



## LexEVS 5.x API

LexEVS 5.x Design and Architecture Guide LexBIG Extensions > LexEVS 5.x Design and Architecture Guide LexEVS Information Models > LexEVS 5.x Design and Architecture Guide LexEVS Architecture > LexEVS 5.x Programmer's Guide > LexEVS 5.x API

### vocabkc contents

- [Main Page](#)
- [What's New](#)
- [Forums](#)
- [Bugzilla](#)
- [Code Repository](#)
- [Feedback](#)
- [Contact Us](#)

### tools

- [LexBIG/LexEVS](#)
- [LexWiki](#)
- [NCI Protégé](#)
- [Related Tools and Models](#)

### projects

- [LexAjax](#)
- [LexGrid](#)
- [Cancer Data Standards Repository \(caDSR\)](#)
- [Common Terminology Criteria for Adverse Events \(CTCAE\)](#)
- [Open Health Natural Language Processing \(OHNLP\) Consortium](#)
- [Ontology Development and Information Extraction \(ODIE\)](#)

### semantic infrastructure

- [SI Main Page](#)
- [Initiatives](#)
- [Requirements](#)

### other resources

- [Library of Documents](#)
- [Documentation and Training for Tools](#)
- [Index of Terminologies](#)
- [Standards and Standards Influencing Organizations](#)
- [Outreach](#)

### external links

- [VCDE Workspace](#)
- [caBIG@ Community Website](#)
- [caBIG@ Support Service Providers](#)

### help

- [Editing Wiki Pages](#)
- [Editing Forum Posts](#)
- [Contact Us](#)

### Contents [\[hide\]](#)

- 1 [Introduction](#)
- 2 [Core Services](#)
- 3 [Service Extensions](#)
  - 3.1 [Query Extensions](#)
  - 3.2 [Load Extensions](#)
  - 3.3 [Export Extensions](#)
  - 3.4 [Index Extensions](#)
  - 3.5 [Generic Extensions](#)
- 4 [Utilities](#)
  - 4.1 [Iterators](#)
  - 4.2 [Search Algorithms](#)
  - 4.3 [Additional Utility Classes](#)
- 5 [Code Examples](#)
  - 5.1 [Concept Resolution](#)
  - 5.2 [Service Metadata Retrieval](#)
  - 5.3 [Combinatorial Queries](#)
  - 5.4 [Additional Resources](#)
- 6 [LexEVS GUI](#)
  - 6.1 [Launching the GUI](#)
  - 6.2 [Overview](#)
  - 6.3 [Creating New Queries](#)
  - 6.4 [Customizing Queries](#)
  - 6.5 [Working with Code Sets](#)
  - 6.6 [Working with Code Graphs](#)
  - 6.7 [Viewing Query Results](#)
- 7 [Value Domain Services](#)
- 8 [Pick List Services](#)

## Introduction

This document is a section of the [Programmer's Guide](#).

The LexEVS APIs fall into three primary categories:

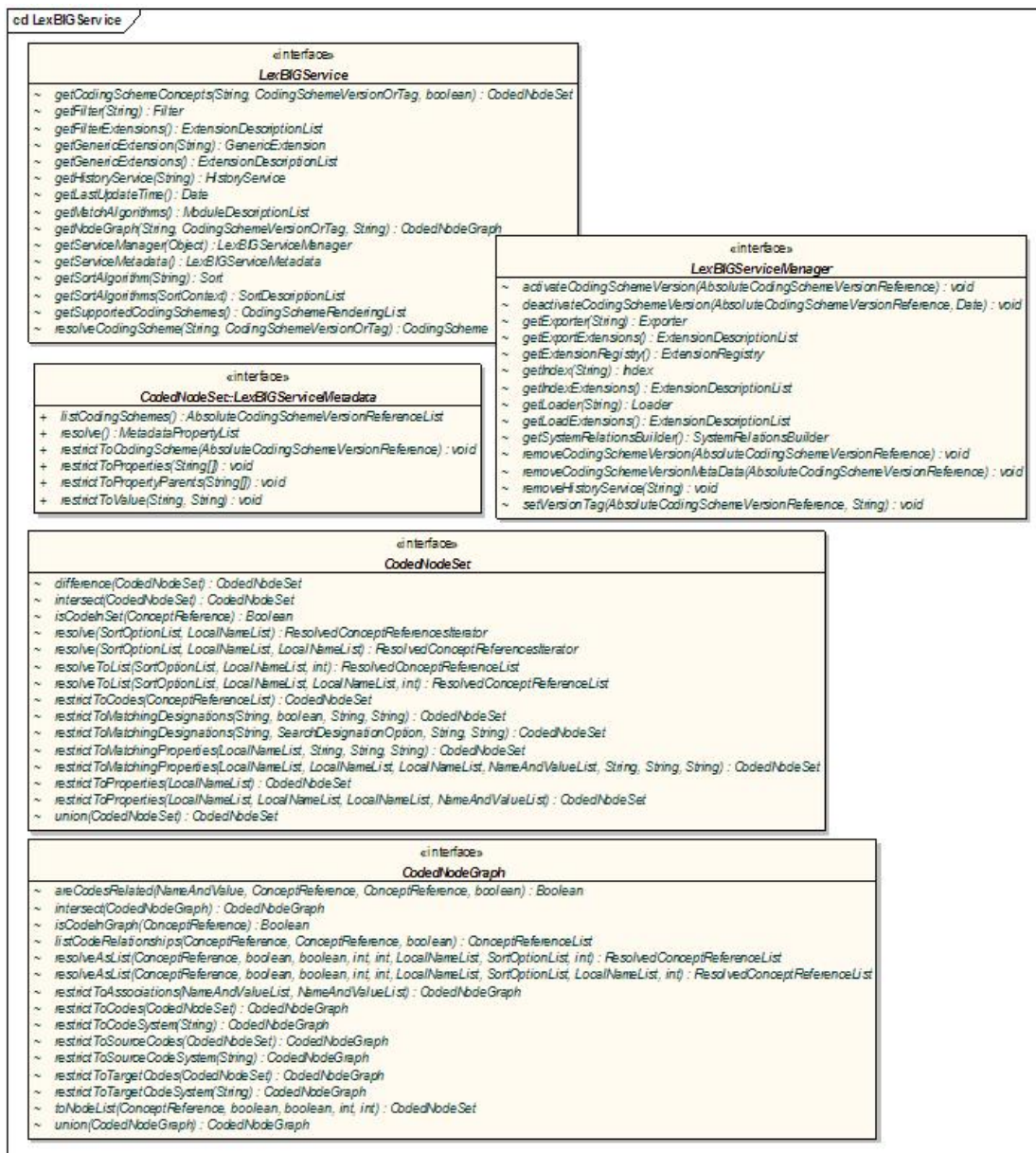
- **Core Services**
  - Includes the LexBIGService, LexBIGServiceManager, CodedNodeSet and CodedNodeGraph classes, which provide the initial entry points for programmatic access to all system features and data.
- **Service Extensions**
  - The extension mechanism provides for pluggable system features. 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

The LexBIGService illustrated below provides central entry points for programmatic access to system features and data.

search

- toolbox
- What links here
  - Related changes
  - Upload file
  - Special pages
  - Printable version
  - Permanent link
  - Print as PDF



The following are the components of interest:

**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 LexEVS 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.

**Value Domain and Pick List Services**

For details, see [ [the Value Domain and Pick List Services section of this guide](#) ].

**Service Extensions**

Provides registration and lookup for pluggable system features.



The following are the components of interest:

### **ExtensionRegistry**

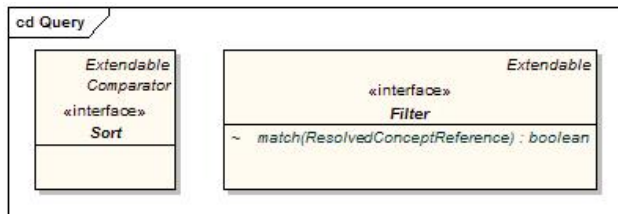
Allows registration and lookup of implementers for extensible pieces of the LexEVS architecture.

### **Extendable**

Marks a class as an extension to the LexEVS 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. For details on the LexEVS v5.1 Query Extension, see the document section [Query Services Extension](#).



The following are the components of interest:

### **Filter**

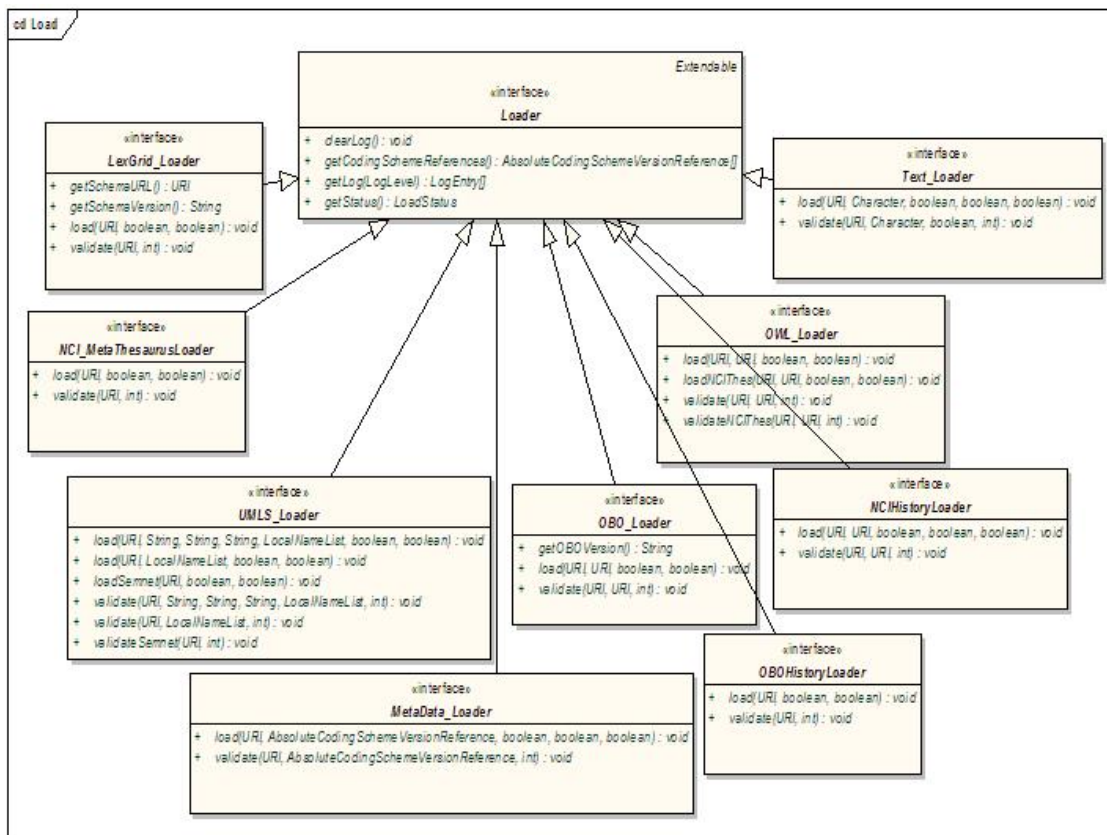
Allows for additional filtering of query results.

### **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 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:

#### **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 LexEVS 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**

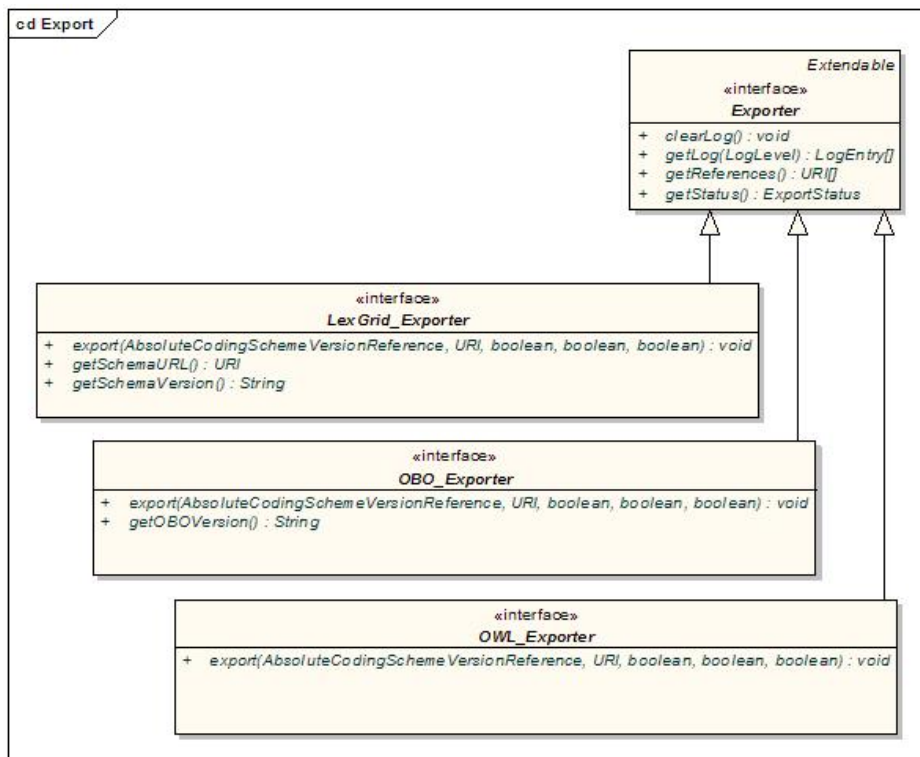
A loader that takes the delimited NCI history file and applies it to a coding scheme.

#### **OBOHistoryLoader**

Load an OBO change history file.

#### **Export Extensions**

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



The following are the components of interest:

#### **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**

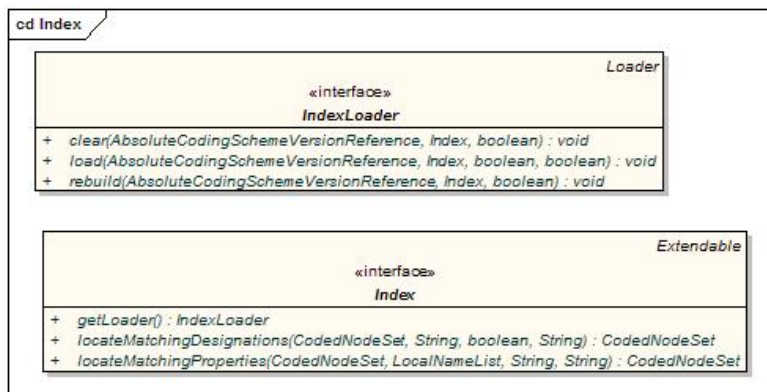
Exports content to OBO text format.

#### **OWL\_Exporter**

Exports content to OWL XML format.

### Index Extensions

Index extensions are built to optimize the finding, sorting and matching of query results.



The following are the components of interest:

#### **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.

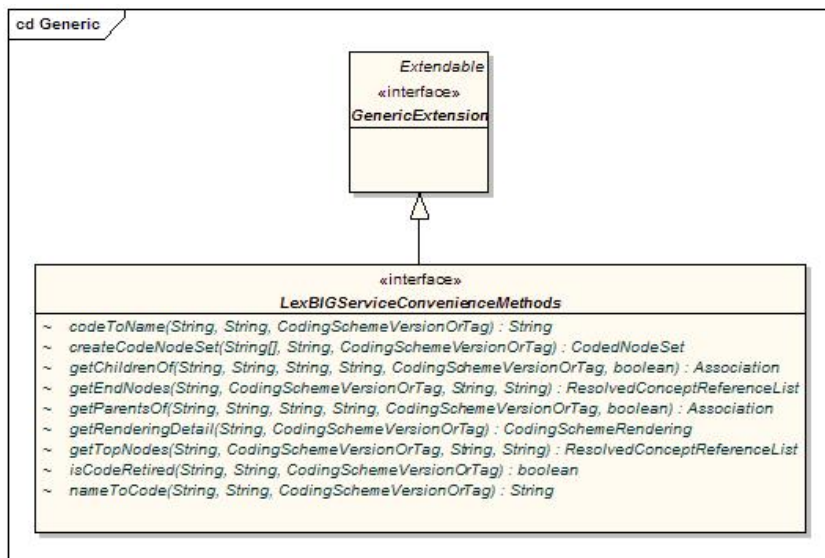
#### **IndexLoader**

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.



The following are the components of interest:

### **GenericExtension**

The generic extension class. Classes that implement this class are accessible via the LexBIGService interface.

### **LexBIGServiceConvenienceMethods**

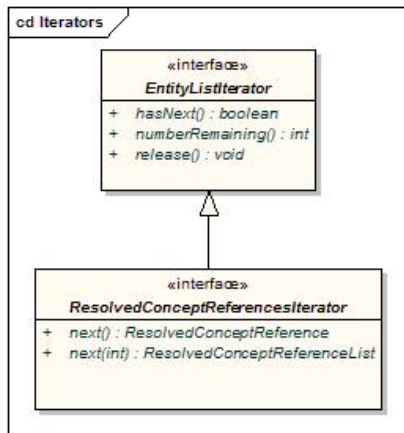
Convenience methods to be implemented as a generic extension of the LexEVS API.

## Utilities

Defines helper classes externalized by the LexEVS API.

## Iterators

Iterators are used to provide controlled resolution of query results.



The following are the components of interest:

### **EntityListIterator**

Generic interface for flexible resolution of LexEVS objects.


### **ResolvedConceptReferencesIterator**

An iterator for retrieving resolved coding scheme references.


## Search Algorithms

Supported LexEVS Search Algorithms


### Search Algorithm

Name: LuceneQuery  
 Version: 1.0  
 Description: Search with the Lucene query syntax.  
 See [http://lucene.apache.org/java/2\\_3\\_2/queryparsersyntax.html](http://lucene.apache.org/java/2_3_2/queryparsersyntax.html) 

### Search Algorithm

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'  
 See [http://lucene.apache.org/java/2\\_3\\_2/queryparsersyntax.html](http://lucene.apache.org/java/2_3_2/queryparsersyntax.html) 

### Search Algorithm

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'  
 See [http://lucene.apache.org/java/2\\_3\\_2/queryparsersyntax.html](http://lucene.apache.org/java/2_3_2/queryparsersyntax.html) 

### Search Algorithm

Name: startsWith  
 Version: 1.0  
 Description: Equivalent to 'term\*' (case insensitive)


### Search Algorithm

Name: exactMatch  
 Version: 1.0  
 Description: Exact match (case insensitive)

### Search Algorithm

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.

### Search Algorithm

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> 

## Additional Utility 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: **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.

## 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 `Concept` objects. The following example provides a simple query of concept codes:

```
// 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 we 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 in the properties list,
// getting the text && 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).getValue());
}
else
{
    System.out.println("No match found");
}
}

```

## 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 a brief example showing how to access and print some of this metadata:

```

// We can get a CodingSchemeRenderingList object directly from 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.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:

```

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 ventricle",
    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 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());
}

```

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 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, 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 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

For more code snippets, see [LexEVS Code Snippets](#).

The examples and automated test programs provided by the LexEVS installation (see file breakdown in [Overview of the Software](#)) are available as additional reference materials.

## 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 [Overview of the Software](#)). 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 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:

The screenshot shows the LexBIG 1.0 alpha application window. The title bar reads "LexBIG 1.0 alpha" and the menu bar includes "Commands", "Load Terminology", "Export Terminology", and "Help". The main area is titled "Available Code Systems" and contains a table with the following data:

Code System Name	Code System Version	URN	Tag	Status	Last Update Time	
Amino Acid Ontology	2006/05/18	http://www.co-ode...		active	10:46:10 AM on 09/	Get Code Set
autos	1.0	urn:oid:11.11.0.1		active	2:48:05 PM on 10/	Get Code Graph
cell	UNASSIGNED	urn:lsid:bioontology...		active	2:22:33 PM on 10/	Get History
COSTAR, 1989-1995	89-95	urn:oid:2.16.840.1...		active	4:32:03 PM on 09/	Refresh
Dictyostelium discoideum anatomy	UNASSIGNED	urn:lsid:bioontology...		active	12:21:51 PM on 09/	Change Tag
Drug Ontology Schema	unknown	http://www.owl-on...		active	4:14:16 PM on 10/	Activate
Galen additions for Lexical Proc...	unknown	http://example.org...		active	4:18:49 PM on 10/	Deactivate
gene_ontology	UNASSIGNED	urn:lsid:bioontology...		active	12:16:29 PM on 09/	Remove
GMP	2.0	urn:oid:11.11.0.2		active	2:48:02 PM on 10/	Remove History
NCI MetaThesaurus	200510E	urn:oid:2.16.840.1...		active	2:06:17 PM on 07/	Rebuild Index
NCI SEER ICD Neoplasm Code ...	1999	urn:oid:2.16.840.1...		inactive	11:11:08 AM on 09/	
NCI_Thesaurus	03.12a	urn:oid:2.16.840.1...	PRODUCTION	active	10:36:35 AM on 10/	
SNODENT	2000	SNODENT		active	10:15:21 AM on 09/	

Below the table are two main sections: "Selected CodedNodeSets and CodedNodeGraphs" and "Restrictions". The left section contains buttons for Union, Intersection, Difference, Restrict to Codes, Rst to Source Codes, Rst to Target Codes, and Remove. The right section contains buttons for Add, Edit, and Remove, and a message: "You must choose a single Code Set or Graph on the left."

## 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 *LexEVS 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 *LexEVS 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 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:

NCI BEER ICD Neoplasm Code ...	1999	urn:oid:2.16.840.1...		inactive	11:11:08 AM on 10/10/2010	
NCI Thesaurus	03.12a	urn:oid:2.16.840.1...	PRODUCTION	active	10:36:35 AM on 10/10/2010	
SNODENT	2000	SNODENT		active	10:15:21 AM on 09/29/2010	

Deactivate  
Remove  
Remove History  
Rebuild Index

---

**Selected CodedNodeSets and CodedNodeGraphs**

0 (CS) - Automobiles 1.0

Union
Intersection
Difference


---

Restrict to Codes
Rst to Source Codes
Rst to Target Codes


---

Remove

**Restrictions**

Coded Node Set 0 - Automobiles 1.0

Add
Edit
Remove

Only Include Active Codes

Set Sort Options
Resolve Code Set

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:

**Configure Restriction** ✕

Restriction Type: Restrict to Matching Designations

---

Match Text:

Match Algorithm: LuceneQuery

Match Language:  

Preferred Only:

Ok
Cancel

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 LexEVS GUI, the lower right section will look like this:

The screenshot displays the LexEVS 5.x API interface. At the top right, there are buttons for 'Remove History' and 'Rebuild Index'. The main interface is divided into two primary sections:

- Selected CodedNodeSets and CodedNodeGraphs:** This section contains a list of selected items: '0 (CS) - Automobiles 1.0' and '1 (CG) - Automobiles 1.0'. To the right of this list are several buttons: 'Union', 'Intersection', 'Difference', 'Restrict to Codes', 'Rst to Source Codes', 'Rst to Target Codes', and 'Remove'.
- Restrictions:** This section is titled 'Coded Node Graph 1 - Automobiles 1.0'. It includes a large empty text area with 'Add', 'Edit', and 'Remove' buttons to its right. Below this are several configuration options:
  - 'Relation Container' with a dropdown menu.
  - 'Focus Code' with a text input field.
  - 'Focus Code System' with a dropdown menu.
  - 'Max Resolve Depth' with a text input field set to '-1' and two checkboxes: 'Resolve Forward' (checked) and 'Resolve Backward' (unchecked).
  - Three buttons at the bottom: 'Set Sort Options', 'Resolve as Set', and 'Resolve as Graph'.

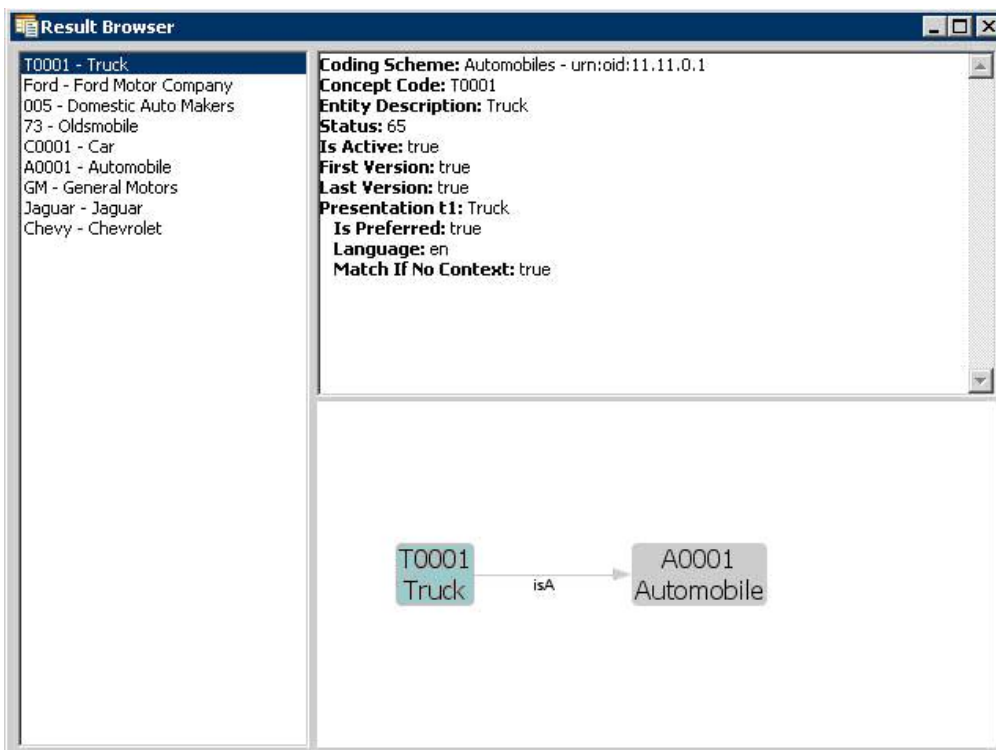
Again, there are two halves to the lower right section. The top half allows restrictions to be applied to the selected Code Graph, and it works the same as it does for a Coded Node Set. Please see the section above on applying restrictions to a Coded Node Set.

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.

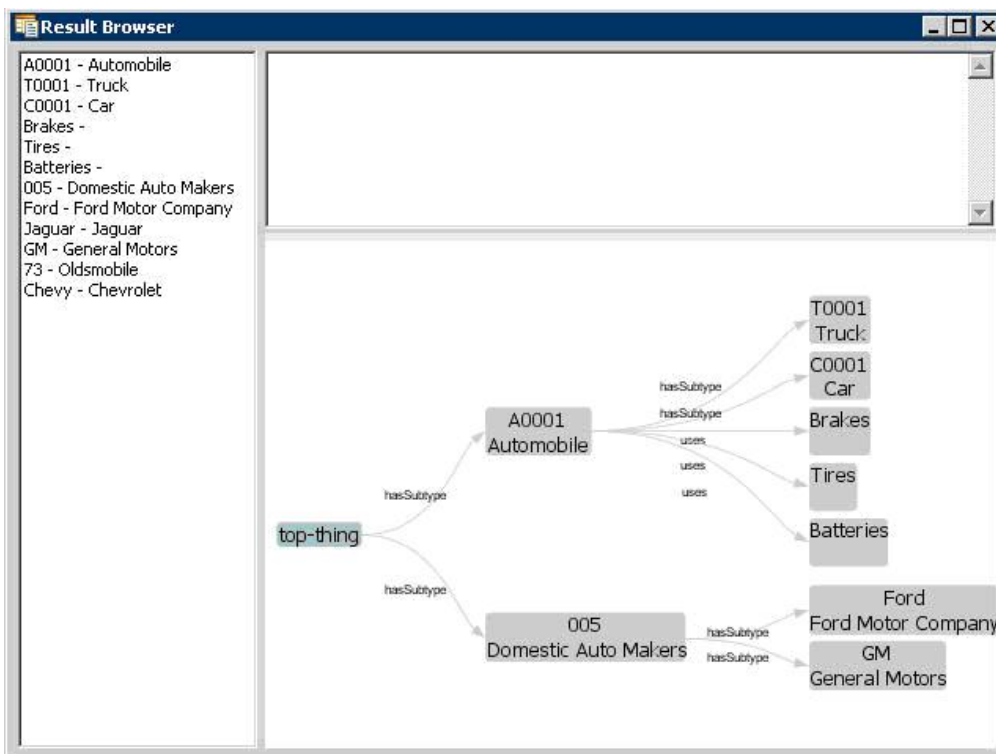
### Viewing Query Results

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 Domain Services

For details about LexEVS Value Domain Services, see [LexEVS Value Domain Services](#)

## Pick List Services

For details about LexEVS Pick List Services, see [LexEVS Pick List Services](#)

Categories: [VKC Contents](#) | [Documentation](#) | [LexEVS Code](#) | [LexEVS](#)



This page was last modified on 20 January 2010, at 15:32. This page has been accessed 185 times.

[CONTACT US](#) | [PRIVACY NOTICE](#) | [DISCLAIMER](#) | [ACCESSIBILITY](#) | [APPLICATION SUPPORT](#)

