ListHierarchyMetaBySource Java

Java Code

```
/*
 * Copyright: (c) 2004-2009 Mayo Foundation for Medical Education and
 * Research (MFMER). All rights reserved. MAYO, MAYO CLINIC, and the
 * triple-shield Mayo logo are trademarks and service marks of MFMER.
 *
 * Except as contained in the copyright notice above, or as used to identify
 * MFMER as the author of this software, the trade names, trademarks, service
 * marks, or product names of the copyright holder shall not be used in
 * advertising, promotion or otherwise in connection with this software without
 * prior written authorization of the copyright holder.
 *
 * Licensed under the Eclipse Public License, Version 1.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *                 http://www.eclipse.org/legal/epl-v10.html
 *
 */
package org.LexGrid.LexBIG.example;

import java.util.Iterator;

import org.LexGrid.LexBIG.DataModel.Collections.AssociatedConceptList;
import org.LexGrid.LexBIG.DataModel.Collections.AssociationList;
import org.LexGrid.LexBIG.DataModel.Collections.ResolvedConceptReferenceList;
import org.LexGrid.LexBIG.DataModel.Core.AssociatedConcept;
import org.LexGrid.LexBIG.DataModel.Core.Association;
import org.LexGrid.LexBIG.DataModel.Core.CodingSchemeSummary;
import org.LexGrid.LexBIG.DataModel.Core.CodingSchemeVersionOrTag;
import org.LexGrid.LexBIG.DataModel.Core.ResolvedConceptReference;
import org.LexGrid.LexBIG.Exceptions.LBException;
import org.LexGrid.LexBIG.Extensions.Generic.LexBIGServiceConvenienceMethods;
import org.LexGrid.LexBIG.Impl.LexBIGServiceImpl;
import org.LexGrid.LexBIG.LexBIGService.CodedNodeSet;
import org.LexGrid.LexBIG.LexBIGService.LexBIGService;
import org.LexGrid.LexBIG.LexBIGService.CodedNodeSet.PropertyType;
import org.LexGrid.LexBIG.Utility.ConvenienceMethods;
import org.LexGrid.commonTypes.EntityDescription;

/**
 * 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).
 */
public class ListHierarchyMetaBySource {
    final static int DEFAULT_DEPTH = 0;

    public ListHierarchyMetaBySource() {
        super();
    }

    /**
     * Entry point for processing.
     *
     * @param args
```

```
     */
    public static void main(String[] args) {
        if (args.length < 2) {
            System.out.println("Example: ListHierarchyMetaBySource 0 \"MDR\"");
            return;
        }

        try {
            int maxDepth = Math.max(Integer.parseInt(args[0]), 0);
            String sab = args[1];
            new ListHierarchyMetaBySource().run(maxDepth, sab);
        } catch (NumberFormatException nfe) {
            System.out.println("Parameter 1 must indicate a maximum depth of the hierarchy to display.\n"
                    + "Example: ListHierarchyMetaBySource 0 \"MDR\"");
            return;
        } catch (Exception e) {
            Util.displayAndLogError("REQUEST FAILED !!!", e);
        }
    }

    public void run(int maxDepth, String sab) throws LBException {
        CodingSchemeSummary css = Util.promptForCodeSystem();
        long ms = System.currentTimeMillis();
        try {
            if (css != null) {
                LexBIGService lbSvc = LexBIGServiceImpl.defaultInstance();
                String scheme = css.getCodingSchemeURI();
                CodingSchemeVersionOrTag csvt = new CodingSchemeVersionOrTag();
                csvt.setVersion(css.getRepresentsVersion());
                print(maxDepth, sab, lbSvc, scheme, csvt);
            }
        } finally {
            System.out.println("Run time (ms): " + (System.currentTimeMillis() - ms));
        }
    }

    /**
     * Handle one level of the tree, and recurse up to the indicated depth.
     *
     * @param depth
     * @param sab
     * @param lbSvc
     * @param csvt
     * @param scheme
     * @param tagOrVersion
     * @throws LBException
     */
    protected void print(int depth, String sab, LexBIGService lbSvc, String scheme, CodingSchemeVersionOrTag
csvt)
            throws LBException {
        Util.displayMessage("================================================");
        Util.displayMessage("PROCESSING SAB: " + sab);
        Util.displayMessage("================================================");

        LexBIGServiceConvenienceMethods lbscm = (LexBIGServiceConvenienceMethods) lbSvc
                .getGenericExtension("LexBIGServiceConvenienceMethods");

        PropertyType[] presentationProps = new PropertyType[] { PropertyType.PRESENTATION };
        // NOTE: Processing relations for a specific hierarchy (by ID) in Meta
        // can be slow, since it relies on transitivity checks to determine
        // whether it is possible to navigate from concepts back to a specific
        // root nodes registered against the hierarchy ID.
        // However, for Meta we really don't need to worry about a specific
        // hierarchy ID. We can assume navigation of all hierarchies (boiled
        // down to PAR/CHD) and restrict nodes and associations by source to
        // get what we need.
        CodedNodeSet rootNodeSet = lbscm.getHierarchyRootSet(scheme, csvt, null);
        rootNodeSet.restrictToProperties(null, presentationProps, ConvenienceMethods.createLocalNameList(sab),
null,
                null);
        ResolvedConceptReferenceList rootNodes = rootNodeSet.resolveToList(null, null, presentationProps, -1);
```

```
        printChain(lbscm, scheme, sab, csvt, rootNodes, depth);
    }

    /**
     * Handles recursive display of hierarchical relations for the given set of
     * nodes, up to the maximum specified depth.
     *
     * @param lbscm
     * @param scheme
     * @param sab
     * @param csvt
     *            ,
     * @param rootNodes
     * @param maxDepth
     * @throws LBException
     */
    protected void printChain(LexBIGServiceConvenienceMethods lbscm, String scheme, String sab,
            CodingSchemeVersionOrTag csvt, ResolvedConceptReferenceList rootNodes, int maxDepth) throws
LBException {
        for (Iterator<ResolvedConceptReference> nodes = rootNodes.iterateResolvedConceptReference(); nodes.
hasNext();)
            printChainForNode(lbscm, scheme, sab, csvt, nodes.next(), 0, maxDepth, null);
    }

    /**
     * Handles recursive display of hierarchy for an individual node, up to the
     * maximum specified depth.
     *
     * @param lbscm
     * @param scheme
     * @param sab
     * @param csvt
     * @param ref
     * @param currentDepth
     * @param maxDepth
     * @param assocName
     * @throws LBException
     */
    protected void printChainForNode(LexBIGServiceConvenienceMethods lbscm, String scheme, String sab,
            CodingSchemeVersionOrTag csvt, ResolvedConceptReference ref, int currentDepth, int maxDepth,
            String assocName) throws LBException {
        // Print the referenced node; indent based on current depth ...
        StringBuffer indent = new StringBuffer();
        for (int i = 0; i <= currentDepth; i++)
            indent.append("    ");

        String code = ref.getConceptCode();
        EntityDescription desc = ref.getEntityDescription();
        Util.displayMessage(new StringBuffer().append(indent).append(assocName != null ? (assocName + "->") :
"")
                .append(code).append(':').append(desc != null ? desc.getContent() : "").toString());

        if (currentDepth < maxDepth) {
            AssociationList aList = lbscm.getHierarchyLevelNext(scheme, csvt, null, code, false,
ConvenienceMethods
                    .createNameAndValueList(sab, "Source"));
            for (int i = 0; i < aList.getAssociationCount(); i++) {
                Association assoc = aList.getAssociation(i);
                AssociatedConceptList acList = assoc.getAssociatedConcepts();
                for (Iterator<AssociatedConcept> nodes = acList.iterateAssociatedConcept(); nodes.hasNext();) {
                    printChainForNode(lbscm, scheme, sab, csvt, nodes.next(), currentDepth + 1, maxDepth, assoc
                            .getDirectionalName());
                }
            }
        }
    }
}
```