LexEVS API Code Examples

Contents of this Page

LexEVS Code Examples Links

- LexEVS API Code Examples
- LexEVS Java RMI Code Examples
- LexEVS CTS2 Code Examples
- LexEVS REST Code Migration Guide
- LexEVS 6.5.1

Code Snippets

Supply the OWL manifest file URI

Java Code Snippet

```
LexBIGService lbs = new LexBIGServiceImpl();
LexBIGServiceManager lbsm = lbs.getServiceManager(null);
OWL_Loader loader = (OWL_Loader) lbsm.getLoader("OWLLoader");

if (toValidateOnly)
{
    loader.validate(source, manifest, vl);
    System.out.println("VALIDATION SUCCESSFUL");
}
else
{
    loader.load(new File("resources/testData/amino-acid.owl").toURI(),
                new File("resources/testData/aa-manifest.xml").toURI(),true,true);
}
```

Supply the non-OWL manifest file URI

Java Code Snippet

```
// Find the registered extension handling this type of load
LexBIGService lbs = new LexBIGServiceImpl();
LexBIGServiceManager lbsm = lbs.getServiceManager(null);
HL7_Loader loader = (HL7_Loader)lbsm.getLoader (org.LexGrid.LexBIG.Impl.loaders .HL7LoaderImpl.name);

// updated to include manifest
loader.setCodingSchemeManifestURI(manifest);

// updated to include loader preferences
loader.setLoaderPreferences(loaderPrefs);
loader.load(dbPath, stopOnErrors, true);
```

Find the immediate relations of a Concept

Java Code Snippet

```
CodedNodeGraph cng = lexevsService.
               getNodeGraph(codingScheme, versionOrTag, relationContainerName);
ResolvedConceptReference[] rcr = cng.resolveAsList(graphFocus, resolveForward,
               resolveBackward, resolveCodedEntryDepth, resolveAssociationDepth,
               propertyNames, propertyTypes, sortOptions, maxToReturn).
                            getResolvedConceptReference();
```

Find a given code, for example, code 'C1234' in the 'NCI Thesaurus' ontology

Java Code Snippet

```
//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):
ConceptReferenceList crefs = ConvenienceMethods.
                createConceptReferenceList(new String[]{"C1234"}, "NCI Thesaurus");
cns.restrictToCodes(crefs);

//Lastly, resolve the match.
ResolvedConceptReferenceList matches = cns.resolveToList(null, null, null, 1);
```

Concept Resolution

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

Java Code Snippet

```
    // 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 an brief example showing how to access and print some of this metadata:

Java Code Snippet

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

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

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

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

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

Combinatorial Queries

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

Java Code Snippet

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

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

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

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

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

// Indicate to return only the assigned UMLS_CUI and
// textualPresentation properties.
LocalNameList restrictTo =ConvenienceMethods.createLocalNameList(
                new String[] {"UMLS_CUI", "textualPresentation"} );

// Still nothing computed yet.
// Perform the query && resolve the sorted/filtered list with a
// maximum of 6 items returned.
ResolvedConceptReferenceList list = cns.resolveToList(
                sortCriteria, restrictTo, null, 6);
// Print the results
ResolvedConceptReference[] rcr = list.getResolvedConceptReference();
for (ResolvedConceptReference rc : rcr)
{
    System.out.println("Resolved Concept: " + rc.getConceptCode());
}
```

Coded Node Graph Restriction

Java Code Snippet

```
cng.restrictToCodeSystem(org.LexGrid.LexBIG.DataModel.cagrid.CodingSchemeIdentification);
```

Resolved Concept References Iterator

Java Code Snippet

```
while(itr.hasNext){
     ResolvedConceptReference ref = itr.next();
}
```

Request Filter Match

Java Code Snippet

```
filter.match(resolvedConceptReference);
```

Request Filter

Java Code Snippet

```
Filter filter = lbs.getFilter(org.LexGrid.LexBIG.DataModel.cagrid.ExtensionIdentification);
```

Request Sort Compare

Java Code Snippet

```
sort.compare(codedNodeReference1, codedNodeReference2);
```

Request Sort

Java Code Snippet

```
Sort sort = lbs.getSortAlgorithm(org.LexGrid.LexBIG.DataModel.cagrid.ExtensionIdentification);
```

Concept Resolution

Java Code Snippet

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

// 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[], 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");       }
```

Metathesaurus CUI search

Java Code Snippet

```
CodedNodeSet nodeSet = evsService.getNodeSet("NCI MetaThesaurus", null, null);

//Tell the api that you want to get back only the PRESENTATION type properties
CodedNodeSet.PropertyType[] types = new CodedNodeSet.PropertyType[1];
types[0] = CodedNodeSet.PropertyType.PRESENTATION;

//Now create the reference to the CUI you wish to retrieve
ConceptReferenceList crl = new ConceptReferenceList();
crl.addConceptReference(ConvenienceMethods.createConceptReference("C0029045","
NCIMetaThesaurus"));
nodeSet = nodeSet.restrictToCodes(crl);
```

Metathesaurus Source Code Search

Java Code Snippet

```
CodedNodeSet nodeSet = evsService.getNodeSet("NCI MetaThesaurus", null, null);

//Tell the api that you want to get back only the PRESENTATION type properties
CodedNodeSet.PropertyType[] types = new CodedNodeSet.PropertyType[1];
types[0] = CodedNodeSet.PropertyType.PRESENTATION;

//Now create a qualifier list containing the code you wish to search
NameAndValueList qualifierList = new NameAndValueList();
NameAndValue nv = new NameAndValue();
nv.setName("source-code");
nv.setContent("C12438");
qualifierList.addNameAndValue(nv);

//Specify the source code should come from the NCI source
LocalNameList LnL = new LocalNameList();
LnL.addEntry("NCI");


nodeSet = nodeSet.restrictToProperties(null,types,LnL,null, qualifierList);
```

Search by Existence of a Property

This will search for all concepts in the given vocabulary that possess the given property, regardless of the property's value.

Java Code Snippet

```
//Identify the vocabulary to search
CodingSchemeVersionOrTag csvt = new CodingSchemeVersionOrTag();
csvt.setTag("PRODUCTION");
String scheme = "NCI Thesaurus";

CodedNodeSet cns = lbSvc.getCodingSchemeConcepts(scheme, csvt);

//constrain: concept property 'GTEx_CDE' exists
cns = cns.restrictToProperties(ConvenienceMethods.createLocalNameList(new String[] { "GTEx_CDE" }), null);

//now get the concepts
ResolvedConceptReferenceList matches = cns.resolveToList(null, null, null, 50);
```

Search by FULL_SYN with specified source

Java Code Snippet

```
            public void testFullSynBySource(){
                try {
                                                String searchTerm = "gene";
                        CodedNodeSet nodeSet = lbSvc.getNodeSet("HUBt", null, null);

                        //Tell the api that you want to search only the PRESENTATION type
properties
                        CodedNodeSet.PropertyType[] types = new CodedNodeSet.PropertyType[1];
                        types[0] = CodedNodeSet.PropertyType.PRESENTATION;

            //Only want to search properties with a source entry of "CAHUB"
                        LocalNameList sourceLnL = new LocalNameList();
                        sourceLnL.addEntry("CAHUB");

            //Only want to search properties with the property name of "FULL_SYN"
                        LocalNameList propLnL = new LocalNameList();
                        propLnL.addEntry("FULL_SYN");

                        nodeSet = nodeSet.restrictToMatchingProperties(propLnL,types,sourceLnL,null, null,
searchTerm,LBConstants.MatchAlgorithms.contains.name(),null);

                        ResolvedConceptReferenceList rcl = nodeSet.resolveToList(null, null, null, 100);
                        int count = rcl.getResolvedConceptReferenceCount();

            //Now iterate through the returned entities and display the FULL_SYN PT property with source=CAHUB
                        for (int i=0; i<rcl.getResolvedConceptReferenceCount();i++){
                                ResolvedConceptReference rcr = rcl.getResolvedConceptReference(i);
                                Entity entity = rcr.getReferencedEntry();
                                Presentation[] presProps = entity.getPresentation();
                                for(int y=0;y<presProps.length;y++){
                                        Presentation pres = presProps[y];
                                        if(pres.getPropertyName().equals("FULL_SYN")&& pres.
getRepresentationalForm().equals("PT") && pres.getSource(0).getContent().equals("CAHUB")){
                                        System.out.println(pres.getValue().getContent());

                                        }
                                }
                        }

                } catch (IndexOutOfBoundsException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                } catch (LBInvocationException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                } catch (LBParameterException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                } catch (LBException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                }
        }
```

Calculate Transitive Closure

Java Code Snippet

```
    public void getTransitiveClosure() {
        //Calculate the transitive closure (all subconcepts) of a given concept
        ResolvedConceptReferencesIterator iterator = null;
        try {
            String codingSchemeName = "NCI Thesaurus";
            String code = "C20181";
            String associationName = "subClassOf";
            boolean resolveForward = false;
            boolean excludeAnonymous = true;
            CodingSchemeVersionOrTag csvt = new CodingSchemeVersionOrTag();
            csvt.setTag("PRODUCTION");
            ConceptReference graphFocus = new ConceptReference();
            graphFocus.setConceptCode(code);
            CodedNodeGraph cng = lbSvc.getNodeGraph(codingSchemeName, csvt, null);
            NameAndValueList asso_list =
                    Constructors.createNameAndValueList(new String[] { associationName }, null);
            cng = cng.restrictToAssociations(asso_list, null);
            boolean resolveBackward = false;
            if (!resolveForward) {
                resolveBackward = true;
            }
            int resolveAssociationDepth = -1;
            int maxReturns = -1;
            CodedNodeSet cns = cng.toNodeList(graphFocus, resolveForward, resolveBackward,
                    resolveAssociationDepth, maxReturns);
            if (excludeAnonymous) {
                CodedNodeSet.AnonymousOption restrictToAnonymous = CodedNodeSet.AnonymousOption.
NON_ANONYMOUS_ONLY;
                cns = cns.restrictToAnonymous(restrictToAnonymous);
            }
            iterator = cns.resolve(null, null, null, null, false);
            while (iterator.hasNext()){
                ResolvedConceptReference rcr = iterator.next();
                String codeReturn = rcr.getCode();
                System.out.println(codeReturn);
            }
        } catch (Exception ex) {
            ex.printStackTrace();
        }
    }
```

Fetch Definition Source based on Concept Code

Java Code Snippet

```
        public void testDefinitionSource(){

                String searchTerm = "C12435";
                String vocabName = "NCI_Thesaurus";
                String vocabTag = "PRODUCTION";
                ConceptReference cref = new ConceptReference();
                cref.setConceptCode(searchTerm);
                ConceptReferenceList ncrl = new ConceptReferenceList();
                ncrl.addConceptReference(cref);

                try {
            CodingSchemeVersionOrTag cvt = new CodingSchemeVersionOrTag();
                        cvt.setTag(vocabTag);
                        CodedNodeSet nodes = lbSvc.getNodeSet(vocabName, cvt, null);
                        nodes = nodes.restrictToCodes(ncrl);
                        ResolvedConceptReferenceList crl = nodes.resolveToList(null, null,
                                        null, 20);
                        //codes are unique in NCIt, so you will only get one entity in the list
                        Entity concept = crl.getResolvedConceptReference(0).getEntity();
                        Definition[] defs = concept.getDefinition();
                        for (Definition def : defs) {
                                //Each definition in NCIt will only have one source.
                                Source[] sources = def.getSource();
                                Source defSource = sources[0];
                                String source = defSource.getContent();
                                System.out.println("Definition source "+ source);
                        }

                } catch (Exception ex) {
                        System.out.println(ex);
                }
        }
```

Examine History for NCI Thesaurus or NCI Metathesaurus

```
        public void testHistory() {
                String NCIt = "NCI Thesaurus";
                String NCIm = "NCI Metathesaurus";
                try {
                        System.out.println("************************");
                        System.out.println("Testing History");
                        System.out.println("************************");
                        HistoryService hs = lbSvc.getHistoryService(NCIt);
                        // Test getBaselines
                        SystemReleaseList srl = hs.getBaselines(null, null);
                        if (srl.getSystemReleaseCount() > 10) {
                                System.out
                                .println("Success : NCI Thesaurus history baselines retrieved");
                        } else {
                                System.out
                                .println("FAILURE : NCI Thesaurus history baselines not found");
                        }
                        //The hold is just a bin to demonstrate the different calls that can be made.
                        String hold = srl.getSystemRelease(0).getReleaseURI();
                        hold = srl.getSystemRelease(28).getReleaseURI();
                        SystemRelease sr = hs.getEarliestBaseline();
                        hold = sr.getEntityDescription().getContent();
                        long time = sr.getReleaseDate().getTime();
                        CodingSchemeVersion csv = hs.getConceptCreationVersion(Constructors.
createConceptReference("C49239", null));
                        hold = csv.getReleaseURN();
                        hold = csv.getVersion();
                        time = csv.getVersionDate().getTime();
```

```
                        hold = csv.getEntityDescription().getContent();


                        // Test get edit actions
                        NCIChangeEventList ncel = hs.getEditActionList(
                                        Constructors.createConceptReference("C7696", null), null,
                                        null);
                        if (ncel.getEntryCount() > 5) {
                                System.out
                                .println("Success : NCI Thesaurus history edits retrieved");
                        } else {
                                System.out
                                .println("FAILURE : NCI Thesaurus history edits not found");
                        }
                        //C41330 and C71027 were merged, with C71027 being retired and C41330 remaining active
                        //getAncestors checks to see if any concepts have contributed to C41330.
                        ncel = hs.getAncestors(Constructors.createConceptReference("C41330", null));
                        NCIChangeEvent[] nce = ncel.getEntry();
                        //getDescendants checks to see if this concepts has contributed to any other concept.
                        //The getDescendants method is useful if the concept you have is retired and you want
to
                        //see what other concept should be used.
                        ncel = hs.getDescendants(Constructors.createConceptReference("C71027", null));
                        nce = ncel.getEntry();
                        //Test Metathesaurus History
                        hs = lbSvc.getHistoryService(NCIm);
                        srl = hs.getBaselines(null, null);
                        if (srl.getSystemReleaseCount() > 10) {
                                System.out
                                .println("Success : Metathesaurus history baselines retrieved");
                        } else {
                                System.out
                                .println("FAILURE : MetaThesaurus history baselines not found");
                        }
                        ncel = hs.getEditActionList(
                                        Constructors.createConceptReference("C0359583", null),
                                        null, null);
                        if (ncel.getEntryCount() > 0) {
                                System.out
                                .println("Success : Metathesaurus history edits retrieved");
                        } else {
                                System.out
                                .println("FAILURE : Metathesaurus history edits not found");
                        }
                } catch (Exception e) {
                        e.printStackTrace();
                }
        }
```

Asserted Value Set Search

```
                        //Instantiate the search extension
                 LexBIGService lbsvc = LexBIGServiceImpl.defaultInstance();
                 SourceAssertedValueSetSearchExtensionImpl assertedVSsvc = null;
                       try {
                assertedVSsvc = (SourceAssertedValueSetSearchExtensionImpl) lbsvc
                   .getGenericExtension("AssertedValueSetSearchExtension");
                    } catch (LBException e) {
                e.printStackTrace();
                    }
                        //Search for results based on an exact match of the unique identifier of the value set
member
                ResolvedConceptReferencesIterator itr = assertedVSsvc.
                search("C48323", null, null, MatchAlgorithm.CODE_EXACT,
                false, false);
                        while(itr.hasNext()){
                                ResolvedConceptReference ref = itr.next();
                                System.out.println("description: " + ref.getEntityDescription().getContent()):
                        }
                        //Search for results based on an exact match of the preferred text designation of the
value set member
                ResolvedConceptReferencesIterator itr = assertedVSsvc.
                search("Black", null, null,
                MatchAlgorithm.PRESENTATION_EXACT, false, false);
                        while(itr.hasNext()){
                                ResolvedConceptReference ref = itr.next();
                                System.out.println("description: " + ref.getEntityDescription().getContent()):
                        }

            //Search for results based on an containing match of the preferred text designation of the value
set member
            ResolvedConceptReferencesIterator itr = assertedVSsvc.
                search("Blinding", null, null,
                MatchAlgorithm.PRESENTATION_CONTAINS, false, false);
            while(itr.hasNext()){
                ResolvedConceptReference ref = itr.next();
                System.out.println("description: " + ref.getEntityDescription().getContent()):
            }

                        //Search for results based on an lucene query parser match of the preferred text
designation of the value set member
            ResolvedConceptReferencesIterator itr = assertedVSsvc.
                search("BlindingWhite", null, null,
                MatchAlgorithm.LUCENE, false, false);
            while(itr.hasNext()){
                ResolvedConceptReference ref = itr.next();
                System.out.println("description: " + ref.getEntityDescription().getContent()):
            }

                    //Search for results based on an exact match of some property of the value set member
                        //Also, restrict the search to a given value set designation
                        CodingSchemeReference csRef = new CodingSchemeReference();
                csRef.setCodingScheme("http://evs.nci.nih.gov/valueset/FDA/C54453");
                csRef.setVersionOrTag(Constructors.createCodingSchemeVersionOrTagFromVersion("0.1.5.1"));
                ResolvedConceptReferencesIterator itr = assertedVSsvc.search("Black", csRef,  MatchAlgorithm.
PROPERTY_EXACT);
                        while(itr.hasNext()){
                                ResolvedConceptReference ref = itr.next();
                                System.out.println("description: " + ref.getEntityDescription().getContent()):
                        }
```

Resolved Value Set Services

```
//Instantiate the resolved value set service with default parameters allowing the NCIt self asserting value
sets to be accessed
LexEVSResolvedValueSetService service = new LexEVSResolvedValueSetServiceImpl(new AssertedValueSetParameters.
Builder().build());

//Get a list of all resolved value sets in a coding scheme representation with entities included
List<CodingScheme> list = service.listAllResolvedValueSets();

//Get a list of all resolved value sets in a coding scheme representation -- no entity resolution, should be
faster
List<CodingScheme> schemes = service.getMinimalResolvedValueSetSchemes();

//Get a resolved value set in coding scheme form based on the uniquely identifying URI
URI uri = new URI("http://evs.nci.nih.gov/valueset/TEST/C48323");
CodingScheme ref = service.getResolvedValueSetForValueSetURI(uri);
//Just get the entities of that value set
ResolvedConceptReferenceList refs = service.getValueSetEntitiesForURI(uri.toString());
//Get an iterator instead of a list
ResolvedConceptReferencesIterator refs = service.getValueSetIteratorForURI(uri.toString());

//Get a resolved value set in coding scheme form based on a partly populated concept reference
ConceptReference ref = new ConceptReference();
ref.setCode("C123434");
ref.setCodeNamespace("ncit");
ref.setCodingSchemeName("NCI Thesaurus");
List<CodingScheme> schemes = service.getResolvedValueSetsForConceptReference(ref);

//Search value sets for text match
//This will match both traditional resolved value sets and asserted value sets
List<AbsoluteCodingSchemeVersionReference> refs =
                service.getResolvedValueSetsforTextSearch("blood",
                        MatchAlgorithm.LUCENE);
```

Code Files

BuildTreeForCode

Attempts to provide a tree, based on a focus code, that includes the following information:

- All paths from the hierarchy root to one or more focus codes.
- Immediate children of every node in path to root
- Indicator to show whether any unexpanded node can be further expanded

This example accepts two parameters... The first parameter is required, and must contain at least one code in a comma-delimited list. A tree is produced for each code. Time to produce the tree for each code is printed in milliseconds. In order to factor out costs of startup and shutdown, resolving multiple codes may offer a better overall estimate performance.

The second parameter is optional, and can indicate a hierarchy ID to navigate when resolving child nodes. If not provided, "is a" is assumed.

CodingSchemeSelectionMenu

Displays a list of available coding schemes.

FindCodesForDescription

Example showing how to find codes matching descriptive text. The program accepts up to two parameters...

The first param (required) indicates the text used to search matching descriptions. Matches are determined through a customized match algorithm, which uses a simple heuristic to try and rank returned values by relevance.

The second param (optional) indicates the type of entity to search. Possible values include the LexGrid built-in types "concept" and "instance". Additional supported types can be defined uniquely to a coding scheme. If provided, this should be a comma-delimited list of types. If not provided, all entity types are searched.

Example: FindCodesForDescription "blood" Example: FindCodesForDescription "breast cancer" "concept"

FindDescriptionForCode

Example showing how to find the entity description assigned to a specific code. The program accepts one parameter, the entity code.

FindPropsAndAssocForCode

Example showing how to find concept properties and associations based on a code.

FindRelatedCodes

Example showing how to find all concepts codes related to another code with distance 1.

FindRelatedCodesWithPropertyLinks

Example showing how to find all concepts codes related to another code with distance 1, plus the Property Link relations.

FindRelatedNodesForTermAndAssoc

Example showing how to find all endpoints of a named association for which the given term matches as source or target.

Note: the match algorithm applied to the term is the standard lucene query syntax.

FindUMLSContextsForCUI

Example showing any source-asserted hierarchies (based on import of MRHIER HCD) for a CUI. The program takes a single argument (the UMLS CUI), prompts for the code system to query in the LexGrid repository, and displays the available hierarchical relationships.

ListHierarchy

Example showing how to determine and display an unsorted list of root and subsumed nodes, up to a specified depth, for hierarchical relationships.

This program accepts two parameters:

The first parameter indicates the depth to display for the hierarchy. If 1, nodes immediately subsumed by the root are displayed. If 2, grandchildren are displayed, etc. If absent or < 0, a default depth of 3 is assumed.

The second parameter optionally indicates a specific hierarchy to navigate. If provided, this must match a registered identifier in the coding scheme supported hierarchy metadata. If left unspecified, all hierarchical associations are navigated. If an incorrect value is specified, a list of supported values will be output for future reference.

BACKGROUND: From a database perspective, LexBIG stores relationships internally in a forward direction, source to target. Due to differences in source formats, however, a wide variety of associations may be used ('PAR', 'CHD', 'isa', 'hasSubtype', etc). In addition, the direction of navigation may vary ('isa' expands in a reverse direction whereas 'hasSubtype' expands in a forward direction.

The intent of the getHierarchy* methods on the LexBIGServiceConvenienceMethods interface is to simplify the process of hierarchy discovery and navigation. These methods significantly reduce the need to understand conventions for root nodes, associations, and direction of navigation for a specific source format.

ListHierarchyByCode

Example showing how to determine and display the hierarchical relationships for a specific code, ancestors or descendants, within a fixed distance.

This program accepts two parameters, indicating the code and distance. The first parameter is the code (required). The second parameter is the distance (optional). If 1, immediate children are displayed. If 2, grandchildren are displayed, etc. If absent or < 0, all downstream branches are displayed.

BACKGROUND: From a database perspective, LexBIG stores relationships internally in a forward direction, source to target. Due to differences in source formats, however, a wide variety of associations may be used ('PAR', 'CHD', 'isa', 'hasSubtype', etc). In addition, the direction of navigation may vary ('isa' expands in a reverse direction whereas 'hasSubtype' expands in a forward direction.

The intent of the getHierarchy* methods on the LexBIGServiceConvenienceMethods interface is to simplify the process of hierarchy discovery and navigation. These methods significantly reduce the need to understand conventions for root nodes, associations, and direction of navigation for a specific source format.

ListHierarchyMetaBySource

Example showing how to determine and display an unsorted list of root and subsumed nodes, up to a specified depth, for hierarchical relationships. It is written specifically to handle display of relationships for a designated source within the NCI Metathesaurus.

This program accepts two parameters. The first indicates the depth to display hierarchical relations. If 0, only the root nodes are displayed. If 1, nodes immediately subsumed by the root are also displayed, etc. If < 0, a default depth of 0 is assumed.

The second parameter must provide the source abbreviation (SAB) of the Metathesaurus source to be evaluated (e.g. ICD9CM, MDR, SNOMEDCT).

ListHierarchyPathToRoot

Example showing how to determine and display paths from a given concept back to defined root nodes through any hierarchies registered for the coding scheme.

This program accepts one parameter (required), indicating the code to evaluate.

BACKGROUND: From a database perspective, LexBIG stores relationships internally in a forward direction, source to target. Due to differences in source formats, however, a wide variety of associations may be used ('PAR', 'CHD', 'isa', 'hasSubtype', etc). In addition, the direction of navigation may vary ('isa' expands in a reverse direction whereas 'hasSubtype' expands in a forward direction.

The intent of the getHierarchy* methods on the LexBIGServiceConvenienceMethods interface is to simplify the process of hierarchy discovery and navigation. These methods significantly reduce the need to understand conventions for root nodes, associations, and direction of navigation for a specific source format.

MetaDataSearch

Example how to query stored metadata for a code system. For the example, use the LoadSampleMetaDataData.bat to load the required code system and metadata.

MetaMatch

Example attempting to approximate some characteristics of the Metaphrase search algorithm. However, full Metaphrase compatibility is not anticipated.

ProfileScheme

Requires loading valid scheme (must have root node named @ pointing to top nodes) Profiles a coding scheme based on unique URN, version, relation and scheme name.

Note: If the URN and version values are unspecified, a list of available coding schemes will be presented for user selection.

ScoredIterator

Used to wrap scored results for consumption as a standard ResolvedConceptReferenceIterator.

ScoredTerm

Used to manage and sort search results based on a scoring algorithm.

ScoreTerm

Example showing a simple scoring algorithm that evaluates a provided term against available terms in a code system. A cutoff percentage can optionally be provided.

SoundsLike

Example showing how to list concepts with presentation text that 'sounds like' a specified value.

Util

Utility functions to support the examples.