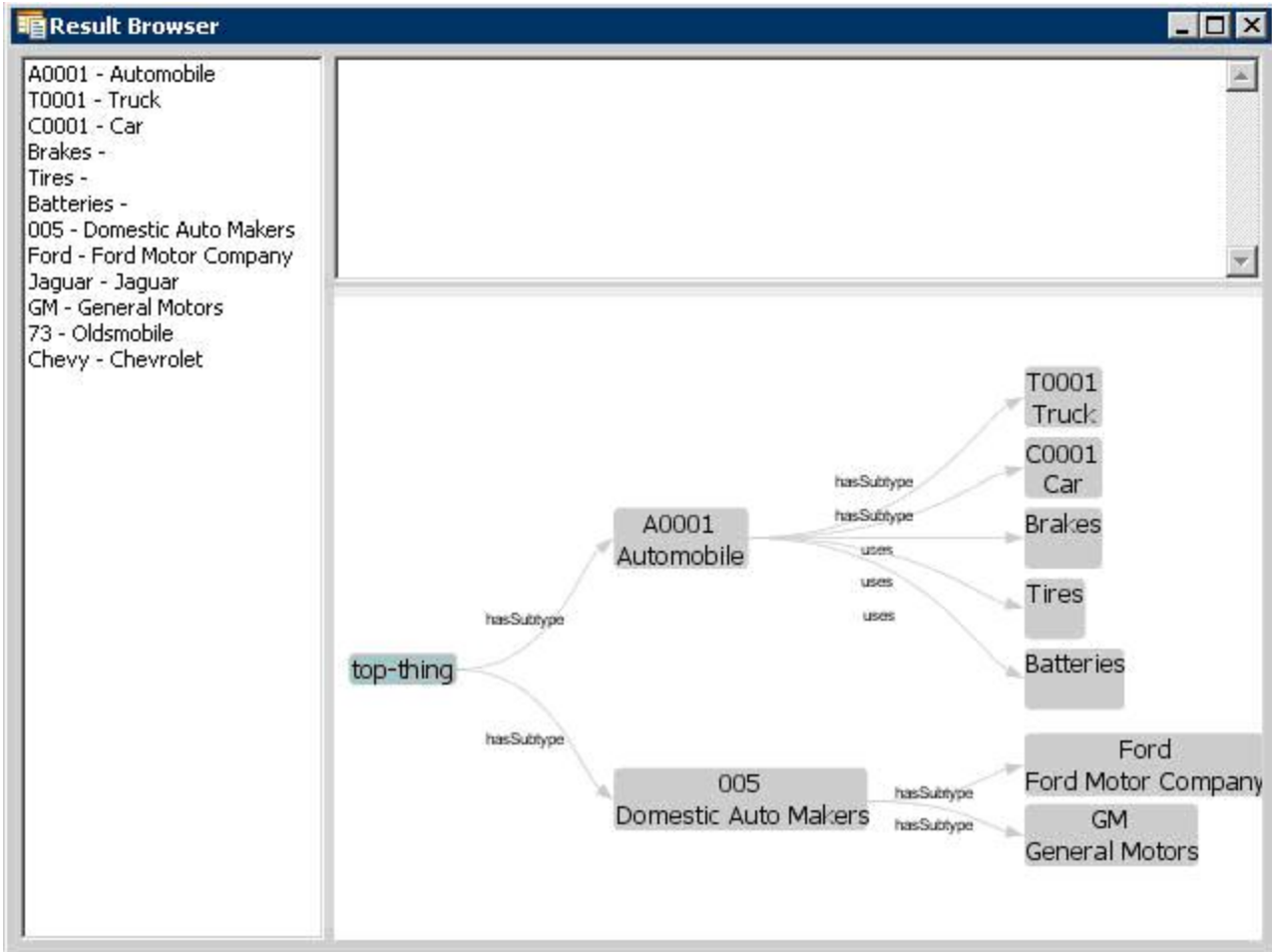
The left side still has a list of all of the concepts in the graph. The upper right will give a description of the selected concept. The lower right shows the entire graph.

The lower right section is adjustable, and dynamic. It responds to mouse clicks, dragging, and numerous key combinations. Beyond a depth of 3, the graph may "collapse" and not show all of the nodes until you click on a node. Clicking on a node will cause it to expand out and display its children. Here are a list of key combinations recognized by the graph viewer:

- Left Click + Mouse Movement - Drags the view.
- Right Click + Mouse Movement Up Or Down - Zooms in or out.
- Right Click (on white space) - Zooms the view to fit.
- Ctrl + '+' - Expands the graph connection lines
- Ctrl + '-' - Contracts the graph connection lines
- Ctrl + '1' (or '2' or '3' or '4') - Changes the orientation of the graph.

Appendix A References

This appendix includes lists and hypertext links, where appropriate, to technical manuals, articles, scientific publications and online resources related to the LexBIG project.

- LexBIG GForge project Docs archive, Files archive
- LexBIG Project Administration Materials
- LexGrid Home Page for this release
- Vocabulary Knowledge Center
- Sun Java Tools (JDK, JRE, NetBeans) 🖅
- Eclipse Project (IDE) 🖅

Appendix B Included Materials

Components

The following Java archives are distributed with the LexBIG runtime environment and may be of interest to programmers:

Module	Function

|	
`lbRuntime`	The LexBIG runtime code, including all necessary code and dependencies required for direct Java-to-Java invocation of the LexBIG API. This consolidates LexBIG code and 3rd party modules in order to simplify configuration for program execution.

> ⓘ **Note**
>
> This archive is not available for redistribution simply because individual contributions are not easily separated and the combined content cannot be shared under a single license. It is provided strictly as a convenience for simplified program execution and in accordance with the user having agreed to all terms and conditions during product installation.

Redistributable components (e.g. those listed below) and associated license terms are also made available. The redistributable components provide equivalent content and function, but require more extensive configuration for program execution.

`lexbig`	This archive includes the LexBIG runtime code, excluding all dependencies.
`lbGUI`	The LexBIG graphical user interface runtime code, excluding all dependencies.
activation ⬈	Provided by Sun's reference implementation of the JavaBeans Activation Framework (JAF) ⬈ standard extension. Used for e-mail notification when runtime errors occur.
caGrid	Grid infrastructure to support the caBIG® community. Contains tools for creating and deploying caBIG®-compliant grid services.
commons-cli ⬈	Provides a simple API for working with command line arguments, options, option groups, mandatory options and so forth.
commons-codec ⬈	Provides implementations of common encoders and decoders such as Base64, Hex, Phonetic and URLs.
commons-collections ⬈	Provides a suite of classes that extend or augment the Java Collections Framework.
commons-lang ⬈	Provides a very common set of utility classes that provide extra functionality for classes in the `java.lang` package.
commons-logging ⬈	Provides a bridge between different logging libraries.
commons-pool ⬈	Provides a generic object pooling interface, a toolkit for creating modular object pools and several general purpose pool implementations.
gnu-regexp ⬈	Provides a Java language implementation of standard NFA regular expression features.
hsqldb ⬈	SQL relational database engine written in Java.
icu4j ⬈	International components for Unicode processing.
jakarta-regexp ⬈	Java package for processing regular expressions.
jcalendar ⬈	Java date chooser bean for graphically picking a date.
jdom ⬈	Java-based solution for accessing, manipulating, and outputting XML data from Java code.
jena ⬈	Java framework for building Semantic Web ⬈ applications.
junit ⬈	Java regression test framework.
log4j ⬈	Runtime logging services.
lucene-core ⬈	Text search engine library written in Java. Provides support for regular expression-based queries. Provides stemming support for indexed concepts.
mail ⬈	Provided by the Sun JavaMail API ⬈ . Used for e-mail notification when runtime errors occur.
mm.mysql (drivers, 2.0.6) ⬈	JDBC drivers for MySQL database.
org.eclipse. ⬈ *	Used internally by LexBIG load and export extensions to access and manipulate Eclipse Modeling Framework (EMF) ⬈ model representations.
postgresql (drivers) ⬈	JDBC drivers for PostgreSQL database.
prefuse ⬈	Used for graph representations in the LexBIG GUI.
swt ⬈ (**swt** .jar)	Provides the underlying widget toolkit used by the LexBIG GUI.
xerces ⬈	XML parsing services.

Appendix C Additional Terms and Conditions

Refer to the license.pdf and license.txt files (installed to the LexBIG root directory) for the license terms and conditions of included components.