1 - LexEVS 6.x API

Contents of this Page

LexEVS 6.x Programmers Links

- Programmer's Guide Main Page
  - LexEVS API
  - LexEVS 6.0 CTS2 API
  - LexEVS 6.x CTS2 API Quick Start
- Value Set and Pick List Guide
- LexEVS 6.0 Main Page
- LexEVS Current Release

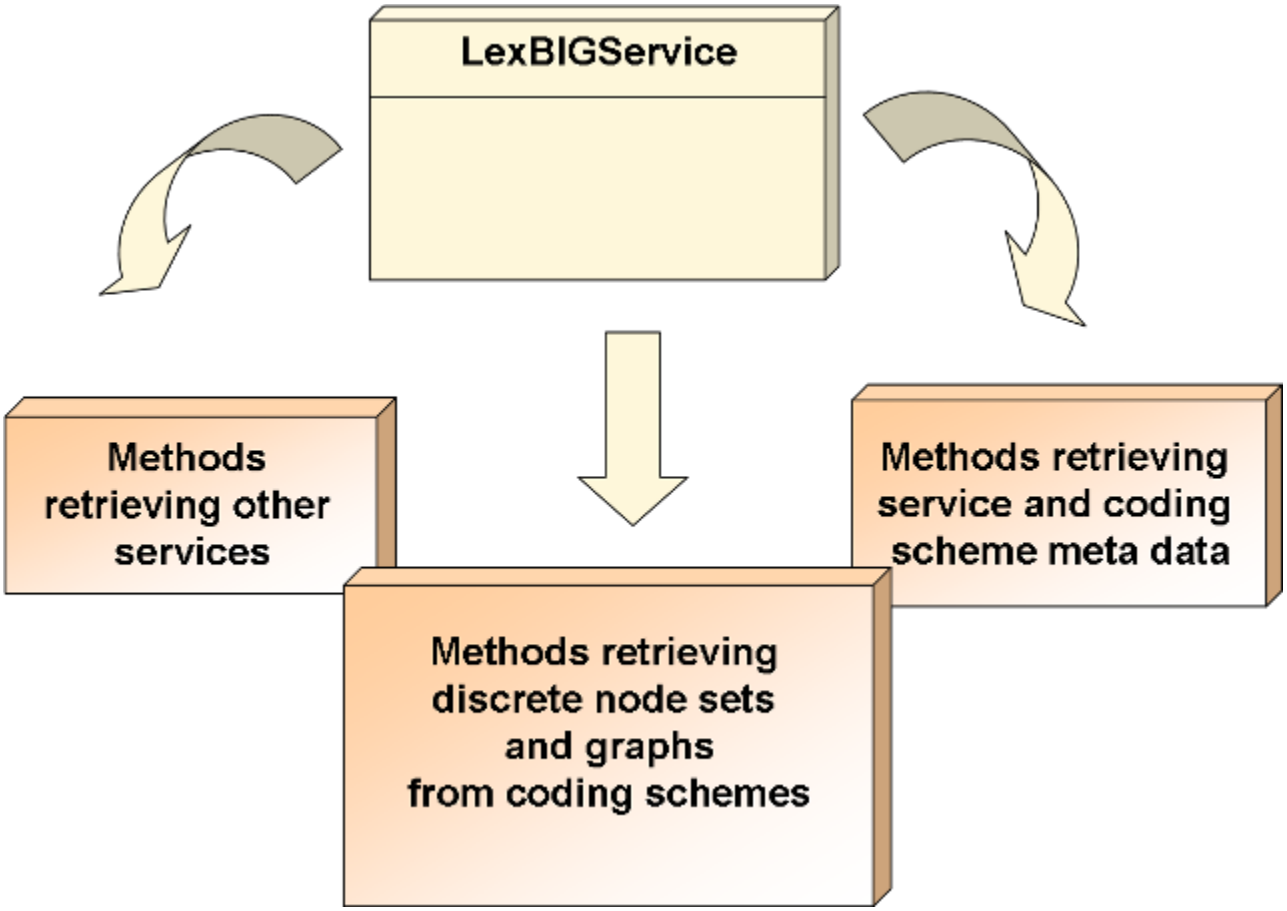
Introduction

The LexEVS APIs fall into three primary categories:

- **Core Services** - Core services include 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

The **LexBIGService** provides central entry points for programmatic access to system features and data. In the following figure, LexBIGService provides Methods retrieving other services, Methods retrieving discrete node sets and graphs from coding schemes, and Methods retrieving service and coding sheme meta data.



LexEVS Services and Components

Interface Link	Description
LexBIGService	This interface represents the core interface to a LexEVS service and acts as an entry point to other API components and services
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.
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.
LexEVSAuthoringService	Authoring service interface for creating Mappings and Coding Scheme relationships
Value Set, Asserted Value Set, and Pick List Services	The LexBIGService interface is not an entry point for Value Sets and Pick Lists. For details, see the Value Set and Pick List Services section of this guide.

JavaDocs for the LexEVS Service Interfaces

Service Extensions

Provides registration and lookup for pluggable system features.

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

Javadocs for LexEVS Extensions

Query Extensions

Query extensions provide the ability to further constrain or manage query results. For details on the LexEVS 6.0 Query Extension, see the document section Query Services Extension.

Interface Link	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.
Search	Allows for unique search of query results.

Javadocs for LexEVS Query Extensions

Load Extensions

Load extensions are responsible for the validation and import of content to the LexEVS repository. Vocabularies may be imported from a variety of formats including LexGrid canonical XML, NCI Thesaurus (OWL), and NCI MetaThesaurus (UMLS RRF). For details on LexEVS loaders and the Loader Framework, see the Loader Guide.

The following are the components of interest:

Interface Link	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 Metathesaurus Loader	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.
OWL2 Loader	Validates and/or loads content provided by the latest OWL standard and as interpreted by the OWL API.
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.
MrMapLoader	Load mappings between coding schemes from UMLS formatted MRMAP.RRF and MRSAT.RRF files.
ClaML Loader	Loads representations sourced in the Classification Markup Language such as ICD-10 (No longer supported)
Radlex Protege Frames Loader	Loads the Radlex terminology from a Protege Frames formatted source. (No longer supported)

Javadocs for LexEVS Load Extensions

Loader Construction

While not specifically developed as an API loader interfaces for the LexEVS API exist new loaders are regularly written. Some unsupported, community-contributed loaders have been created such as those for RXNORM and NDFRT content.

We provide instructions for creating loaders of your own and include a framework for loaders that can be written using Spring Batch.

LexEVS 6.x Loader Implementation

Loader Frame Work

Export Extensions

Export extensions are responsible for the export of content from the LexEVS repository to other representative vocabulary formats.

Interface Link	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.

Javadocs for LexEVS Export Extensions

Index Extensions

Index extensions are built to optimize the finding, sorting and matching of query results. It is the responsibility of the loader to properly interpret each index it services by name, version, and provider.

Interface Link	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.

Javadocs for LexEVS Index Exensions

Generic Extensions

Generic extensions provides a mechanism to register application-specific extensions for reference and reuse.

Interface Link	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 LexEVS API.
MappingExtension	A grouping of Mapping Coding Scheme related functionality.
SupplementExtension	A grouping of Coding Scheme Supplement related functionality.

Javadocs for LexEVS Generic Extensions

Utilities

Defines helper classes externalized by the LexEVS API.

Convenience Classes

> **Note**
>
> It is highly recommended that all LexEVS 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:

Interface Link	Description
Constructors	Helper class to ease creating common objects.
ConvenienceMethods	One-stop shopping for convenience methods that have been implemented against the LexEVS API.
LBConstants	Provides constants for use in the LexEVS API.
ObjectToString	Provides centralized formatting of LexEVS Objects to String representations.
ServiceUtility	Provides utility methods for the LexEVS services.
VSObjectToString	Provides centralized formatting of LexEVS Value Set Object to String representations.

Javadocs for LexEVS Utility Classes

Iterators

Iterators are used to provide controlled resolution of query results.

Interface Link	Description
EntityListIterator	Generic interface for flexible resolution of LexEVS objects
ResolvedConceptReferencesIterator	An iterator for retrieving resolved coding scheme references.

Javadocs for LexEVS Iterator classes

Search Algorithms - Supported LexEVS Search Algorithms

Lucene Based Algorithms

See the Lucene Query Parser documentation for more information on these Lucene query expressions.

```
Name: LuceneQuery
Version: 1.0
Description: Search with the Lucene query syntax.
```

```
Name: DoubleMetaphoneLuceneQuery

Version: 1.0
Description: Search with the Lucene query syntax, using a 'sounds like' algorithm.
A search for 'atack' will get a hit on 'attack'
```

```
Name: StemmedLuceneQuery

Version: 1.0
Description: Search with the Lucene query syntax, using stemmed terms.
A search for 'trees' will get a hit on 'tree'
```

```
Name: startsWith

Version: 1.0
Description: Equivalent to 'term*' (case insensitive)
```

```
Name: exactMatch

Version: 1.0
Description: Exact match (case insensitive)
```

```
Name: contains

Version: 1.0
Description: Equivalent to '* term* *' - in other words - a trailing wildcard on a term
(but no leading wild card) and the term can appear at any position.
```

Apache Regular Expressions

```
Name: RegExp
Version: 1.0
Description: A Regular Expression query.  Searches against the lowercased text, so a
regular expression that specifies an uppercase character will never return a match.
Additionally, this searches against the entire string as a single token, rather than
the tokenized string - so write your regular expression accordingly.
Supported syntax is documented here:
http://jakarta.apache.org/regexp/apidocs/org/apache/regexp/RE.html
```

Custom Extensions

The following custom extensions include adaptations of Lucene searches with characters normally stripped by Lucene (Literals).

```
Name: phrase
Description: Searches for a Phrase in text.
Base Class: org.LexGrid.LexBIG.Extensions.Query.Search
Extension Class: org.LexGrid.LexBIG.Impl.Extensions.Search.PhraseSearch
Version: 1.0
```

```
Name: LeadingAndTrailingWildcard

Description: Equivalent to '*term*'.
Base Class: org.LexGrid.LexBIG.Extensions.Query.Search
Extension Class: org.LexGrid.LexBIG.Impl.Extensions.Search.LeadingAndTrailingWildcardSearch
Version: 1.0
```

```
Name: subString

Description: Search based on a "*some sub-string here*". Functions much like the Java String.indexOf method.
Base Class: org.LexGrid.LexBIG.Extensions.Query.Search
Extension Class: org.LexGrid.LexBIG.Impl.Extensions.Search.SubStringSearch
Version: 1.0
```

```
Name: SpellingErrorTolerantSubStringSearch

Description: Adds Spelling-error tolerance to 'subString' search.
Base Class: org.LexGrid.LexBIG.Extensions.Query.Search
Extension Class: org.LexGrid.LexBIG.Impl.Extensions.Search.SpellingErrorTolerantSubStringSearch
Version: 1.0
```

```
Name: literalContains

Description: Equivalent to '* term* *' - in other words - a trailing wildcard on a term (
but no leading wild card) and the term can appear at any position. Includes special characters.
Base Class: org.LexGrid.LexBIG.Extensions.Query.Search
Extension Class: org.LexGrid.LexBIG.Impl.Extensions.Search.LiteralContainsSearch
Version: 1.0
```

```
Name: nonLeadingWildcardLiteralSubString

Description: Search based on a "*some sub-string here*". Functions much like the
Java String.indexOf method. Singe term searches will match '*term' and 'term*' butnot '*term*'.
This is because leading wildcards are very inefficient.  Special characters are included.
Base Class: org.LexGrid.LexBIG.Extensions.Query.Search
Extension Class: org.LexGrid.LexBIG.Impl.Extensions.Search.NonLeadingWildcardLiteralSubStringSearch
Version: 1.0
```

```
Name: literal

Description: All special characters are taken literally.
Base Class: org.LexGrid.LexBIG.Extensions.Query.Search
Extension Class: org.LexGrid.LexBIG.Impl.Extensions.Search.LiteralSearch
Version: 1.0
```

Code Examples

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 Entity objects. The following example provides a simple query of concept codes:

Java Code Snippet

```
// Create a basic service object for data retrieval
LexBIGServiceImpl lbs = LexBIGServiceImpl.defaultInstance();

// Create a list of unique references (concept codes) for this coding scheme.
// Parameters:
//       A String array initialized with a single concept code
//       The name of the target Coding Scheme.
ConceptReferenceList crefs = ConvenienceMethods.createConceptReferenceList(
                new String[], SAMPLE_SCHEME);

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

// Initialize a CodedNodeSet Object with all possible concepts in our sample coding
// scheme, then restrict the node set to a single node using restrictToCodes(crefs)
CodedNodeSet nodes = lbs.getCodingSchemeConcepts(SAMPLE_SCHEME, csvt).
                restrictToCodes(crefs);

// Build a potential list of references from the current (and already restricted) set
// and restrict them to the single property name "textualPresentation" and
// allow the list a size of 1.
ResolvedConceptReferenceList matches = nodes.resolveToList(
                null, ConvenienceMethods.createLocalNameList("textualPresentation"), null, 1);

// Check the list size to see if any references are returned.  If true
// get the only referenced entity in the list and print out it's "presentation"
// or textual representation.
if(matches.getResolvedConceptReferenceCount() > 0)
{
    ResolvedConceptReference ref = (ResolvedConceptReference)matches.
                enumerateResolvedConceptReference().nextElement();
    Entity entry = ref.getReferencedEntry();
    System.out.println("Matching Name: " +
            entry.getPresentation(0).getValue().getContent() );
```

Service Metadata Retrieval

The LexEVS 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:

Java Code Snippet

```
// We can get a CodingSchemeRenderingList object directly from LexBigService
LexBIGService lbs = LexBIGServiceImpl.defaultInstance();
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.getCodingSchemeURI();
    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 LexEVS architecture is the ability to define multiple search and sort criteria without intermediate retrieval of data from the LexEVS service. Consider the following code snippet:

Java Code Snippet

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

// Start with an unconstrained set of all codes for the vocabulary
CodingSchemeVersionOrTag csvt = new CodingSchemeVersionOrTag();
csvt.setVersion(VERSION2);
CodedNodeSet cns = lbs.getCodingSchemeConcepts(SAMPLE_SCHEME2, csvt);

// Constrain to concepts with designations (assigned text presentations
// that contain text that sounds like 'Short Saphenous Vein'
cns = cns.restrictToMatchingDesignations(
                "Short Safinus Vane",
                SearchDesignationOption.ALL,
                MatchAlgorithms.DoubleMetaphoneLuceneQuery.toString(),
                null);

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

// Indicate that the resulting list should be sorted with the 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 && 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: " + rc.getConceptCode());
}
```

The example above 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 LexEVS 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 LexEVS 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 LexEVS 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 LexEVS 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, we recommend using literal search mechanisms such as the following:

- literalContains
- literal
- literalSubString

Additionally, 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 LexEVS 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 LexEVS 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 LexEVS service.

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

Additional Resources

- LexEVS 6.0 Local Runtime API javadocs
- LexEVS 6.x Local Runtime Installation Directory Guide

LexEVS GUI

The LexEVS Graphical User Interface, or GUI, is an optional component of the LexEVS install which will be in the /gui folder of the base LexEVS installation (see file breakdown in LexEVS 6.x Local Runtime Installation Directory Guide). The GUI is meant to provide a simple tool to test LexEVS API methods and quickly view the results; almost all public methods defined by the LexEVS API are supported. This guide provides a brief overview of how the GUI can aid programmers in writing code to the LexEVS API.

> **Note**
>
> The LexEVS GUI supports both administrative and test functions. Please refer to the *LexEVS Administratration Guide* for instructions on using the GUI as an administration tool.

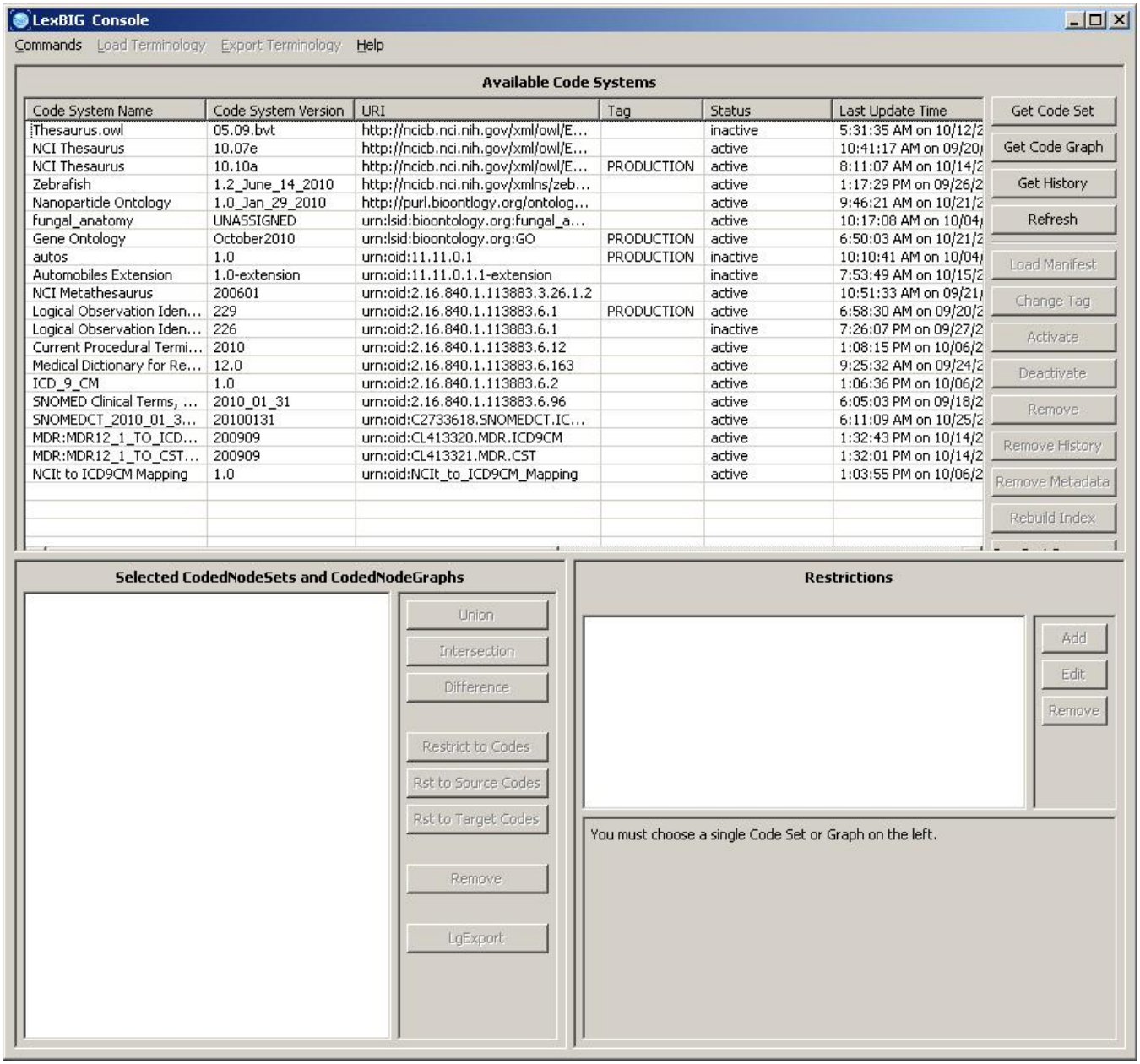
Launching the GUI

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. Opening the GUI for the first time you'll find no terminologies loaded unless you have run shell scripts to do so already. Instructions in the LexEVS Administration Guide will provide you with the instructions needed to load terminologies using the GUI.

A terminology service with loaded content will look something like this:



Overview

The top bar contains Administrative function drop down menus. The top right side bar with buttons has first query selection buttons and next administration functions. Just under these menus is a table of the current terminology set for this service. The left and right lower sections provide function for manipulating code sets and graphs. The lower left section displays the current set or graph sets on which Boolean logic, restriction and resolution functions can be performed. Code set and graph results can be restricted and tailored to the users needs on the lower right.

> **Note**
>
> The drop down menu options on the top bar are primarily used for administrative functions, like terminology loads, and are covered in detail by the *LexEVS Administration Guide* .

Creating New Queries

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

- **Refresh** - This button causes the LexEVS GUI to reread the available terminologies and their respective metadata. This can be useful when using the GUI to view a LexEVS 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

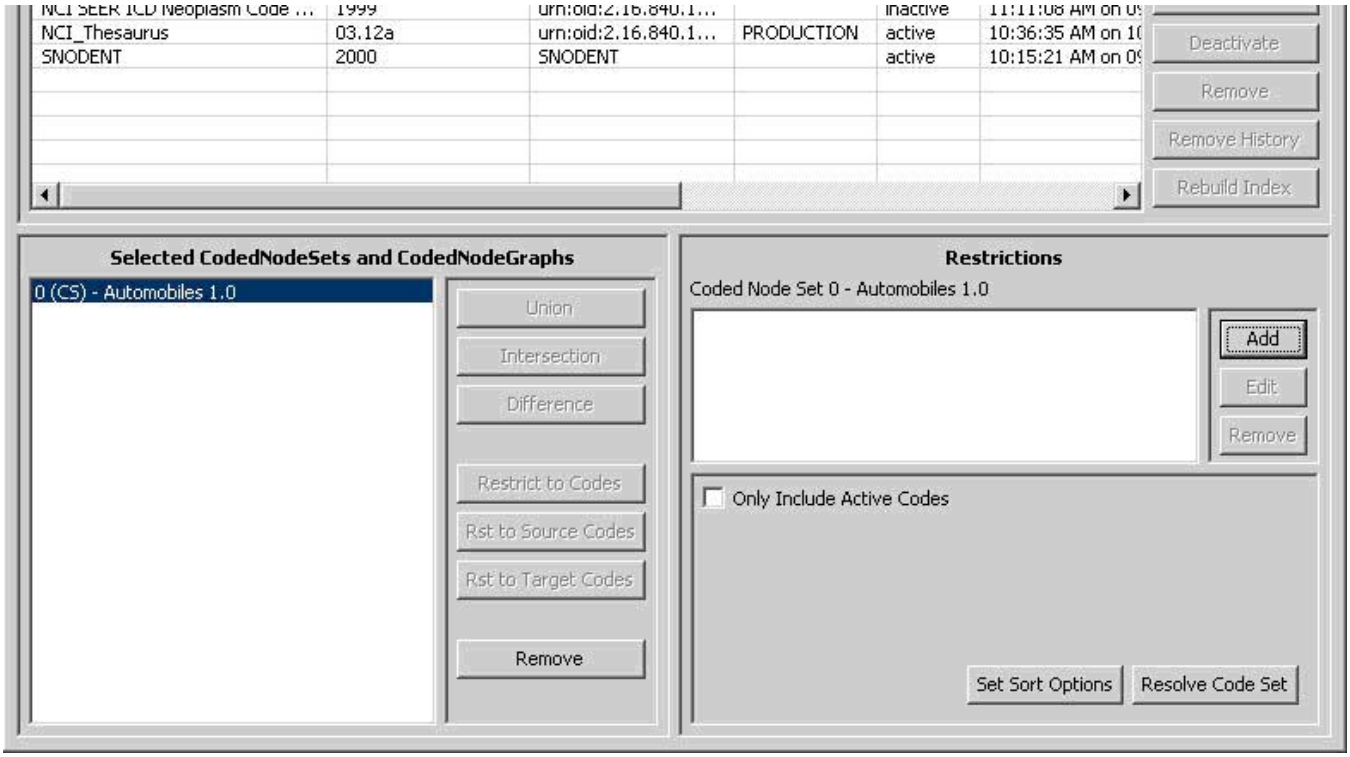
After selecting a code system and clicking on **Get Code Set** or **Get Code Graph**, a row will be added to the lower left section of the GUI for each click. There are seven buttons in the lower left section that allow combinatorial logic between the code sets in the lower left.

- **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.

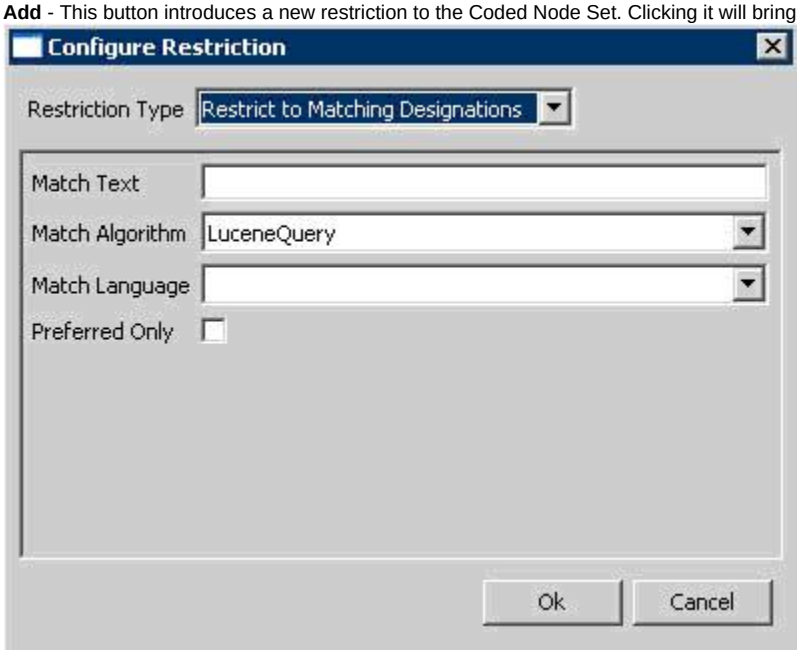
Working with Code Sets

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:

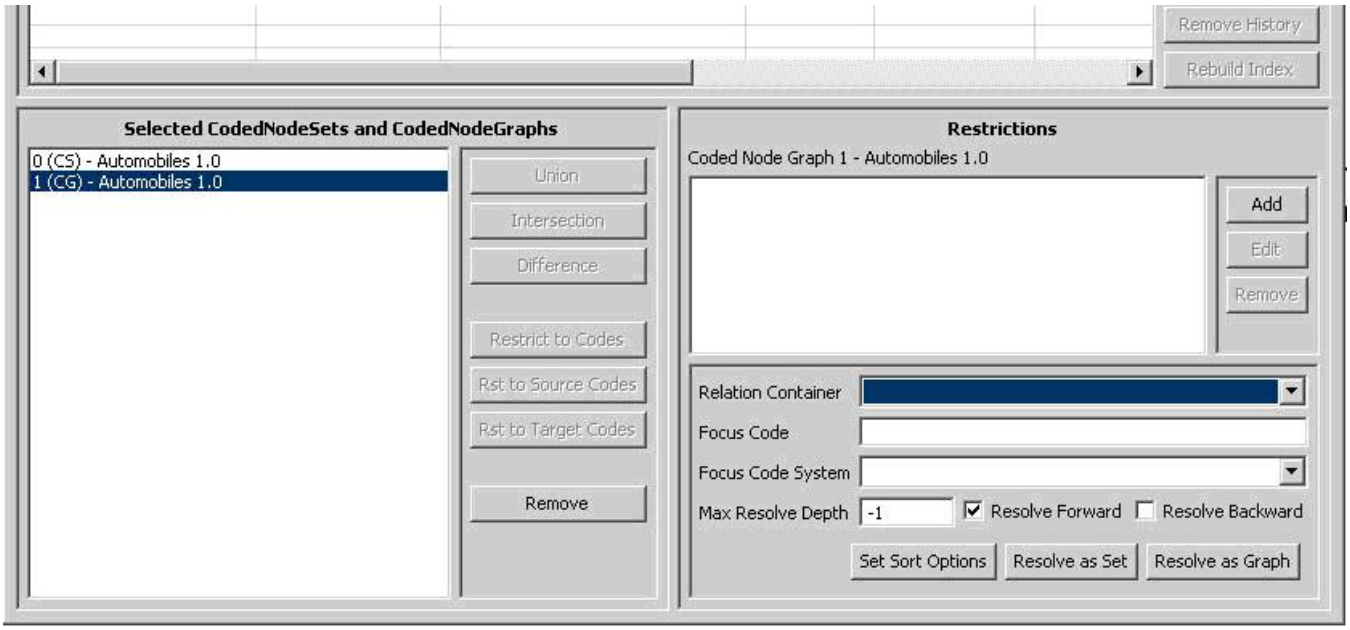


The top drop down list indicates the type of restriction to add. The rest of the dialog box will change depending on the type of restriction selected. All required parameters for the selected restriction type will be presented.

- **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 s elected 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 LexEVS GUI, the lower right section will look like this:



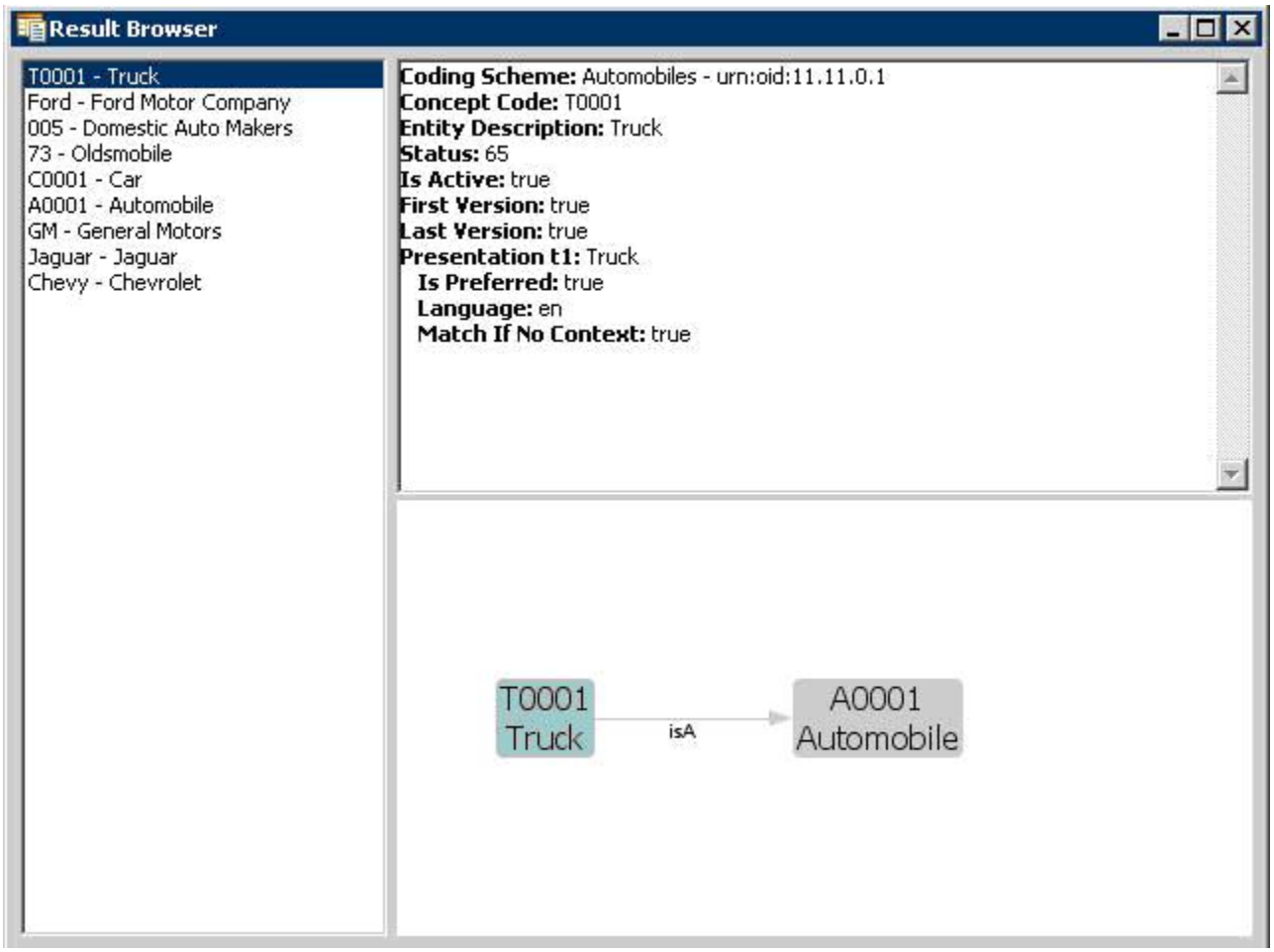
Again, there are two halves to the lower right section. The top half allows restrictions to be applied to the selected Code Graph, and it works the same as it does for a Coded Node Set. Please see the section above on applying restrictions to a Coded Node Set.

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

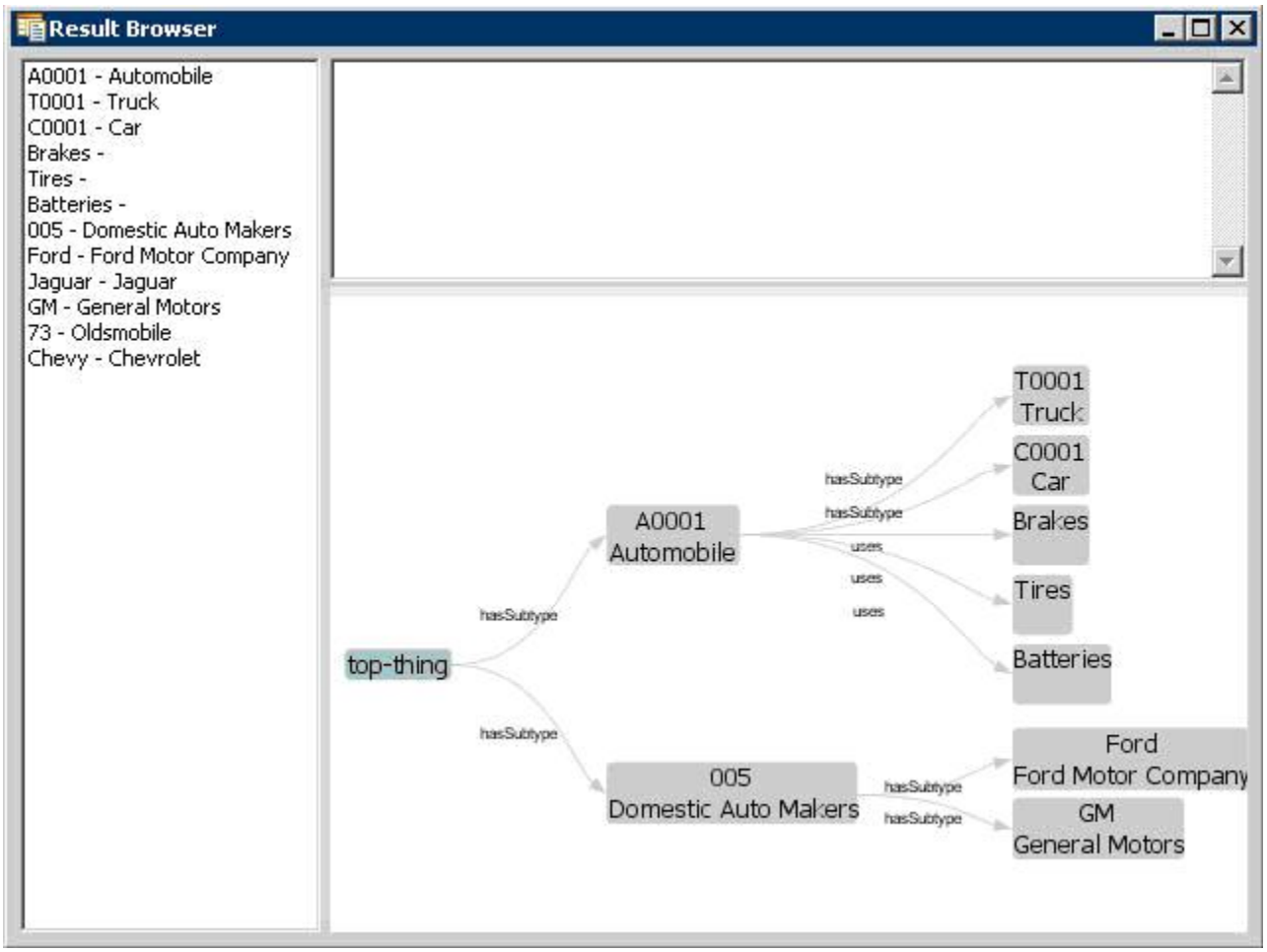
Viewing Query Results

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.

Value Set Services

For details about LexEVS Value Set Services, see LexEVS Value Set Service.

Pick List Services

For details about LexEVS Pick List Services, see LexEVS Pick List Service.

Asserted Value Set Services

For details about LexEVS Source Asserted Value Set Services, see LexEVS Asserted Value Set Service