LexGrid Background Information

Contents of this Page

The Lexical Grid

Currently, there are many terminologies and ontological resources available (ICD-9, NCI Thesaurus, SNOMED-CT). However, many of these resources involve incompatible formats, tooling, and programming interfaces. This can make it difficult to use these resources to their full potential. The Lexical Grid (LexGrid) project is an initiative of the Mayo Clinic Division of Biomedical Informatics (BMI) designed to bridge this gap using common tools, data formats, and read/update mechanisms. Resources on the Lexical Grid are intended to be:

- represented using a single information model accessible through a set of common application programming interfaces (APIs)
- joined through shared indices accessible online
- downloadable
- loosely coupled
- locally extendable
- globally revised
- available in web-space on web-time
- cross-linked

The realization of this vision requires three interlocking components:

1. Standards: access methods (programming APIs) and formats need to be published and openly available
2. Tools: standards-based tools must be readily available
3. Content: commonly used vocabularies and ontologies have to be available for access and download

In short, LexGrid provides the standardized building blocks and tools to take advantage of vocabulary and ontology content where and when needed, thereby providing the infrastructure necessary to support large-scale terminology adoption and use. Additional information for LexGrid is available on the LexGrid page.

LexGrid Features

- Accommodation of multiple vocabulary and ontology distribution formats
- Support of multiple data stores to accommodate federated vocabulary distribution
- Consistent and standardized access across multiple vocabularies
- Rich API for supporting lexical and graph search and traversal
- Fully compatible with HL7-CTS implementation
- Support for programmatic access via Java, .NET, and web services
- Open source tooling and code to facilitate adoption and use

LexGrid Users

LexGrid is intended to meet the needs of the following groups:

- Vocabulary service providers - Describes organizations currently supporting externalized API-level interfaces to vocabulary content
- Vocabulary integrators - Describes organizations that desire to integrate new vocabulary content or relations to be served locally
- Vocabulary users - Describes persons and organizations desiring common, consistent access to vocabulary content for a supporting multiple application development uses

LexGrid Principles

LexGrid software is based on a model-driven architecture. The LexGrid model (further described below) is maintained in XML-Schema format and represents a core component of design. The LexBIG API, a Java-based API to LexGrid content (also described below) is formally modeled and accommodates registration of additional load, index, and search functions. The LexBIG API also provides a conscious separation of service and data classes in order to support deferred query resolution and software iterators.

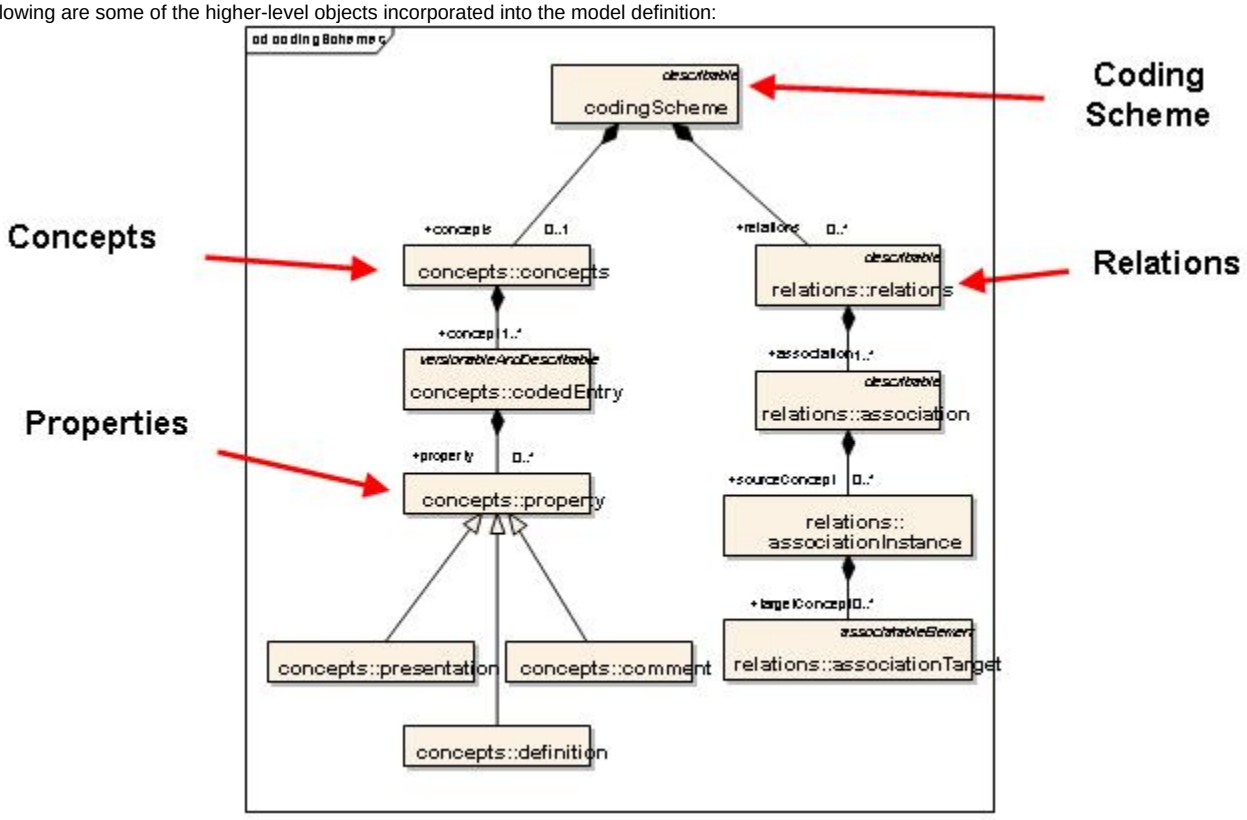
LexGrid Model

The LexGrid Model is Mayo'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, e.g., relational database, LDAP) and a XML format. This model provides the core representation for all data managed and retrieved through the LexBIG system, and is now rich enough to represent vocabularies provided in numerous source formats including:

- Open Biomedical Ontologies (OBO)
- Web Ontology Language (OWL), e.g., NCI Thesaurus
- Unified Medical Language System (UMLS) Rich Release Format (RRF), e.g., NCI MetaThesaurus

This common model is a critical component of the LexGrid project. Once disparate vocabulary information can be represented in a standardized model, it becomes possible to build common repositories to store vocabulary content and common programming interfaces and tools to access and manipulate that content. The HL7 Common Terminology Services (CTS) and LexBIG API as developed for the Cancer Biomedical Informatics Grid (caBIG®) initiative are two examples of APIs able to query information stored in the LexGrid Model.

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 codedEntry) within a particular domain of discourse. Each concept is unique within the code system that defines it. To be valid, a codedEntry must be qualified by at least one term or 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 codedEntry. A concept may be the source for 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 relation's 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).

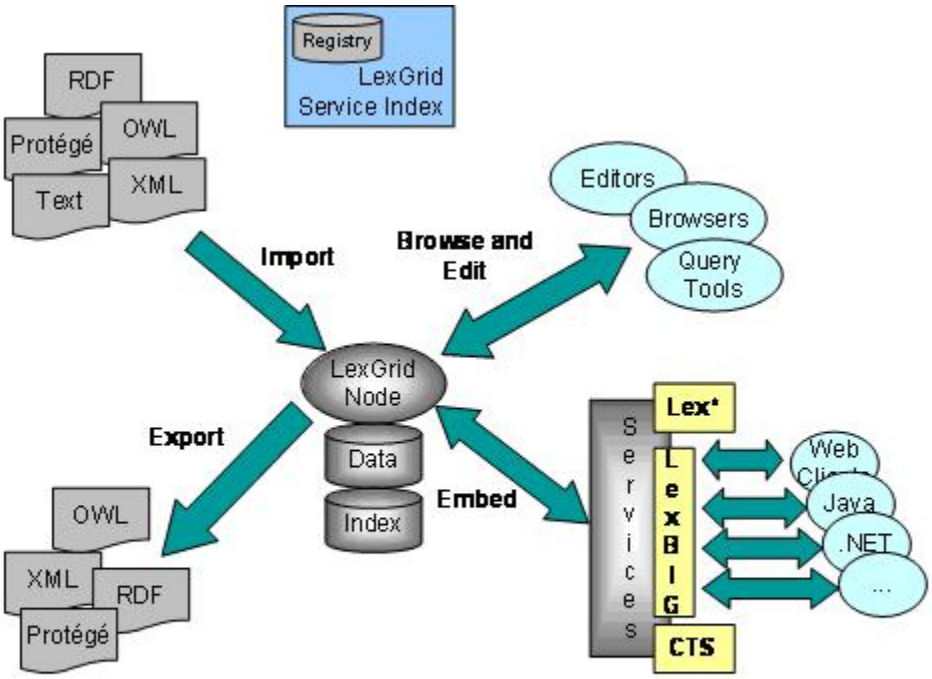
Available Representations

The master representation of the LexGrid Model is provided in XML Schema Definition (XSD) format. Conversions to other formal representations are available, including XML Metadata Interchange (XMI) and Unified Modeling Language (UML).

Implementation or technology-specific renderings of the model also exist. These include relational database schema (MySQL, PostgreSQL, DB2, Oracle, etc.) and Lightweight Directory Access Protocol (LDAP) schema. Programming interfaces generated from the formal representation include Java bean interfaces based on the Eclipse Modeling Framework (EMF) and Castor frameworks.

LexGrid Components

This section provides a basic description of the building blocks, interfaces, and tools comprising the LexGrid project.



LexGrid Node

A **LexGrid Node** represents both software and a backing data store that provide a logical persistence layer for accessing, storing, and managing vocabulary content according to a formalized information model (described above). LexGrid nodes typically utilize relational database management systems for management of data and indexing functions. Implementations include but are not limited to MySQL, PostgreSQL, UDB/DB2, Oracle, Hypersonic, and LDAP/BDB.

The *LexGrid Services provide an access layer to search and access vocabulary content. Specific APIs have been implemented to access LexGrid content. Health Level Seven Common Terminology Services specification (HL7-CTS) is an HL7 standard. LexBIG is a rich API created to support the use cases and requirements to support the National Cancer Institute and Cancer Biomedical Informatics Grid (caBIG®). Lex* represent other APIs to access vocabulary content. The set of LexGrid APIs can be implemented using standard technologies like Web Services (SOAP), Java, and .Net.

The *LexGrid Registry Service provides a mechanism to locate available vocabularies that are stored on one or more LexGrid nodes.

LexGrid Projects

LexGrid technologies are integrated and play an essential role in many vocabulary and ontology projects, meeting the needs of both the Mayo Clinic and external organizations. Some projects include the following:

- LexCTS - HL7 Common Terminology Services reference implementation
- LexBIG - LexGrid Vocabulary Services for caBIG®
- LexBIO - LexGrid Vocabulary Services for NCBO
- Mayo Clinic Life Sciences vocabulary services
- Mayo Surgical Index Retrieval Services ICD-9 codebook

LexBIG Project

Of these projects, perhaps the most significant and influential to LexGrid architecture, has been the LexBIG (LexGrid Vocabulary Services for caBIG®) project, which in turn provides the core infrastructure for the LexBIO and LexPHIN projects.

What is LexBIG?

LexBIG is a 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 API capable of accessing and distributing vocabularies as commodity resources. Primary objectives for the project include:

- Provide a robust and scalable open-source implementation of EVS-compliant vocabulary services. The API specification is based on but not limited to fulfillment of the caCORE EVS API. The specification also accommodates 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).

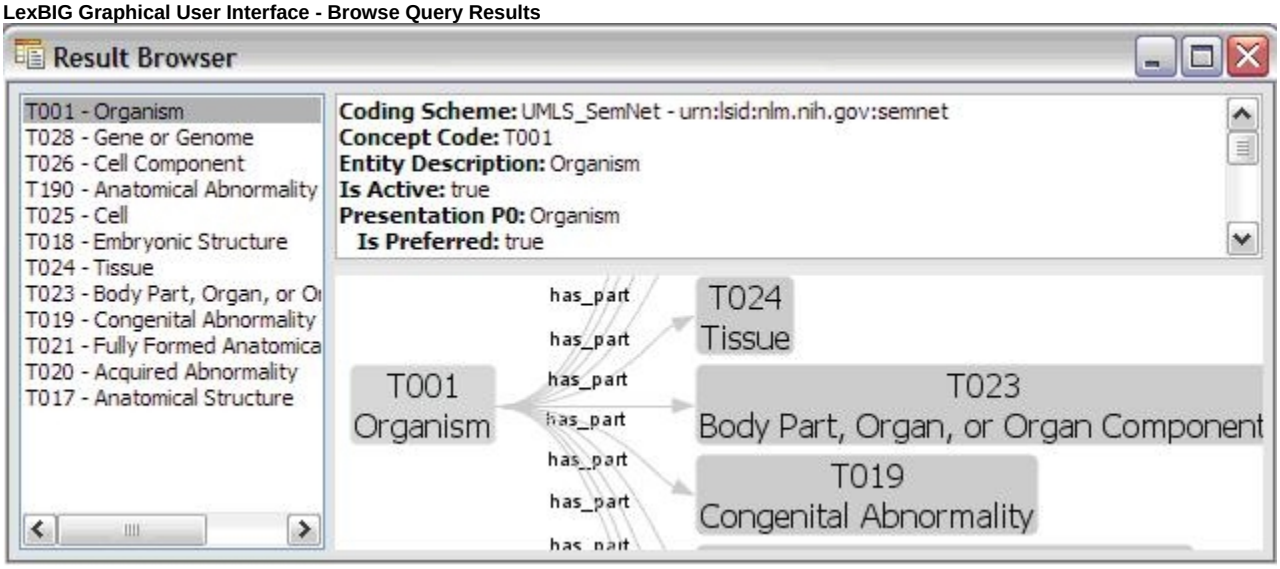
The LexBIG package represents a compressive set of software and services to load, publish, and access vocabulary. Cancer Centers can use the LexBIG package to install the NCI Thesaurus and NCI MetaThesaurus and query content via a rich API. LexBIG services can also be incorporated into numerous applications wherever vocabulary content is needed.

LexBIG Components

Major components within the software package include the following:

- Service Management programs to load, index, publish, and manage vocabulary content for the vocabulary server
- Application Programming Interface (API) providing Java interfaces to various functions including Lexical queries, Graph representation and traversal, and NCI change event history
- Graphical User Interface providing access to administrative and API functions
- Documentation consisting of JavaDocs, Administrator, and Programmer Guides
- Examples providing sample source code for common vocabulary queries
- Test Suite to validate the LexBIG installation

LexBIG Graphical User Interface - Browse Query Results



LexBIG API

This section provides a brief introduction to the LexBIG application programming interface.

Programs access coded concepts within a vocabulary or ontology by first acquiring 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. Each concept reference identifies a concept within the vocabulary or ontology.

One of the most powerful features of the LexBIG API 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...
CodedNodeSet cns = lbs.getCodingSchemeConcepts("NCI_Thesaurus", null, false);

//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
//'Anatomical Structure'.
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, 6);

//Print the results ...
ResolvedConceptReference[] rcr = list.getResolvedConceptReference();
for(ResolvedConceptReference rc : rcr)
      System.out.println("Resolved Concept: " + ObjectToString.toString(rc));
```

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, distributed with the LexBIG software package).

Note that while this section provides one example of combining criteria, this same pattern can be applied to many of the CodedNodeSet and CodedNodeGraph operations. Steps include the following:

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 is 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 restrictToMatchingDesignations() and restrictToMatchingProperties() 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.

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 six. 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: Additional information regarding API options and invocation is provided by the LexBIG Programmer's Guide (provided with the software distribution).