- Home
- Knowledge Centers
    - caGrid
    - Clinical Trials Management Systems
    - Data Sharing and Intellectual Capital
    - Molecular Analysis Tools
    - Tissue/Biospecimen Banking and Technology Tool
    - Vocabulary
- Discussion Forums
    - caGrid
    - Clinical Trials Management Systems
    - Data Sharing and Intellectual Capital
    - Molecular Analysis Tools
    - Tissue/Biospecimen Banking and Technology Tool
    - Vocabulary
- Bugs/Feature Requests
- Development Code Repository

LexEVS 5.0 Migration Guide

From Vocab_Wiki

Contents

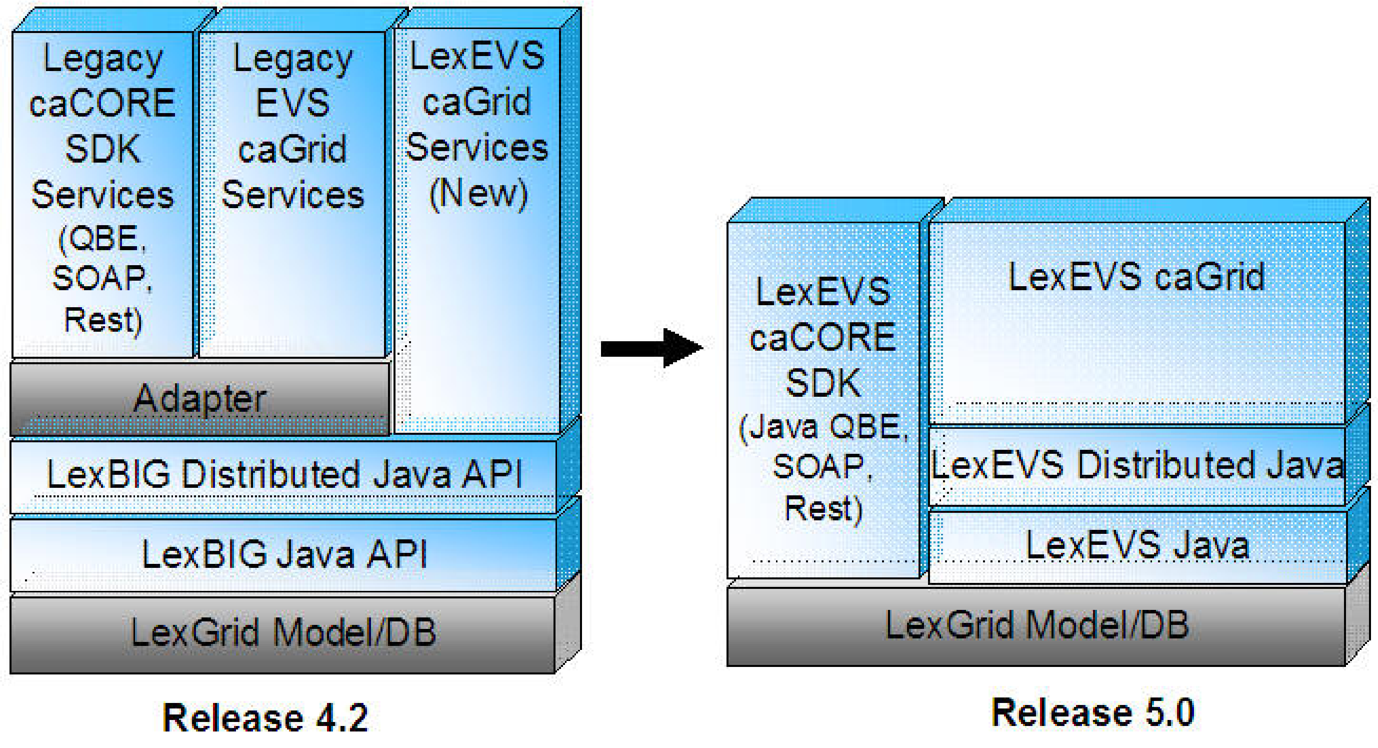
Overview

LexEVS 5.0 represents the next generation of NCI Enterprise Vocabulary Services. In this release, the LexBIG Java API and LexGrid model become the strategic EVS interfaces, replacing the legacy EVS API and the EVS 3.2 model.

LexEVS 5.0 Highlights

- LexEVS 5.0 is the first release to completely shift from the EVS Model and EVS API to LexBIG API (LexEVS).
- Consistent naming and release numbers for API and services.
- Introduction of LexGrid-based QBE services
- Unified OWL loader.
- The 2008/01 model is updated to the 2009/01 LexGrid Model.

Unified Design
The unified design of LexEVS 5.0 no longer supports the EVS Model and EVS API. Both have been completely replaced with LexEVS components as represented in the following diagram.



Legacy caCORE SDK Services (QBE, SOAP, Rest)
Legacy EVS caGrid Services
LexEVS caGrid Services (New)
Adapter
LexBIG Distributed Java API
LexBIG Java API
LexGrid Model/DB
Release 4.2
LexEVS caCORE SDK (Java QBE, SOAP, Rest)
LexEVS caGrid
LexEVS Distributed Java
LexEVS Java
LexGrid Model/DB
Release 5.0

The convergence of LexEVS 5.0 components has introduced new naming of components:

Design Components	Release 4.2	Release 5.0
EVS Model Version	3.2	No Longer Available
EVS API Version	3.2	No Longer Available
LexGrid Model Version	2008/01	2009/01

LexBIG API Version	2.3.0	LexEVS 5.0

The supported programming interfaces are now all provided by LexEVS:

Supported Programming Interfaces	Release 4.2	Release 5.0
Direct Java	LexBIG	LexEVS
Distributed Java (RMI)	LexBIG	LexEVS
caCORE SDK Services	EVS	LexEVS
caGRID Service	EVS, LexEVS	LexEVS

As a result, definitions have been unified to represent LexEVS. The following definitions are provided for reference.

Term	Definition
LexGrid	▪ LexGridVocabulary model underlying the LexBIG API. ▪ Sometimes used as a generic reference to work based off this model.
LexBIG	▪ A new API with rich functionality developed for NCI caBIG® to access LexGrid-based vocabularies. ▪ Serves as the internal ‘engine’ for traditional EVS APIs.
EVS	▪ NCI Enterprise Vocabulary Services model, API, and content. ▪ For model and API, references legacy components being replaced by LexGrid (model) and LexBIG (API).
LexEVS	▪ Adopted as project name to describe merging of LexGrid model and LexBIG API as the mainstream EVS interfaces.

LexGrid-based QBE services

LexEVS 5.0 brings the addition of QBE/Data Services. Detailed information about the Data Services can be found in the LexEVS 5.0 Programmer's Guide.

LexEVS Model

The transition from the 2008/02 model to the 2009/01 model has introduced numerous enhancements. Information regarding the LexEVS model can be found in the LexEVS 5.0 Design and Architecture Guide.

The 2009/01 EA representation of the model can be found at the model and scheme page.

Model Highlights

- Accommodate entities other than concept/instance/association
- Converge attributes (e.g. associated properties) to ‘Entity’ superclass
- Single resource can be defined as multiple types
- Allow more granular version tracking (e.g. per concept or per property)
- Extensive updates to value domain and pick list representation
- Remove antiquated packages & classes (e.g. LDAP)
- Accuracy and alignment of internal lexicon (URNMap -> URIMap)
- Influenced by CTS2, OWL, XMDR, GE/IHC
- Formalized (EA model available)

Detailed changes are documented below:

Model Change	**Type**	**Description**	**How Implemented**
Remove LDAP Implementation	Feature Request	The LDAP implementation of the LexGrid model is no longer being used. The LDAP specific elements in the LexGrid model should be removed, as they add a degree of complexity and confusion that is no longer justified.	▪ Removed LDAP Package ▪ Removed NumericOID type ▪ Removed all LDAP annotation on the individual entities ▪ Removed the "dc" type on aggregation columns ▪ Removed the constraint that all nodes had to have a single identifier that was unique in the context of the parent
Model Clarification	Enhancement Request	There are several issues that have made the model difficult to explain, implement, and use. These issues include: 1. The inconsistent use of names - some core data types begin with “ts” and others don't. 2. Naming confusion - “URN” is used in several places where the data type should be a URI, labels say “id” when they mean “code”, etc. 3. Inconsistent typing - localId is used as a type throughout much of the model instead of specifying the particular domain (e.g. source, language, namespace name, etc) 4. Inconsistent use of the “any” type - it has a misleading label in the builtins section (tsCaseSensitiveDirectoryString), and then isn't used consistently through the model. 5. While we aren't yet resorting to relaxNG, we would like the XML validation to catch more obvious errors. Weak typing prevents some of this validation, but setting minLength to “1” on fields that are expected to have content will reduce the number of XML documents that validate but don't load correctly	
			Changed builtins name to "tsAnType". Added an optional dataType attribute to

All text type fields need data types	Enhancement Request	We need the ability to add a data type (format) to the target of associations as well as the target of any property. Right now, it only applies to property	the field. Changed the types of "text" and "entityDescription to "tsAnyType". Changed property and associationData to have a reference to an entity of type "text" rather than being a kind of "text"
Flexible Entity types	Enhancement Request	The 2008/01 model supports a fixed set of entity types - concept, relation and instance. While this aligns with OWL/DL, it doesn't account for (a) terminologies that haven't differentiated classes from individuals, (b) OWL Full, where an entity can be both a class and an individual and (c) other type systems, such as that supported by KTMI	Made "entity" a first class class, that included all properties and characteristics. Added an optional, repeating entityType field which allowed the entity to be classified, added supportedEntityType in the mappings to allow types to be customized, removed the fixed enumeration of types and added constraints that define "concept", "instance" and "relation" by the entityType field
Incremental Revisions	Enhancement Request	LexGrid updates need to be communicated as sets of changes rather than complete sets of contents. The history mechanism needs to be extended so that a collection of changes can be communicated that will allow an existing system to be updated incrementally	Changed the definition of versionable, added entityState and revision model elements and changed the definition of systemRelease to carry a collection of revisions. Note: This change is closely coupled with the Refined History Model change request.
Refined History Model	Enhancement Request	LexGrid needs the ability to version and activate changes on the property, entity, association instance, pick list, pick list entry level in addition to the concept level. Each of these entities need to support a status, state (active or inactive) the date/time when the change is to become active, the data/time when it is to become inactive, and additional metadata about who, how and why the change should be applied.	▪ Revised versionable to support the activation state, status, effective and expiration dates ▪ Provided an optional link from versionable to an entryState record that carried the type of change and the revision that this change was included in. Created a model of state changes (changeType), and created machanisms for traversing revisions to determine what changed, when, where, etc.
			Added a new localId type, "NamespaceName" and a

Namespaces Aren't CodingSchemes	Bug Fix	The namespace used to qualify the URI of a coded entity isn't necessarily the namespace of the coding scheme making an assertion about the entity. These two elements are convoluted in the current LexGrid model.	new mapping, "supportedNamespace". Changed the codingSchemeId attribute of "entity" to entityCodeNamespace, which references a supportedNamespace. If entityCodeNamespace is present, it references a supportedNamespace, which, in turn, has an attribute, "equivalentCodingScheme", which has the local name of the codingScheme that corresponds to the entityCodeNamespace
Revise Value Domains / Pick Lists	Enhancement Request	The value domain model needs to be extended to support the definitions represented in the IHC model. In addition, the model needs to support (a) HL7's value domain definition model and the sort of definitions that can be created through the DTS editor. The model of pick lists need to be extended accordingly to meet multiple GE/IHC requirements	Completely replaced the ValueDomains package to carry the new model.
Dual Type Properties	Enhancement Request	RDF based loaders transform triples into a combination of (a) first class attributes (e.g. entityDescription, copyRight, presentation, definition, ...) (b) generic lexical properties or (c) relations. Properties and relations preserve the original RDF type, but the first class attributes lose the information about where the resource was derived from. In addition, there is no way to assign status and provenance information to the first class attributes (see: Refined History Model)	Model philosophy was changed to have first class attributes represented in two forms: as an attribute and as a property that is identified as being an attribute. To do this, we added new attributes to property and propertyQualifier that, if non-blank, stated the first class entity that this property (or propertyQualifier) represented. As an example, the copyRight entity would also be represented as a property with a propertyType of "copyRight"
Backwards Compatibility	Feature Request	The LexBIG API history API goes directly against the NCI Cumulative History content and renders its output in terms of codingSchemeVersion, version and entity version. These elements need to be preserved in the new LexGrid model as deprecated elements that exist for backwards compatibility with the LexBIG API.	Preserved Version, Version Reference, representsVersion, entityVersion and codingSchemeVersion, but marked them as "deprecated"
		The 2008/01 model has two different “mappings” entities, one for	

Common Mappings Type	Feature Request	codingScheme and one for valueDomain. Each has a different collection of supported, and the order of the entries are confusing and arbitrary. With the addition of another “mappings” entry for pick list, we suggest that the three mappings be consolidated into one type, and the contents be alphabetized. This will make code management and authoring easier. (second entry) It should be possible to enter a codingScheme, valueDomain or pickList without having *any* mappings entries and have the loader fill out the information of all of the localId's and, where possible, the URI's that they map to. This should not be the function of an editor or type transformation package	Created a new "mappings" entry in Naming package, removed the existing entries from codingScheme and valueDomain and pointed them at the new entry. Alphabetized the entries and made them all optional.
Agent Role on supportedSource	Bug Fix	supportedSource has an agentRole field, but role is a property of the association of the source with the entity (e.g. the source may be author on one field, editor on another).	Removed agentRole from supportedSource
Assertions can be inferred, entities cannot	Bug Fix	isInferred is listed as a property of a concept. DL can infer additional axioms about a concept, but they cannot infer the existence of a concept that isn't already specified.	Moved isInferred from concept to associatiableElement
PropertyLink.propertyLink is confusing	Bug Fix	the link attribute in the propertyLink element was renamed to "propertyLink". This is confusing	(not fixed yet)
isTranslationAssociation is not a property of the association, but how it is used.	Bug Fix	Association was made a first class entity in the 2008/01 model, meaning that all of the characteristics had to be properties of the association itself, not how it used in a particular relations collection. isTranslationAssociation is a property of use, yet is listed as a property of the association itself	Removed isTranslationAssociation from the model. No alternative available at the moment
targetCodingScheme is not a property of an association, but how it is used.	Bug Fix	Association was made a first class entity in the 2008/01 model, meaning that all of the characteristics had to be properties of the association itself, not how it used in a particular relations collection. targetCodingScheme is a function of how it is used, not the association itself	Removed targetCodingScheme from the model, meaning that mappings across coding schemes will always have to provide the namespace id for the target element.
associationName is a localId, not an entityCode	Bug Fix	Association was made a first class entity in the 2008/01 model, and associationName was removed anticipating that the entityCode and associationName would always be the same thing. This may not be the case,	Reintroduced associationName as a localId with the

		however, as an ontology may use, say "isA" as the name of an association, but define it as being the same as an entirely different concept in a different namespace.	supportedAssociation mapping entries
Type is an attribute of entity, not usage. An entity can have multiple types	Bug Fix	sourceEntityType and targetEntityType are incorrect in the associations package, as they assume that the type is part of an entity's identity (i.e. you can have two entities with the same URI, one of which is a class and one an individual).	sourceEntityType and targetEntityType were removed from the model
Need to select associations by context	Enhancement	IHC needs to be able to select associations based on a passed context	added usageContext attribute to associatableElement
Need instance identifiers on associations	Bug Fix	You can assign an identifier to a DataProperty type association, but not to ObjectProperty type association. Both IHC and SNOMED-CT maintain unique identifiers on associations	Removed dataId from the associationData and created associationInstanceId in the associatableElement class. This field is optional, as dataId originally existed for LDAP compatibility.
Need to know whether an association participates in the definition of a concept	Enhancement	While OWL doesn't currently support this, it is useful to understand whether a assertion is considered to be part of the definition of an entity or simply an additional fact that is known about that entity.	Added isDefining attribute to the associatableElement class

LexEVS Database Enhancements

To support the LexEVS 2009/01 Model, numerous changes to the database were necessary. Although the database is not exposed to the user, it is important to take note of the changes. A 2009/01 sample database (Microsoft Access format) can be found at the model and scheme page.

Table Name	Column Name	Changes
codingScheme	formalName	Can be null
	defaultLangauge	Can be null
	codingSchemeURI	renamed from registeredName
	isActive	added
	releaseURI	added
	firstRelease	removed
	modifiedInRelease	removed
	deprecated	removed
codingSchemeSupportedAttrib	uri	renamed from urn
codingSchemeProp	isActive	added
entity	codingSchemeName	renamed from codingSchemeId
table renamed from concept	entityCodeNamespace	added, primaryKey

	entityCode	renamed from id
	firstRelease	removed
	modifiedInRelease	removed
	deprecated	removed
	isInferred	removed
entityType	codingSchemeName	added
New table	entityCodeNamespace	added
	entityCode	added
	entityType	added
entityProperty	codingSchemeName	renamed from codingSchemeId
	entityType	removed
	entityCode	renamed from entityId
	format	removed
	entityCodeNamespace	added, primaryKey
	isActive	added
entityPropertyLink	codingSchemeName	renamed from codingSchemeId
	entityType	removed
	entityCodeNamespace	added, primaryKey
entityPropertyMultiAttrib	codingSchemeName	renamed from codingSchemeId
	entityType	removed
	entityCodeNamespace	added, primaryKey
	qualType	added
relation	codingSchemeName	renamed from codingSchemeId
	containerName	renamed from containerDC
association	codingSchemeName	renamed from codingSchemeId
	containerName	renamed from containerDC
	entityCode	renamed from id
	associationName	added
entityAssnsToEntity	codingSchemeName	renamed from codingSchemeId
	containerName	renamed from containerDC
	entityCode	renamed from associationId
	sourceEntityCodeNamespace	renamed from sourceCodingSchemeId
	sourceEntityCode	renamed from sourceId
	sourceType	removed
	targetEntityCodeNamespace	renamed from targetCodingSchemeId
	targetEntityCode	renamed from targetId

	targetType	removed
	firstRelease	removed
	modifiedInRelease	removed
	deprecated	removed
	associationInstanceId	renamed from multiAttributesKey
	isDefining	added
	isInferred	added
	isActive	added
entityAssnsToEQuals	codingSchemeName	renamed from codingSchemeId
entityAssnsToData	codingSchemeName	renamed from codingSchemeId
	containerName	renamed from containerDC
	entityCode	renamed from associationId
	sourceEntityCodeNamespace	renamed from sourceCodingSchemeId
	sourceEntityCode	renamed from sourceId
	associationInstanceId	renamed from multiAttributesKey
	isDefining	added
	isInferred	added
	isActive	added
	firstRelease	removed
	modifiedInRelease	removed
	deprecated	removed
	dataId	removed
entityAssnsToDQuals	codingSchemeName	renamed from codingSchemeId
entityAssnsToEntityTr	codingSchemeName	renamed from codingSchemeId
	containerName	renamed from containerDC
	entityCode	renamed from associationId
	sourceEntityCodeNamespace	renamed from sourceCodingSchemeId
	sourceEntityCode	renamed from sourceId
	sourceType	removed
	targetEntityCodeNamespace	renamed from targetCodingSchemeId
	targetEntityCode	renamed from targetId
nciThesHist	entityCode	renamed from entityId

EVS API to LexEVS API Migration

The transition from EVS API to LexEVS API will require the use of different methods to accomplish the same function as was previously provided in EVS API. This section will identify each deprecated class and provide the alternative LexEVS API.

As a result of the deprecation of the EVS API, the following classes are no longer available:

- EVSQuery and EVSQueryImpl
- EVSQueryDAOImpl - Not a public API.
- EVSWSDAOImpl - Not a public API.
- EVSWSQuery - Not a public API.
- DLBAdapter - Not a public API.
- DLBWrapper (DLBWrapper is extended by DLBAdapter and it is not been used anywhere else) - Not a public API.
- EVSApplicationService and EVSApplicationServiceImpl

EVSApplicationService and LexEVS Counterparts

EVSApplicationService

evsSearch and search

These methods are replaced either by the LexEVS API or the LexEVS caCORE SDK Data Service 'search' method. This uses the standard caCORE SDK API. More information about the caCORE SDK can be obtained on the caCORE SDK site.

EVSQuery Methods and LexEVS Counterparts

EVSQuery

getTree

- all 'getTree' functionality is replaced by the LexEVS **CodedNodeGraph** API. For example:

```
 public void getTree(String vocabularyName, String rootCode, boolean
direction, boolean isaFlag, int attributes, int levels, Vector roles);
```

is replaced by:

```
 CodedNodeGraph cng = lexevsService.getNodeGraph(String codingScheme,
CodingSchemeVersionOrTag, versionOrTag, String relationContainerName);
 ResolvedConceptReference[] rcr = cng.resolveAsList(ConceptReference
graphFocus, boolean resolveForward,
                                                     boolean
resolveBackward, int resolveCodedEntryDepth, int
resolveAssociationDepth,
                                                     LocalNameList
propertyNames, PropertyType[] propertyTypes, SortOptionList
sortOptions,
                                                     int
maxToReturn).getResolvedConceptReference();
```

Also, various methods in **LexBIGServiceConvenienceMethods** can be used to show hierarchies.

- See methods
  - getHierarchyRoots
  - getHierarchyRootSet
  - getHierarchyLevelNext
  - getHierarchyLevelPrev

    - getHierarchyPathToRoot

- For examples, see the LexEVS Example classes
    - org.LexGrid.LexBIG.example.BuildTreeForCode
    - org.LexGrid.LexBIG.example.ListHierarchy
    - org.LexGrid.LexBIG.example.ListHierarchyByCode
    - org.LexGrid.LexBIG.example.ListHierarchyPathToRoot

---

searchDescLogicConcepts

LexEVS provides many ways to restrict the result of a query. The method 'searchDescLogicConcepts' searches for matches based on a text String. To conduct similar queries using LexEVS, use the **CodedNodeSet** API.

Obtain a CodedNodeSet from LexEVS:

```
 CodedNodeSet nodes = lexevsService.getNodeSet(String codingScheme,
CodingSchemeVersionOrTag versionOrTag, LocalNameList entityTypes);
```

Once established, the CodedNodeSet can be further restricted using the various 'restrict' methods in the **CodedNodeSet** API.

- For examples, see the LexEVS Example classes
    - org.LexGrid.LexBIG.example.SoundsLike
    - org.LexGrid.LexBIG.example.FindCodesForDescription

---

getConceptWithPropertyMatching

See example above. For Property-specific matching, see the following method in the **CodedNodeSet** API

- restrictToProperties

This will ensure that each of the results will have at least one Propety that matches the supplied criteria.

---

isSubConcept

In LexEVS, use the **CodedNodeGraph** API to find the immediate relations of a Concept. For instance:

```
 CodedNodeGraph cng = lexevsService.getNodeGraph(String codingScheme,
CodingSchemeVersionOrTag versionOrTag, String relationContainerName);
 ResolvedConceptReference[] rcr = cng.resolveAsList(ConceptReference
graphFocus, boolean resolveForward, boolean resolveBackward,
                                                    int
resolveCodedEntryDepth, int resolveAssociationDepth, LocalNameList
propertyNames,
                                                    PropertyType[]
propertyTypes, SortOptionList sortOptions, int
maxToReturn).getResolvedConceptReference();
```

Set the **ConceptReference graphFocus** to the desired code, this will focus the Graph. The check the relationships.

Alternatively, use the **CodedNodeGraph** API method 'areCodesRelated'

```
 Boolean areCodesRelated(NameAndValue association, ConceptReference
sourceCode, ConceptReference targetCode, boolean directOnly)
```

---

isRetired

Use the **LexBIGServiceConvenienceMethods** API method 'isCodeRetired' method.

---

getDescendants

Use the LexEVS **HistoryService** API

Obtain a the HistoryService API as follows:

```
 HistoryService historySvc = lexevsService.getHistoryService
(StringcodingSchemeName);
```

To get the decendents of a given code, use the 'getDecendants' method:

```
 NCIChangeEventList changeEventList = historySvc.getDescendants
(ConceptReference conceptReference);
```

---

getPropertyValues

Through the ***Entity*** class, all Properties (Presentations, Definitions, etc) are available.

- Entity
- To find the 'Presentations' of a given Entity:

```
   Entity entity = ....;
   Presentation[] presentations = entity.getPresentation();
```

- To find the 'Definitions' of a given Entity:

```
   Entity entity = ....;
   Definition[] definitions = entity.getDefinition();
```

- To find the 'Comments' of a given Entity:

```
   Entity entity = ....;
   Comment[] comments= entity.getComment();
```

- To find the non-classified Properties of a given Entity:

```
   Entity entity = ....;
   Property[] properties = entity.getProperties();
```

---

getAncestors

Use the LexEVS **HistoryService** API

Obtain a the HistoryService API as follows:

```
 HistoryService historySvc = lexevsService.getHistoryService
(StringcodingSchemeName);
```

To get the ancestors of a given code, use the 'getDecendants' method:

```
 NCIChangeEventList changeEventList = historySvc.getAncestors
(ConceptReference conceptReference);
```

---

getSubConcepts

Use the **CodedNodeGraph** API to find the immediate relations of a Concept. For instance:

```
 CodedNodeGraph cng = lexevsService.getNodeGraph(String codingScheme,
CodingSchemeVersionOrTag versionOrTag, String relationContainerName);
 ResolvedConceptReference[] rcr = cng.resolveAsList(ConceptReference
graphFocus, boolean resolveForward, boolean resolveBackward,
                                                    int
resolveCodedEntryDepth, int resolveAssociationDepth, LocalNameList
propertyNames,
                                                    PropertyType[]
propertyTypes, SortOptionList sortOptions, int
maxToReturn).getResolvedConceptReference();
```

Focus the 'graphFocus' on the desired Concept to see relationships from a given Concept.

- For examples, see the LexEVS Example classes
  - org.LexGrid.LexBIG.example.FindRelatedCodes
  - org.LexGrid.LexBIG.example.FindPropsAndAssocForCode

getSuperConcepts

See above 'getSubConcepts' -- setting your resolve direction (boolean resolveForward, boolean resolveBackward) will determine if sub-concepts or super-concepts are resolved.

getPropertiesByConceptCode

To find the Properties of a given code, for example, code 'C1234' in the 'NCI Thesaurus' ontology - first obtain a 'CodedNodeSet' from the 'NCI Thesaurus' ontology:

```
 ResolvedConceptReferenceList cns = lbSvc.getCodingSchemeConcepts("NCI
Thesaurus", null);
```

Next, restrict that to the desired Code ('C1234' in this example):

```
 //First create a ConceptReferenceList to describe the Concept to search for.
 //In this example we use the helper class 'ConvenienceMethods'.
 ConceptReferenceList crefs =
ConvenienceMethods.createConceptReferenceList(new String[]
{ "C1234"}, "NCI Thesaurus");

 //Next, restrice the CodedNodeSet.
 cns.restrictToCodes(crefs);
```

Lastly, resolve the match.

```
 ResolvedConceptReferenceList matches = cns.resolveToList(null, null,
null, 1);
```

To view the Properties, see 'getPropertyValues' above.

- For examples, see the LexEVS Example classes
  - org.LexGrid.LexBIG.example.FindPropsAndAssocForCode

getVocabularyNames

Use the 'LexBIGService' API method 'getSupportedCodingSchemes' - and extract the Names (local name, registered name, etc...) as needed.

getAllVocabularies

Use the 'LexBIGService' API method 'getSupportedCodingSchemes'.

---

getVocabularyByName

Use 'LexBIGService' API method 'resolveCodingScheme'.

---

getVocabularyVersion

Use 'LexBIGService' API method 'resolveCodingScheme' and extract the 'representsVersion' attribute from the Resulting CodingScheme.

---

getConceptEditAction

Use the LexEVS **HistoryService** API

Obtain a the HistoryService API as follows:

```
 HistoryService historySvc = lexevsService.getHistoryService
(StringcodingSchemeName);
```

To get the 'EditActionList' of a given code, use the 'getEditActionList' method:

```
 NCIChangeEventList changeEventList = historySvc.getEditActionList(ConceptReference conceptReference,
CodingSchemeVersion codingSchemeVersion);
```

---

getRootConcepts

Use 'LexBIGServiceConvenienceMethods' API method 'getHierarchyRoots' or 'getHierarchyRootSet'

---

searchSourceByCode

Use 'CodedNodeSet' API - adding a 'restrictToCodes' restriction.

To find the a given code, for example, code 'C1234' in the 'NCI Thesaurus' ontology - first obtain a 'CodedNodeSet' from the 'NCI Thesaurus' ontology:

```
 ResolvedConceptReferenceList cns = lbSvc.getCodingSchemeConcepts("NCI
Thesaurus", null);
```

Next, restrict that to the desired Code ('C1234' in this example):

```
 //First create a ConceptReferenceList to describe the Concept to
search for.
 //In this example we use the helper class 'ConvenienceMethods'.
 ConceptReferenceList crefs = ConvenienceMethods.createConceptReferenceList
(new String[]
{ "C1234"}, "NCI Thesaurus");

 //Next, restrice the CodedNodeSet.
 cns.restrictToCodes(crefs);
```

Lastly, resolve the match.

```
 ResolvedConceptReferenceList matches = cns.resolveToList(null, null,
null, 1);
```

---

searchSourceByAtomCode

In the NCI MetaThesaurus, fidning the 'source' of an 'Atom' is equivalent to finding the 'source' of a given Property of an Entity. Each CUI (which is equivalent to an Entity in LexEVS) may contain several Presentation Properties (Atoms or AUI's of that CUI). Each of these Presentation Properties is Qualified by a 'source-code' Qualifier, which reflects the code of this Atom in its original source, and a 'source' qualifier, which states the source itself that this Atom came from.

---

getMetaConceptNameByCode

To find the Properties of a given code, for example, code 'C1234567' in the 'NCI MetaThesaurus' ontology - first obtain a 'CodedNodeSet' from the 'NCI MetaThesaurus' ontology:

```
 ResolvedConceptReferenceList cns = lbSvc.getCodingSchemeConcepts("NCI
MetaThesaurus", null);
```

Next, restrict that to the desired Code ('C1234567' in this example):

```
 //First create a ConceptReferenceList to describe the Concept to
search for.
 //In this example we use the helper class 'ConvenienceMethods'.
 ConceptReferenceList crefs =
ConvenienceMethods.createConceptReferenceList(new String[]
{ "C1234567"}, "NCI MetaThesaurus");

 //Next, restrice the CodedNodeSet.
 cns.restrictToCodes(crefs);
```

Lastly, resolve the match.

```
 ResolvedConceptReferenceList matches = cns.resolveToList(null, null,
null, 1);
```

To see the name of the code, use 'getEntityDescription' on the resulting ResolvedConceptReference. The 'EntityDescription' will always be equal to the Preferred Presentation in the Default Language.

---

getMetaSources

Use the 'LexBIGService' API method 'getMappings'. Extract for this the Supported Association Qualifiers using the 'getSupportedSource' method. **Note:** This can be applied to any Coding Scheme, not just the NCI MetaThesaurus.

---

getChildren

Use the 'CodedNodeGraph' API.

- For examples, see the LexEVS Example classes
  - org.LexGrid.LexBIG.example.FindRelatedCodes
  - org.LexGrid.LexBIG.example.FindPropsAndAssocForCode

---

getParent

Use the 'CodedNodeGraph' API.

- For examples, see the LexEVS Example classes
  - org.LexGrid.LexBIG.example.FindRelatedCodes
  - org.LexGrid.LexBIG.example.FindPropsAndAssocForCode

---

getBroaderConcepts

Use the **CodedNodeGraph** API to find the immediate relations of a Concept. Resolve forward or backwards based on the hierarchy structure of the ontology.

- For examples, see the LexEVS Example classes
  - org.LexGrid.LexBIG.example.FindRelatedCodes
  - org.LexGrid.LexBIG.example.FindPropsAndAssocForCode

---

getNarrowerConcepts

Use the **CodedNodeGraph** API to find the immediate relations of a Concept. Resolve forward or backwards based on the hierarchy structure of the ontology.

- For examples, see the LexEVS Example classes
  - org.LexGrid.LexBIG.example.FindRelatedCodes
  - org.LexGrid.LexBIG.example.FindPropsAndAssocForCode

---

getRelatedConcepts

Use the **CodedNodeGraph** API to find the immediate relations of a Concept.

- For examples, see the LexEVS Example classes

- org.LexGrid.LexBIG.example.FindRelatedCodes
- org.LexGrid.LexBIG.example.FindPropsAndAssocForCode

---

containsInverseRole

Use the **LexBIGServiceConvenienceMethods** API method 'isReverseName' method.

---

containsRole

Use the **LexBIGServiceConvenienceMethods** API method 'isForwardName' method.

---

getAllAssociationTypes

Use the 'LexBIGService' API method 'getMappings'. Extract for this the Supported Associations using the 'getSupportedAssociation' method.

---

getAllConceptAssociationQualifierTypes

Use the 'LexBIGService' API method 'getMappings'. Extract for this the Supported Association Qualifiers using the 'getSupportedAssociationQualifier' method.

---

getAllConceptAssociationTypes

Use the 'LexBIGService' API method 'getMappings'. Extract for this the Supported Associations using the 'getSupportedAssociation' method.

---

getAllConceptPropertyQualifierTypes

Use the 'LexBIGService' API method 'getMappings'. Extract for this the Supported Property Qualifiers using the 'getSupportedPropertyQualifier' method.

---

getAllConceptPropertyTypes

Use the 'LexBIGService' API method 'getMappings'. Extract for this the Supported Properties using the 'getSupportedProperty' method.

---

getAllLicenses

Use the 'LexBIGService' API method 'resolveCodingSchemeCopyright'. To get the Copyright for every loaded ontology, do this for each one.

---

getAllPropertyTypes

Use the 'LexBIGService' API method 'getMappings'. Extract for this the Supported Properties using the 'getSupportedProperty' method.

---

getAllQualifierTypes

Use the 'LexBIGService' API method 'getMappings'. Extract for this the Supported Property Qualifiers using the 'getSupportedPropertyQualifier' method.

---

getAllRoleNames

Use the 'LexBIGService' API method 'resolveCodingScheme'. Once the CodingScheme Object is obtained, use the method 'getRelations'.

---

getAllSubConceptCodes

Use the **CodedNodeGraph** API to find the immediate relations of a Concept.

- For examples, see the LexEVS Example classes
  - org.LexGrid.LexBIG.example.FindRelatedCodes
  - org.LexGrid.LexBIG.example.FindPropsAndAssocForCode

---

getAllSynonymTypes

Use the 'LexBIGService' API method 'getMappings'. Extract for this the Supported Associations using the 'getSupportedAssociation' method. NOTE: Different ontologies may describe their 'Synonym' relations differently.

---

getAllTermAssociationQualifierTypes

Use the 'LexBIGService' API method 'getMappings'. Extract for this the Supported Association Qualifiers using the 'getSupportedAssociationQualifier' method.

---

getAllTermPropertyQualifierTypes

Use the 'LexBIGService' API method 'getMappings'. Extract for this the Supported Property Qualifiers using the 'getSupportedPropertyQualifier' method.

---

getAllTermPropertyTypes

Use the 'LexBIGService' API method 'getMappings'. Extract for this the Supported Property using the 'getSupportedProperty' method.

---

getParentConcepts

Use the **CodedNodeGraph** API to find the immediate relations of a Concept.

- For examples, see the LexEVS Example classes
  - org.LexGrid.LexBIG.example.FindRelatedCodes
  - org.LexGrid.LexBIG.example.FindPropsAndAssocForCode

---

getChildConcepts

Use the **CodedNodeGraph** API to find the immediate relations of a Concept.

- For examples, see the LexEVS Example classes
  - org.LexGrid.LexBIG.example.FindRelatedCodes
  - org.LexGrid.LexBIG.example.FindPropsAndAssocForCode

---

hasParents

Use the **CodedNodeGraph** API to find the immediate relations of a Concept.

- For examples, see the LexEVS Example classes
  - org.LexGrid.LexBIG.example.FindRelatedCodes
  - org.LexGrid.LexBIG.example.FindPropsAndAssocForCode

---

hasChildren

Use the **CodedNodeGraph** API to find the immediate relations of a Concept.

- For examples, see the LexEVS Example classes
  - org.LexGrid.LexBIG.example.FindRelatedCodes
  - org.LexGrid.LexBIG.example.FindPropsAndAssocForCode

---

getDescLogicConcept

A 'DescLogicConcept' can be thought of as an 'Entity' in LexEVS. To obtain an Entity, use the **CodedNodeSet** API, restricting the query as necessary.

For instance, a 'DescLogicConcept' with a code of 'C1234' can be queried for by the example below. The example will return a ResolvedConceptReference and ultimately an Entity, but is functionally the same as searching for a DescLogicConcept.

To find the a given code, for example, code 'C1234' in the 'NCI Thesaurus' ontology - first obtain a 'CodedNodeSet' from the 'NCI Thesaurus' ontology:

```
 ResolvedConceptReferenceList cns = lbSvc.getCodingSchemeConcepts("NCI
Thesaurus", null);
```

Next, restrict that to the desired Code ('C1234' in this example):

```
 //First create a ConceptReferenceList to describe the Concept to
search for.
 //In this example we use the helper class 'ConvenienceMethods'.
 ConceptReferenceList crefs =
ConvenienceMethods.createConceptReferenceList(new String[]
{ "C1234"}, "NCI Thesaurus");

 //Next, restrice the CodedNodeSet.
 cns.restrictToCodes(crefs);
```

Lastly, resolve the match.

```
 ResolvedConceptReferenceList matches = cns.resolveToList(null, null,
null, 1);
```

---

getHistoryRecords

Use the **HistoryService** API 'getBaselines' and specify the required Data range.

Obtain a the HistoryService API as follows:

```
 HistoryService historySvc = lexevsService.getHistoryService
(StringcodingSchemeName);
```

Use the 'getBaselines' method:

```
 SystemReleaseList systemReleaseList = historySvc.getBaselines(Date
releasedAfter, Date releasedBefore);
```

---

getHistoryStartDate

Use the **HistoryService** API method 'getEarliestBaseline';

Obtain a the HistoryService API as follows:

```
 HistoryService historySvc = lexevsService.getHistoryService
(StringcodingSchemeName);
```

Use the 'getEarliestBaseline' method:

```
SystemRelease systemRelease = historySvc.getEarliestBaseline();
```

---

getHistoryEndDate

Use the **HistoryService** API method 'getLatestBaseline';

Obtain a the HistoryService API as follows:

```
 HistoryService historySvc = lexevsService.getHistoryService
(StringcodingSchemeName);
```

Use the 'getLatestBaseline' method:

```
SystemRelease systemRelease = historySvc.getLatestBaseline();
```

---

getCodeActionChildren

Use the **HistoryService** API method 'getDescendants'; Obtain a the HistoryService API as follows:

```
 HistoryService historySvc = lexevsService.getHistoryService
(StringcodingSchemeName);
```

Use the 'getDescendants' method:

```
 NCIChangeEventList changeEventList = historySvc.getDescendants
(ConceptReference conceptReference);
```

---

getCodeActionParents

Use the **HistoryService** API method 'getAncestors';

Obtain a the HistoryService API as follows:

```
 HistoryService historySvc = lexevsService.getHistoryService
(StringcodingSchemeName);
```

Use the 'getDescendants' method:

```
 NCIChangeEventList changeEventList = historySvc.getAncestors
(ConceptReference conceptReference);
```

---

getAssociationCollectionbyCode

Use the **CodedNodeGraph** API to find the immediate relations of a Concept. For instance:

```
 CodedNodeGraph cng = lexevsService.getNodeGraph(String codingScheme,
CodingSchemeVersionOrTag versionOrTag, String relationContainerName);
 ResolvedConceptReference[] rcr = cng.resolveAsList(ConceptReference
graphFocus, boolean resolveForward, boolean resolveBackward,
                                                    int
resolveCodedEntryDepth, int resolveAssociationDepth, LocalNameList
propertyNames,
                                                    PropertyType[]
propertyTypes, SortOptionList sortOptions, int
maxToReturn).getResolvedConceptReference();
```

Focus the 'graphFocus' on the desired Concept to see relationships from a given Concept.

Once focused and resolved, use the 'getSourceOf' or 'getTargetOf' methods on the ResolvedConceptReference to find the Associations of a given Code.

- For examples, see the LexEVS Example classes
  - org.LexGrid.LexBIG.example.FindRelatedCodes
  - org.LexGrid.LexBIG.example.FindPropsAndAssocForCode

---

getSemanticTypeCollectionbyCui

Use 'CodedNodeSet' API - adding a 'restrictToCodes' restriction.

Note that a CUI is simply a reference to a Code in the NCI MetaThesaurus ontology.

To find the a given code, for example, code 'C1234' in the 'NCI Thesaurus' ontology - first obtain a 'CodedNodeSet' from the 'NCI Thesaurus' ontology:

```
 ResolvedConceptReferenceList cns = lbSvc.getCodingSchemeConcepts("NCI
Thesaurus", null);
```

Next, restrict that to the desired Code ('C1234' in this example):

```
 //First create a ConceptReferenceList to describe the Concept to
search for.
 //In this example we use the helper class 'ConvenienceMethods'.
 ConceptReferenceList crefs =
ConvenienceMethods.createConceptReferenceList(new String[]
{ "C1234"}, "NCI Thesaurus");

 //Next, restrice the CodedNodeSet.
 cns.restrictToCodes(crefs);
```

Lastly, resolve the match.

```
 ResolvedConceptReferenceList matches = cns.resolveToList(null, null,
null, 1);
```

Once the ResolvedConceptReference has been obtained, extract the desired Properties and inspect the Qualifiers for the Semantic Type.

---

getQualifierCollectionbyName

Use 'CodedNodeSet' API - adding a 'restrictToCodes' restriction.

Note that a CUI is simply a reference to a Code in the NCI MetaThesaurus ontology.

To find the a given code, for example, code 'C1234' in the 'NCI Thesaurus' ontology - first obtain a 'CodedNodeSet' from the 'NCI Thesaurus' ontology:

```
 ResolvedConceptReferenceList cns = lbSvc.getCodingSchemeConcepts("NCI
Thesaurus", null);
```

Next, restrict that to the desired Code ('C1234' in this example):

```
 //First create a ConceptReferenceList to describe the Concept to
search for.
 //In this example we use the helper class 'ConvenienceMethods'.
 ConceptReferenceList crefs =
ConvenienceMethods.createConceptReferenceList(new String[]
{ "C1234"}, "NCI Thesaurus");

 //Next, restrice the CodedNodeSet.
 cns.restrictToCodes(crefs);
```

Lastly, resolve the match.

```
 ResolvedConceptReferenceList matches = cns.resolveToList(null, null,
null, 1);
```

Once the ResolvedConceptReference has been obtained, extract the desired Properties and inspect the Qualifiers.

NOTE: Associations between codes may also have Qualifiers.

---

getAtomCollectionbyCui

In LexEVS, a NCI MetaThesaurus CUI is represented by an Entity (with the CUI being the code for that Entity). Atoms of that CUI are represented by 'Presentation'(s) of the Entity.

- For examples, see the LexEVS Example classes
    - org.LexGrid.LexBIG.example.FindPropsAndAssocForCode

---

getSynonymCollectionbyCui

Use 'CodedNodeGraph' API - restricting to Synonym Associations. Note: Each ontology may describe their Synonym Associations differently.

- For examples, see the LexEVS Example classes

- org.LexGrid.LexBIG.example.FindRelatedCodes
- org.LexGrid.LexBIG.example.FindPropsAndAssocForCode

---

getSourceCollectionbyCui

Use 'CodedNodeSet' API - adding a 'restrictToCodes' restriction.

Note that a CUI is simply a reference to a Code in the NCI MetaThesaurus ontology.

To find the a given code, for example, code 'C1234' in the 'NCI Thesaurus' ontology - first obtain a 'CodedNodeSet' from the 'NCI Thesaurus' ontology:

```
 ResolvedConceptReferenceList cns = lbSvc.getCodingSchemeConcepts("NCI
Thesaurus", null);
```

Next, restrict that to the desired Code ('C1234' in this example):

```
 //First create a ConceptReferenceList to describe the Concept to
search for.
 //In this example we use the helper class 'ConvenienceMethods'.
 ConceptReferenceList crefs =
ConvenienceMethods.createConceptReferenceList(new String[]
{ "C1234"}, "NCI Thesaurus");

 //Next, restrice the CodedNodeSet.
 cns.restrictToCodes(crefs);
```

Lastly, resolve the match.

```
 ResolvedConceptReferenceList matches = cns.resolveToList(null, null,
null, 1);
```

Once the ResolvedConceptReference has been obtained, extract the desired Properties and inspect Source using 'getSource'.

---

getSemanticTypeCollectionbyCui

Use 'CodedNodeSet' API - adding a 'restrictToCodes' restriction.

Note that a CUI is simply a reference to a Code in the NCI MetaThesaurus ontology.

To find the a given code, for example, code 'C1234' in the 'NCI Thesaurus' ontology - first obtain a 'CodedNodeSet' from the 'NCI Thesaurus' ontology:

```
 ResolvedConceptReferenceList cns = lbSvc.getCodingSchemeConcepts("NCI
Thesaurus", null);
```

Next, restrict that to the desired Code ('C1234' in this example):

```
 //First create a ConceptReferenceList to describe the Concept to
search for.
 //In this example we use the helper class 'ConvenienceMethods'.
 ConceptReferenceList crefs =
ConvenienceMethods.createConceptReferenceList(new String[]
{ "C1234"}, "NCI Thesaurus");

 //Next, restrice the CodedNodeSet.
 cns.restrictToCodes(crefs);
```

Lastly, resolve the match.

```
 ResolvedConceptReferenceList matches = cns.resolveToList(null, null,
null, 1);
```

Once the ResolvedConceptReference has been obtained, extract the desired Properties and inspect the Semantic Types. Semantic Types are held as Qualifiers to the Properties of an Entity.

---

getSourcebyDefinition

Use 'CodedNodeSet' API - adding a 'restrictToCodes' restriction.

To find the a given code, for example, code 'C1234' in the 'NCI Thesaurus' ontology - first obtain a 'CodedNodeSet' from the 'NCI Thesaurus' ontology:

```
 ResolvedConceptReferenceList cns = lbSvc.getCodingSchemeConcepts("NCI
Thesaurus", null);
```

Next, restrict that to the desired Code ('C1234' in this example):

```
 //First create a ConceptReferenceList to describe the Concept to
search for.
 //In this example we use the helper class 'ConvenienceMethods'.
 ConceptReferenceList crefs =
ConvenienceMethods.createConceptReferenceList(new String[]
{ "C1234"}, "NCI Thesaurus");

 //Next, restrice the CodedNodeSet.
 cns.restrictToCodes(crefs);
```

Lastly, resolve the match.

```
 ResolvedConceptReferenceList matches = cns.resolveToList(null, null,
null, 1);
```

Once the ResolvedConceptReference has been obtained, using 'getDefinition', extract the Definition Collection from the Entity. Then extract the source using 'getSource'

---

getDefinitionCollectionbyCui

Use 'CodedNodeSet' API - adding a 'restrictToCodes' restriction.

To find the a given code, for example, code 'C1234' in the 'NCI Thesaurus' ontology - first obtain a 'CodedNodeSet' from the 'NCI Thesaurus' ontology:

```
 ResolvedConceptReferenceList cns = lbSvc.getCodingSchemeConcepts
("NCI Thesaurus", null);
```

Next, restrict that to the desired Code ('C1234' in this example):

```
 //First create a ConceptReferenceList to describe the Concept to search for.
 //In this example we use the helper class 'ConvenienceMethods'.
 ConceptReferenceList crefs = ConvenienceMethods.createConceptReferenceList
(new String[]
{ "C1234"}, "NCI Thesaurus");

 //Next, restrice the CodedNodeSet.
 cns.restrictToCodes(crefs);
```

Lastly, resolve the match.

```
 ResolvedConceptReferenceList matches = cns.resolveToList(null, null,
null, 1);
```

Once the ResolvedConceptReference has been obtained, use the 'getReferencedEntry' method to obtain the actual Entity. Using the 'getDefinition', extract the Definition Collection from the Entity.

---

getPropertyCollectionbyName

To find the Properties of a given code, for example, a code with a name of 'Heart' in the 'NCI Thesaurus' ontology - first obtain a 'CodedNodeSet' from the 'NCI Thesaurus' ontology:

```
 ResolvedConceptReferenceList cns = lbSvc.getCodingSchemeConcepts
("NCI Thesaurus", null);
```

Next, restrict that to the desired Code Name ('Heart' in this example): **Note:** To match the String 'Heart' exactly, use the search algorithm 'exactMatch'.

```
//Next, restrice the CodedNodeSet.
  cns = cns.restrictToMatchingDesignations("Heart", null, "exactMatch", null);
```

Lastly, resolve the match.

```
ResolvedConceptReferenceList matches = cns.resolveToList(null, null, null, 1);
```

To view the Properties, see 'getPropertyValues' above.

- For examples, see the LexEVS Example classes
    - org.LexGrid.LexBIG.example.FindPropsAndAssocForCode

---

getPropertyCollectionbyCode

To find the Properties of a given code, for example, code 'C1234' in the 'NCI Thesaurus' ontology - first obtain a 'CodedNodeSet' from the 'NCI Thesaurus' ontology:

```
 ResolvedConceptReferenceList cns = lbSvc.getCodingSchemeConcepts
("NCI Thesaurus", null);
```

Next, restrict that to the desired Code ('C1234' in this example):

```
 //First create a ConceptReferenceList to describe the Concept to search for.
 //In this example we use the helper class 'ConvenienceMethods'.
 ConceptReferenceList crefs = ConvenienceMethods.createConceptReferenceList
(new String[]
{ "C1234"}, "NCI Thesaurus");

 //Next, restrice the CodedNodeSet.
 cns.restrictToCodes(crefs);
```

Lastly, resolve the match.

```
 ResolvedConceptReferenceList matches = cns.resolveToList(null, null,
null, 1);
```

To view the Properties, see 'getPropertyValues' above.

- For examples, see the LexEVS Example classes
    - org.LexGrid.LexBIG.example.FindPropsAndAssocForCode

---

fetchPropertyCollectionByCodes

See 'getPropertyCollectionbyCode;

---

getRoleCollectionbyCode

Use the **CodedNodeGraph** API to find the immediate relations of a Concept.

If a desired Relations Collection is requested (for example, 'roles'), the 'relationContainerName' may be restricted. For instance:

```
 //Restrict the 'relationContainerName' to the desired container.
NOTE: These containers are
 //ontology specific--each ontology defines its own relation container names.
 CodedNodeGraph cng = lexevsService.getNodeGraph(String codingScheme,
CodingSchemeVersionOrTag versionOrTag, String relationContainerName);
 ResolvedConceptReference[] rcr = cng.resolveAsList(ConceptReference
graphFocus, boolean resolveForward, boolean resolveBackward,
                                           int resolveCodedEntryDepth, int
resolveAssociationDepth, LocalNameList propertyNames,
                                          PropertyType[] propertyTypes,
SortOptionList sortOptions, int maxToReturn).getResolvedConceptReference();
```

Focus the 'graphFocus' on the desired Concept to see relationships from a given Concept.

Once focused and resolved, use the 'getSourceOf' or 'getTargetOf' methods on the ResolvedConceptReference to find the Associations of a given Code.

- For examples, see the LexEVS Example classes
    - org.LexGrid.LexBIG.example.FindRelatedCodes
    - org.LexGrid.LexBIG.example.FindPropsAndAssocForCode

---

getInverseRoleCollectionbyCode

Use the **CodedNodeGraph** API to find the immediate relations of a Concept. To find the Inverse Roles, restricting the Relations Container may be necessary (see getRoleCollectionbyCode above). Depending on how the onotology defines an 'inverse' role or association, these can be restricted as well.

For instance:

```
 CodedNodeGraph cng = lexevsService.getNodeGraph(String codingScheme,
CodingSchemeVersionOrTag versionOrTag, String relationContainerName);
 ResolvedConceptReference[] rcr = cng.resolveAsList(ConceptReference
graphFocus, boolean resolveForward, boolean resolveBackward,
                                         int resolveCodedEntryDepth, int
resolveAssociationDepth, LocalNameList propertyNames,
                                         PropertyType[] propertyTypes,
SortOptionList sortOptions, int maxToReturn).getResolvedConceptReference();
```

Focus the 'graphFocus' on the desired Concept to see relationships from a given Concept.

Once focused and resolved, use the 'getSourceOf' or 'getTargetOf' methods on the ResolvedConceptReference to find the Associations of a given Code.

- For examples, see the LexEVS Example classes
    - org.LexGrid.LexBIG.example.FindRelatedCodes
    - org.LexGrid.LexBIG.example.FindPropsAndAssocForCode

---

getInverseAssociationCollectionbyCode

Use the **CodedNodeGraph** API to find the immediate relations of a Concept.

To find the Inverse Collections, restricting the Relations Container may be necessary (see getInverseRoleCollectionbyCode above). Depending on how the onotology defines an 'inverse' role or association, these can be restricted as well.

For instance:

```
 CodedNodeGraph cng = lexevsService.getNodeGraph(String codingScheme,
CodingSchemeVersionOrTag versionOrTag, String relationContainerName);
 ResolvedConceptReference[] rcr = cng.resolveAsList(ConceptReference
graphFocus, boolean resolveForward, boolean resolveBackward,
                                       int resolveCodedEntryDepth, int
resolveAssociationDepth, LocalNameList propertyNames,
                                       PropertyType[] propertyTypes,
SortOptionList sortOptions, int maxToReturn).getResolvedConceptReference();
```

Focus the 'graphFocus' on the desired Concept to see relationships from a given Concept.

Once focused and resolved, use the 'getSourceOf' or 'getTargetOf' methods on the ResolvedConceptReference to find the Associations of a given Code.

- For examples, see the LexEVS Example classes
    - org.LexGrid.LexBIG.example.FindRelatedCodes
    - org.LexGrid.LexBIG.example.FindPropsAndAssocForCode

---

getHasParentsbyCode

Either:

- Use the 'LexBIGServiceConvenienceMethods' API method 'getHierarchyLevelPrev'
- Use the CodedNodeGraph API to resolve the immediate Associations of a given code, and check if they exist.

---

getHasChildrenbyCode

Either:

- Use the 'LexBIGServiceConvenienceMethods' API method 'getHierarchyLevelNext'
- Use the CodedNodeGraph API to resolve the immediate Associations of a given code, and check if they exist.

---

getIsRetiredbyCode

Use the 'LexBIGServiceConvenienceMethods' API method 'isCodeRetired'

---

getLocalNames

Use the 'LexBIGService' API method 'getSupportedCodingSchemes' - and extract the Names (local name, registered name, etc...) as needed.

OWL Loader Enhancements

Substantial changes have been implemented in LexEVS 5.0 during the conversion of the OWL loader. The NCI OWL loader has been decommissioned and replaced with a more generic Protégé OWL loader. All effort has been made to ensure that no previous functionally has been lost during this transition. Priority was given to maintaining existing functionality while improving the OWL loader.

Enhancements and changes made to the OWL loader:

- Improve OWL model footprint by upgrading to latest Protégé (3.4 w/improved support for database streaming)

- Provide ability to enable use the Protégé DB support (Protégé database will serve as cache while we build the LexEVS model from OWL)

- Add support for NCI-based complex props (processing of XML fragments)

- Add support for preferences

- Add support for manifest

- Add support to split role and associations (consider splitat relation container level as done by NCIT loader)

- When resolving IndividualProperties, changed casting from 'OWLNamedClass' to super interface 'RDFSNamedClass'.

- When determining the Entity Id, there were some spots that were using the 'getBrowserText()' method on the 'OWLNamedClass' class. The 'getBrowserText()' was intended to give Protege a nice display string -- but in order to get the id we wanted we want to use the 'getLocalName()' method.

- Now we do not create 'domain' and 'range' associations if there is no target of the association.

- When processing OWLObjectProperties, changed casting from 'OWLObjectProperty' to super interface 'RDFProperty'

- When processing Instances, changed casting from 'OWLNamedClass' to super interface 'RDFSClass', and 'OWLIndividual' to super interface 'RDFResource'.

- When determining the the Entity code during load of an association, we now parse the string based on a colon OR hash symbol.
  For example:
      http://someNamespace.org:C12345 would resolve to 'C12345'
  and
      http://someNamespace.org#C12345 would also resolve to 'C12345'
  We used to process only the colon.

- The isDefined() property is now set on created entities.

- Removed the following OWL preferences - dataTypeNameBoolean, associatonNameHasType, and associationNameHasTypeURN.

- Annotation properties are now stored in terms of presentation/comments.

- Manifest supports forward and reverse association names.

- The codedNodeSet restriction added to restrict lucene-based queries.

- RDF local names are used instead of 'textualPresentation' and 'comment' property names.
- Updated SupportedCodingScheme.isImported set to "true" as default.
- The previous NCI Loader and related dependencies have been removed.
- Non-concept entities by EMF EntityService are being handled correctly.
- Memory profiling options 0 and 3 removed from external interfaces.
- Instances are streamed under the enhanced memory profile options.
- Update made to properly store/retrieve the entity type in lucene indexing.
- Update made for use of association code as the 'id' in supported associations are consistent with hierarchy and general API declarations that work with associations (same for GUI interfaces).
- Loader preference "CreateConceptForObjectProp" is added. It controls whether concept entities are created for object properties defined in the OWL source. The default is false.
- Loader preference "DatatypePropSwitch" is added. It controls how data type properties are converted to components of the LexGrid model. If 'association' is specified, each data type property is recorded in LexGrid as an entity-to-entity relationship. If 'conceptProperty' is specified, traditional LexGrid properties are created and assigned directly to new entities. If 'both' is specified, both entity relationships and standard LexGrid entity properties are generated. The default is 'both'.
- Namespace prefixes from the owl source will be registered as supportedNamespace instead of supportedCodingScheme.
- Copyright information is no more hardcoded into the loader. The copyright should be specified in the manifest.
- The Loader will not hardcode the codingschemeName as NCI_Thesaurus. Manifest option has to be used to change it.
- Associations have been distributed among two containers (association and roles)
- Concepts will not have properties "NCI-preferred-term" and "CONCEPT-NAME". How ever, required properties can be introduced by using preferences option "PrioritizedPresentationNames", "PrioritizedDefinitionNames" and "PrioritizedCommentNames".
- Complex properties are not handled by default by the owl loader. Use preference option "ProcessComplexProps" to enable it.
- The restrictions an equivalent class are connected to the parent concept as it was done in NCI-OWL loader. However, if strict owl implementation is required (restrictions an equivalent class not connected to the parent concept) , use the preference option "StrictOWLImplementation"
- Deprecated concepts issue has been resolved by comparing "rdfResource.getRDFType().getName()" with the literal.
- Root node identification: If the preference option "MatchRootName" is specified, the root nodes are identified from it. Otherwise root node is identified from the protege owl api.
- The associations "hasInstance", "hasDomain", "hasRange", "hasDatatype" and "hasDatatypeValue" has

been renamed to "instance", "domain", "range", "datatype", "datatypevalue" respectively.

- LexGrid data streaming options have been introduced for effective memory utilizations. Users can choose the memory safe modes based on the requirements.

Deployment Artifacts

LexEVS Components

LexEVS 5.0 deployment artifacts have been completely refactored. The usage and installation of these components is documented in the LexEVS 5.0 Installation Guide.

Filename	Description
LexEVS_50_localRuntime.jar	LexEVS Local Runtime Environment - Includes the LexBIG API, loaders, and administrative utilities developed as part of the LexEVS project.
LexEVS_50_localRuntime_dependencies.jar	LexEVS Local Runtime 3rd Party Dependencies - Includes code from other open source projects required by the LexEVS Java API, packaged as a single jar for convenient deployment.
LexEVS_50_client.jar	LexEVS Java Client - Enables Java programs to establish a connection to LexEVS distributed, web or caGrid runtime services.
LexEVS_50_clientDependencies.zip	LexEVS Java Client 3rd Party Dependencies - Contains all code required by the LexEVS Java Client.
LexEVS_50_webRuntime_tomcat.zip LexEVS_50_webRuntime_jboss.zip	LexEVS Web-Enabled Runtime Environment - Includes Java runtime and dependencies, Java distributed API, and caCORE SDK-generated services. Can be deployed to Apache Tomcat or JBoss containers. Each zip file contains *lexevsapi50.war* file.
LexEVS_50_caGRIDServices_analytical_tomcat.zip LexEVS_50_caGRIDServices_analytical_jboss.zip	LexEVS caGrid Analytical Services - Includes analytic caGrid services working in terms of the LexGrid model and LexBIG API, resectively. Can be deployed to Apache Tomcat or JBoss containers. Each zip file contains *wsrf.war* file.
LexEVS_50_caGRIDServices_data_tomcat.zip LexEVS_50_caGRIDServices_data_jboss.zip	LexEVS caGrid Data Services - Includes data caGrid services working in terms of the LexGrid model and LexBIG API, resectively. Can be deployed to Apache Tomcat or JBoss containers. Each zip file contains *wsrf.war* file.
LexEVS_50_caGridGUI.zip	LexEVS caGrid GUI - Provides a traditional (fat-client) graphical user interface that provides access to basic browse/query functions provided by LexEVS caGrid Services.
LexEVS_50_localRuntimeAndGUI_installer.jar	LexEVS Installer - Automated installer used to unzip user selected components to a user specified directory.

LexEVS_50_Source.zip	LexEVS source code - Contains full source code.
LexEVS_50_caGRIDServices_analytical_client.jar	LexEVS Analytical Grid Services Client - Enables Java programs to establish a connection to LexEVS analytical grid services.
LexEVS_50_caGRIDServices_data_client.jar	LexEVS Data Grid Services Client - Enables Java programs to establish a connection to LexEVS data grid services.
LexEVS_50_caGRIDServices_analytical_client_dependencies.zip	LexEVS Analytical Grid Services 3rd Party Dependencies - Includes code from other open source projects required by the grid services, packaged as a zip for convenient deployment.
LexEVS_50_caGRIDServices_data_client_dependencies.zip	LexEVS Data Grid Services 3rd Party Dependencies - Includes code from other open source projects required by the grid services, packaged as a zip for convenient deployment.
[LexEVS_50_JavaDocs.zip	LexEVS JavaDocs (HTML in ZIP file format)
LexEVS_50_source_readme.txt	LexEVS source readme file - Overview of what is included in the source distribution.
LexEVS_50_releasenotes.html	LexEVS release notes - Overview of resolved issues and enhancements provided in the release.
LexEVS_50_readme.txt	LexEVS readme - Recent information for this release.
LexEVS_50_Example_Code.zip	LexEVS example code - code to demonstrate LexEVS API.