```
 1  /*
 2   * File:  CreateDOMDocument_JMH_v01.cpp
 3   * Jesse M. Heines, UMass Lowell Computer Science, heines@cs.uml.edu
 4   * Adapted from Xerces C++ API for XML examples CreateDOMDocument and DOMPrint.
 5   * Modifications copyright (c) 2015 by Jesse M. Heines.  All rights reserved.
 6   *   May be freely copied or excerpted for educational purposes with credit
 7   *   to the author.
 8   * updated by JMH on November 8, 2015 at 10:16 PM
 9   */
10
11
12  /*
13   * Licensed to the Apache Software Foundation (ASF) under one or more
14   * contributor license agreements.  See the NOTICE file distributed with
15   * this work for additional information regarding copyright ownership.
16   * The ASF licenses this file to You under the Apache License, Version 2.0
17   * (the "License"); you may not use this file except in compliance with
18   * the License.  You may obtain a copy of the License at
19   *
20   *      http://www.apache.org/licenses/LICENSE-2.0
21   *
22   * Unless required by applicable law or agreed to in writing, software
23   * distributed under the License is distributed on an "AS IS" BASIS,
24   * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
25   * See the License for the specific language governing permissions and
26   * limitations under the License.
27   */
28
29  /*
30   * $Id: CreateDOMDocument.cpp 676796 2008-07-15 05:04:13Z dbertoni $
31   */
32
33  /*
34   * This sample illustrates how you can create a DOM tree in memory.
35   * It then prints the count of elements in the tree.
36   */
37
38
39  // ---------------------------------------------------------------------------
40  //  Includes
41  // ---------------------------------------------------------------------------
42  #include <xercesc/util/PlatformUtils.hpp>
43  #include <xercesc/util/XMLString.hpp>
44  #include <xercesc/dom/DOM.hpp>
45  #include <xercesc/util/OutOfMemoryException.hpp>
46
47  #include <xercesc/util/PlatformUtils.hpp>
48
49  #include <xercesc/dom/DOM.hpp>
50
51  #include <xercesc/framework/StdOutFormatTarget.hpp>
52  #include <xercesc/framework/LocalFileFormatTarget.hpp>
53  #include <xercesc/parsers/XercesDOMParser.hpp>
54  #include <xercesc/util/XMLUni.hpp>
55
56  #include "DOMTreeErrorReporter.hpp"
57  #include "DOMPrintFilter.hpp"
58  #include "DOMPrintErrorHandler.hpp"
59
60  #include <xercesc/dom/DOMTreeWalker.hpp>
61
62  #include <string.h>
63  #include <string>
64  #include <stdlib.h>
65  #include <time.h>
66
```

```
 67  #if defined(XERCES_NEW_IOSTREAMS)
 68  #include <iostream>
 69  #else
 70  #include <iostream.h>
 71  #endif
 72
 73
 74  XERCES_CPP_NAMESPACE_USE
 75
 76  // ---------------------------------------------------------------------------
 77  //  This is a simple class that lets us do easy (though not terribly efficient)
 78  //  transcoding of char* data to XMLCh data.
 79  // ---------------------------------------------------------------------------
 80  class XStr
 81  {
 82  public :
 83      // -----------------------------------------------------------------------
 84      //  Constructors and Destructor
 85      // -----------------------------------------------------------------------
 86      XStr(const char* const toTranscode)
 87      {
 88          // Call the private transcoding method
 89          fUnicodeForm = XMLString::transcode(toTranscode);
 90      }
 91
 92      ~XStr()
 93      {
 94          XMLString::release(&fUnicodeForm);
 95      }
 96
 97
 98      // -----------------------------------------------------------------------
 99      //  Getter methods
100      // -----------------------------------------------------------------------
101      const XMLCh* unicodeForm() const
102      {
103          return fUnicodeForm;
104      }
105
106  private :
107      // -----------------------------------------------------------------------
108      //  Private data members
109      //
110      //  fUnicodeForm
111      //      This is the Unicode XMLCh format of the string.
112      // -----------------------------------------------------------------------
113      XMLCh*   fUnicodeForm;
114  };
115
116  #define X(str) XStr(str).unicodeForm()
117
118
119  // ---------------------------------------------------------------------------
120  //  Local data
121  //
122  //  gXmlFile
123  //      The path to the file to parser. Set via command line.
124  //
125  //  gDoNamespaces
126  //      Indicates whether namespace processing should be done.
127  //
128  //  gDoSchema
129  //      Indicates whether schema processing should be done.
130  //
131  //  gSchemaFullChecking
132  //      Indicates whether full schema constraint checking should be done.
133  //
134  //  gDoCreate
135  //      Indicates whether entity reference nodes needs to be created or not
136  //      Defaults to false
137  //
```

```
138 //  gOutputEncoding
139 //      The encoding we are to output in. If not set on the command line,
140 //      then it is defaults to the encoding of the input XML file.
141 //
142 //  gSplitCdataSections
143 //      Indicates whether split-cdata-sections is to be enabled or not.
144 //
145 //  gDiscardDefaultContent
146 //      Indicates whether default content is discarded or not.
147 //
148 //  gUseFilter
149 //      Indicates if user wants to plug in the DOMPrintFilter.
150 //
151 //  gValScheme
152 //      Indicates what validation scheme to use. It defaults to 'auto', but
153 //      can be set via the -v= command.
154 //
155 // ---------------------------------------------------------------------------
156 // not used, as reported by -Wall   static char* gXmlFile            = 0;
157 // not used, as reported by -Wall   static bool  gDoNamespaces       = false;
158 // not used, as reported by -Wall   static bool  gDoSchema           = false;
159 // not used, as reported by -Wall   static bool  gSchemaFullChecking = false;
160 // not used, as reported by -Wall   static bool  gDoCreate           = false;
161
162 static char* goutputfile            = 0;
163 static char* gXPathExpression       = 0;
164
165 // options for DOMLSSerializer's features
166 static XMLCh* gOutputEncoding        = 0;
167
168 static bool gSplitCdataSections    = true;
169 static bool gDiscardDefaultContent = true;
170 static bool gUseFilter             = false;
171 static bool gFormatPrettyPrint     = true;
172 static bool gWriteBOM              = false;
173
174
175
176 void PrintDOM( DOMDocument* doc ) {
177
178     // retval was used in a previous version of this function that returned an int
179     // it is no longer used, but has been kept for future use
180     // in this case, however, the second statement below is required to avoid a compiler warning
181     int retval = 0 ;  // function return value
182     retval = retval ; // to avoid warning that retval is not used
183
184     // If the parse was successful, output the document data from the DOM tree
185     // if (!errorsOccured && !errReporter->getSawErrors())
186     // {
187         DOMPrintFilter   *myFilter = 0;
188
189         try
190         {
191             // get a serializer, an instance of DOMLSSerializer
192             XMLCh tempStr[3] = {chLatin_L, chLatin_S, chNull};
193             DOMImplementation *impl          = DOMImplementationRegistry::getDOMImplementation(tempStr);
194             DOMLSSerializer   *theSerializer = ((DOMImplementationLS*)impl)->createLSSerializer();
195             DOMLSOutput       *theOutputDesc = ((DOMImplementationLS*)impl)->createLSOutput();
196
197             // set user specified output encoding
198             theOutputDesc->setEncoding(gOutputEncoding);
199
200             // plug in user's own filter
201             if (gUseFilter)
202             {
203                 // even we say to show attribute, but the DOMLSSerializer
204                 // will not show attribute nodes to the filter as
205                 // the specs explicitly says that DOMLSSerializer shall
206                 // NOT show attributes to DOMLSSerializerFilter.
207                 //
208                 // so DOMNodeFilter::SHOW_ATTRIBUTE has no effect.
```

```
209                 // same DOMNodeFilter::SHOW_DOCUMENT_TYPE, no effect.
210                 //
211                 myFilter = new DOMPrintFilter(DOMNodeFilter::SHOW_ELEMENT   |
212                                               DOMNodeFilter::SHOW_ATTRIBUTE |
213                                               DOMNodeFilter::SHOW_DOCUMENT_TYPE);
214                 theSerializer->setFilter(myFilter);
215             }
216
217             // plug in user's own error handler
218             DOMErrorHandler *myErrorHandler = new DOMPrintErrorHandler();
219             DOMConfiguration* serializerConfig=theSerializer->getDomConfig();
220             serializerConfig->setParameter(XMLUni::fgDOMErrorHandler, myErrorHandler);
221
222             // set feature if the serializer supports the feature/mode
223             if (serializerConfig->canSetParameter(XMLUni::fgDOMWRTSplitCdataSections, gSplitCdataSections))
224                 serializerConfig->setParameter(XMLUni::fgDOMWRTSplitCdataSections, gSplitCdataSections);
225
226             if (serializerConfig->canSetParameter(XMLUni::fgDOMWRTDiscardDefaultContent,
gDiscardDefaultContent))
227                 serializerConfig->setParameter(XMLUni::fgDOMWRTDiscardDefaultContent,
gDiscardDefaultContent);
228
229             if (serializerConfig->canSetParameter(XMLUni::fgDOMWRTFormatPrettyPrint, gFormatPrettyPrint))
230                 serializerConfig->setParameter(XMLUni::fgDOMWRTFormatPrettyPrint, gFormatPrettyPrint);
231
232             if (serializerConfig->canSetParameter(XMLUni::fgDOMWRTBOM, gWriteBOM))
233                 serializerConfig->setParameter(XMLUni::fgDOMWRTBOM, gWriteBOM);
234
235             //
236             // Plug in a format target to receive the resultant
237             // XML stream from the serializer.
238             //
239             // StdOutFormatTarget prints the resultant XML stream
240             // to stdout once it receives any thing from the serializer.
241             //
242             XMLFormatTarget *myFormTarget;
243             if (goutputfile) {
244                 myFormTarget=new LocalFileFormatTarget(goutputfile);
245             } else {
246                 myFormTarget=new StdOutFormatTarget();
247             }
248             theOutputDesc->setByteStream(myFormTarget);
249
250             // get the DOM representation
251             // DOMDocument *doc = parser->getDocument();
252
253             //
254             // do the serialization through DOMLSSerializer::write();
255             //
256             if(gXPathExpression!=NULL)
257             {
258                 XMLCh* xpathStr=XMLString::transcode(gXPathExpression);
259                 DOMElement* root = doc->getDocumentElement();
260                 try
261                 {
262                     DOMXPathNSResolver* resolver=doc->createNSResolver(root);
263                     DOMXPathResult* result=doc->evaluate(
264                       xpathStr,
265                       root,
266                       resolver,
267                       DOMXPathResult::ORDERED_NODE_SNAPSHOT_TYPE,
268                       NULL);
269
```

```
270                     XMLSize_t nLength = result->getSnapshotLength();
271                     for(XMLSize_t i = 0; i < nLength; i++)
272                     {
273                       result->snapshotItem(i);
274                       theSerializer->write(result->getNodeValue(), theOutputDesc);
275                     }
276
277                     result->release();
278                     resolver->release ();
279                 }
280                 catch(const DOMXPathException& e)
281                 {
282                     XERCES_STD_QUALIFIER cerr << "An error occurred during processing of the XPath
expression. Msg is:"
283                         << XERCES_STD_QUALIFIER endl
284                         << StrX(e.getMessage()) << XERCES_STD_QUALIFIER endl;
285                     retval = 4;
286                 }
287                 catch(const DOMException& e)
288                 {
289                     XERCES_STD_QUALIFIER cerr << "An error occurred during processing of the XPath
expression. Msg is:"
290                         << XERCES_STD_QUALIFIER endl
291                         << StrX(e.getMessage()) << XERCES_STD_QUALIFIER endl;
292                     retval = 4;
293                 }
294                 XMLString::release(&xpathStr);
295             }
296             else {
297                 theSerializer->write(doc, theOutputDesc);
298             }
299
300             theOutputDesc->release();
301             theSerializer->release();
302
303             //
304             // Filter, formatTarget and error handler
305             // are NOT owned by the serializer.
306             //
307             delete myFormTarget;
308             delete myErrorHandler;
309
310             if (gUseFilter)
311                 delete myFilter;
312
313         }
314         catch (const OutOfMemoryException&)
315         {
316             XERCES_STD_QUALIFIER cerr << "OutOfMemoryException" << XERCES_STD_QUALIFIER endl;
317             retval = 5;
318         }
319         catch (XMLException& e)
320         {
321             XERCES_STD_QUALIFIER cerr << "An error occurred during creation of output transcoder. Msg is:"
322                 << XERCES_STD_QUALIFIER endl
323                 << StrX(e.getMessage()) << XERCES_STD_QUALIFIER endl;
324             retval = 4;
325         }
326     // }
327 }
328
329
```

```
330 // ---------------------------------------------------------------------------
331 //  main
332 // ---------------------------------------------------------------------------
333
334 int main(int argC, char*[])
335 {
336     // Initialize the XML4C2 system.
337     try
338     {
339         XMLPlatformUtils::Initialize();
340     }
341
342     catch(const XMLException& toCatch)
343     {
344         char *pMsg = XMLString::transcode(toCatch.getMessage());
345         XERCES_STD_QUALIFIER cerr << "Error during Xerces-c Initialization.\n"
346              << "  Exception message:"
347              << pMsg;
348         XMLString::release(&pMsg);
349         return 1;
350     }
351
352     // Watch for special case help request
353     int errorCode = 0;
354     if (argC > 1)
355     {
356         XERCES_STD_QUALIFIER cout << "\nUsage:\n"
357                 "    CreateDOMDocument\n\n"
358                 "This program creates a new DOM document from scratch in memory.\n"
359                 "It then prints the count of elements in the tree.\n"
360              << XERCES_STD_QUALIFIER endl;
361         errorCode = 1;
362     }
363     if(errorCode) {
364         XMLPlatformUtils::Terminate();
365         return errorCode;
366     }
367
368    {
369        //  Nest entire test in an inner block.
370        //  The tree we create below is the same that the XercesDOMParser would
371        //  have created, except that no whitespace text nodes would be created.
372
373        // <company>
374        //     <product>Xerces-C</product>
375        //     <category idea='great'>XML Parsing Tools</category>
376        //     <developedBy>Apache Software Foundation</developedBy>
377        // </company>
378
379        DOMImplementation* impl =  DOMImplementationRegistry::getDOMImplementation(X("Core"));
380
381        if (impl != NULL)
382        {
383            try
384            {
385 /*
386  *             DOMDocument* doc = impl->createDocument(
387  *                         0,                    // root element namespace URI.
388  *                         X("company"),         // root element name
389  *                         0);                   // document type object (DTD).
390  *
391  *             DOMElement* elemRoot = doc->getDocumentElement();
392  *
393  *             DOMElement*  prodElem = doc->createElement(X("product"));
394  *             elemRoot->appendChild(prodElem);
395  *
396  *             DOMText*    prodDataVal = doc->createTextNode(X("Xerces-C"));
397  *             prodElem->appendChild(prodDataVal);
398  *
```

```
399   *             DOMElement*  catElem = doc->createElement(X("category"));
400   *             elemRoot->appendChild(catElem);
401   *
402   *             catElem->setAttribute(X("idea"), X("great"));
403   *
404   *             DOMText*    catDataVal = doc->createTextNode(X("XML Parsing Tools"));
405   *             catElem->appendChild(catDataVal);
406   *
407   *             DOMElement*  devByElem = doc->createElement(X("developedBy"));
408   *             elemRoot->appendChild(devByElem);
409   *
410   *             DOMText*    devByDataVal = doc->createTextNode(X("Apache Software Foundation"));
411   *             devByElem->appendChild(devByDataVal);
412   *
413   *             DOMElement* elemJMH1 = doc->createElement(X("modified-by"));
414   *             elemRoot->appendChild(elemJMH1);
415   *             elemJMH1->setAttribute(X("full-name"), X("Jesse M. Heines"));
416   *             elemJMH1->setAttribute(X("title"), X("Professor"));
417   *             elemJMH1->setAttribute(X("department"), X("Computer Science"));
418   *             // DOMText*     textJMH1 = doc->createTextNode(X("Jesse M. Heines"));
419   *             // elemJMH1->appendChild(textJMH1);
420   *             DOMElement* elemJMH2 = doc->createElement(X("last-name"));
421   *             elemJMH1->appendChild(elemJMH2);
422   *             DOMText* textJMH1 = doc->createTextNode(X("Heines"));
423   *             elemJMH2->appendChild(textJMH1);
424   *
425   *             elemJMH2 = doc->createElement(X("first-name"));
426   *             elemJMH1->appendChild(elemJMH2);
427   *             textJMH1 = doc->createTextNode(X("Jesse"));
428   *             elemJMH2->appendChild(textJMH1);
429   *
430   *             DOMElement* elemJMH3 = doc->createElement(X("original-name"));
431   *             elemJMH2->appendChild(elemJMH3);
432   *             textJMH1 = doc->createTextNode(X("Shanus"));
433   *             elemJMH3->appendChild(textJMH1);
434   */
435                 // current date and time
436                 // http://stackoverflow.com/questions/997946/c-get-current-time-and-date
437                 time_t     now = time(0);
438                 struct tm  tstruct;
439                 char       buf[80];
440                 tstruct = *localtime(&now);
441                 // http://www.cplusplus.com/reference/clibrary/ctime/strftime/
442                 strftime(buf, sizeof(buf), "updated by JMH at %B %d, %Y at %I:%M %p", &tstruct);
443
444                 DOMDocument* doc = impl->createDocument(
445                             0,                    // root element namespace URI.
446                             X("course"),          // root element name
447                             0);                   // document type object (DTD).
448
449                 DOMElement* elemRoot= doc->getDocumentElement();
450
451                 DOMComment* comm = doc->createComment(X("\n  File:
/~heines/91.204/91.204-2012-13s/course.xml\
452   \n  Jesse M. Heines, UMass Lowell Computer Science, heines@cs.uml.edu\
453   \n  Copyright (c) 2012 by Jesse M. Heines.  All rights reserved.  May be freely\
454   \n    copied or excerpted for educational purposes with credit to the author.\
455   \n  updated by JMH on September 21, 2012 at 12:35 PM\
456   \n\
457   \n  N.B. Use &#146; rather than ' in names such as O'Connell to avoid XSL errors.\n"));
458                 elemRoot->appendChild(comm);
459
460                 DOMElement* elem = doc->createElement(X("timestamp"));
461                 elemRoot->appendChild(elem);
462                 DOMText* text = doc->createTextNode(X(buf));
463                 elem->appendChild(text);
464
465                 elem = doc->createElement(X("title"));
466                 elemRoot->appendChild(elem);
467                 text = doc->createTextNode(X("Computing IV"));
468                 elem->appendChild(text);
```

```
469
470                 elem = doc->createElement(X("section"));
471                 elemRoot->appendChild(elem);
472                 text = doc->createTextNode(X("201"));
473                 elem->appendChild(text);
474
475                 elem = doc->createElement(X("semester"));
476                 elemRoot->appendChild(elem);
477                 text = doc->createTextNode(X("2015-16f"));
478                 elem->appendChild(text);
479
480                 elem = doc->createElement(X("academic-calendar-link-pdf"));
481                 elemRoot->appendChild(elem);
482                 text = doc->createTextNode(X("http://www.uml.edu/docs/Acal_2016F_tcm18-142546.pdf"));
483                 elem->appendChild(text);
484
485                 elem = doc->createElement(X("discussion-board"));
486                 elemRoot->appendChild(elem);
487                 text = doc->createTextNode(X("https://piazza.com/uml/fall2015/91204/home"));
488                 elem->appendChild(text);
489
490                 elem = doc->createElement(X("class-videos-url"));
491                 elemRoot->appendChild(elem);
492                 text = doc->createTextNode(X("http://echo360.uml.edu/heines201516/computingIV.html"));
493                 elem->appendChild(text);
494
495                 // start professor structure
496                 elem = doc->createElement(X("professor"));
497
498                 DOMElement* elem2 = doc->createElement(X("name"));
499                 elem->appendChild(elem2);
500                 text = doc->createTextNode(X("Prof. Jesse M. Heines"));
501                 elem2->appendChild(text);
502
503                 elem2 = doc->createElement(X("lastnamefirst"));
504                 elem->appendChild(elem2);
505                 text = doc->createTextNode(X("Heines, Jesse (Computer Science)"));
506                 elem2->appendChild(text);
507
508                 elem2 = doc->createElement(X("email"));
509                 elem->appendChild(elem2);
510                 text = doc->createTextNode(X("heines@cs.uml.edu"));
511                 elem2->appendChild(text);
512
513                 elemRoot->appendChild(elem);
514                 // end professor structure
515
516                 // start roster
517                 DOMElement* elemRoster = doc->createElement(X("roster"));
518                 elemRoot->appendChild(elemRoster);
519
520                 elem2 = doc->createElement(X("student"));
521                 elemRoster->appendChild(elem2);
522                 elem2->setAttribute(X("name"),X("Antrobus, Michael Andrew")) ;
523                 elem2->setAttribute(X("major"),X("Computer Science")) ;
524                 elem2->setAttribute(X("wantstobecalled"),X("Mike")) ;
525                 DOMElement* elemEmail = doc->createElement(X("email"));
526                 elem2->appendChild(elemEmail);
527                 text = doc->createTextNode(X("michael_antrobus@student.uml.edu"));
528                 elemEmail->appendChild(text);
529
530                 elem2 = doc->createElement(X("student"));
531                 elemRoster->appendChild(elem2);
532                 elem2->setAttribute(X("name"),X("Arzuaga, Jeremy Pedro")) ;
533                 elem2->setAttribute(X("major"),X("Computer Science")) ;
534                 elem2->setAttribute(X("wantstobecalled"),X("")) ;
535                 elemEmail = doc->createElement(X("email"));
536                 elem2->appendChild(elemEmail);
537                 text = doc->createTextNode(X("jeremy_arzuaga@student.uml.edu"));
538                 elemEmail->appendChild(text);
539
```

```
540                 elem2 = doc->createElement(X("student"));
541                 elemRoster->appendChild(elem2);
542                 elem2->setAttribute(X("name"),X("Bainbridge, Tyler Jason")) ;
543                 elem2->setAttribute(X("major"),X("Computer Science")) ;
544                 elem2->setAttribute(X("wantstobecalled"),X("")) ;
545                 elemEmail = doc->createElement(X("email"));
546                 elem2->appendChild(elemEmail);
547                 text = doc->createTextNode(X("tyler_bainbridge@student.uml.edu"));
548                 elemEmail->appendChild(text);
549
550                 // end roster
551
522  /*
523   *      <student name="Antrobus, Michael Andrew" major="Computer Science" level="Junior"
wantstobecalled="Mike">
524   *        <email>michael_antrobus@student.uml.edu</email>
525   *        <picturefolder>/Pictures/UMLCS/2015-09-Fall-91.204/images/</picturefolder>
526   *        <picture></picture>
527   *      </student>
528   *      <student name="Arzuaga, Jeremy Pedro" major="Computer Science" level="Senior" wantstobecalled="">
529   *        <email>jeremy_arzuaga@student.uml.edu</email>
530   *        <picturefolder>/Pictures/UMLCS/2015-09-Fall-91.204/images/</picturefolder>
531   *        <picture></picture>
532   *      </student>
533   *      <student name="Bainbridge, Tyler Jason" major="Computer Science" level="Sophomore"
wantstobecalled="">
534   *        <email>tyler_bainbridge@student.uml.edu</email>
535   *        <picturefolder>/Pictures/UMLCS/2015-09-Fall-91.204/images/</picturefolder>
536   *        <picture></picture>
537   *      </student>
538   */
539
540                 //
541                 // Now count the number of elements in the above DOM tree.
542                 //
543
544                 const XMLSize_t elementCount = doc->getElementsByTagName(X("*"))->getLength();
545                 XERCES_STD_QUALIFIER cout << "The tree just created contains: " << elementCount
546                      << " elements.\n" << XERCES_STD_QUALIFIER endl;
547
548                 PrintDOM( doc ) ;  // added by JMH
549                 std::cout << "\n-----------------------------" << std::endl;
550
551                 // JMH: http://www.ibm.com/developerworks/xml/library/x-xercc2/
552                 // JMH: but note that a DOMTreeWalker* is returned, not a DOMTreeWalker
553                 // JMH: this was learned by using NetBeans autocomplete
554
555                 // JMH: Note that SHOW_ATTRIBUTE is meaningful only when creating an DOMNodeIterator or
556                 // DOMTreeWalker with an attribute node as its root; in this case, it means that the
attribute
557                 // node will appear in the first position of the iteration or traversal. Since attributes
are
558                 // never children of other nodes, they do not appear when traversing over the document tree.
559                 //   -- http://xerces.apache.org/xerces-c/apiDocs-3/classDOMNodeFilter.html
560
561                 // JMH: See
http://xerces.apache.org/xerces-c/apiDocs-3/classDOMNode.html#6237ede96be83ff729807688e4f638c5
562                 // for table of values of nodeName, nodeValue, and attributes for given node types
563
564                 // create a walker to visit all text nodes
565                 // DOMTreeWalker* walker = doc->createTreeWalker(elemRoot, DOMNodeFilter::SHOW_TEXT, NULL,
true) ;
566                 DOMTreeWalker* walker = doc->createTreeWalker(elemRoot, DOMNodeFilter::SHOW_ALL, NULL, true)
;
567                 for (DOMNode* current = walker->nextNode(); current != 0; current = walker->nextNode() ) {
568                   // note: this leaks memory!
569                   // std::cout << current->getNodeValue() << std::endl ; // .transcode();
570                   // JMH: the following was found in DOMPrint.cpp
571                   // JMH: see also http://xerces.apache.org/xerces-c/apiDocs-3/classDOMNode.html
572                   // std::cout << current->getNodeType() << " | " ;
573                   // JMH: see http://xerces.apache.org/xerces-c/apiDocs-3/classXMLString.html for equals()
```

```
574                   if ( ! XMLString::equals( XMLString::transcode(current->getNodeName()), "#text" ) )  {
575                     std::cout << "\nNode Name: " << XMLString::transcode(current->getNodeName()) << " | " ;
576                   }
577                   if ( current->getNodeValue() != NULL ) {
578                     std::cout << "Node Value: " << XMLString::transcode(current->getNodeValue()) << " | " ;
579                   }
580
581                   // see http://xerces.apache.org/xerces-c/apiDocs-3/classDOMNode.html
582                   DOMNamedNodeMap *map = current->getAttributes() ;
583                   // see http://xerces.apache.org/xerces-c/apiDocs-3/classDOMNamedNodeMap.html
584                   if ( map == NULL || map->getLength() == 0 ) {
585                     // std::cout << "no attributes" ;
586                   } else {
587                     // std::cout << map->getLength() << " attribute(s)" ;
588                     for ( unsigned int k = 0 ; k < map->getLength() ; k++ ) {
589                       std::cout << "\n  Attribute " << k+1 << ": " <<
590                           XMLString::transcode( map->item( k )->getNodeName() ) << " = " <<
591                           XMLString::transcode( map->item( k )->getNodeValue() ) << " | " ;
592                     }
593                   }
594
595                 }
596                 std::cout << std::endl;
597
598                 doc->release();
599             }
600             catch (const OutOfMemoryException&)
601             {
602                 XERCES_STD_QUALIFIER cerr << "OutOfMemoryException" << XERCES_STD_QUALIFIER endl;
603                 errorCode = 5;
604             }
605             catch (const DOMException& e)
606             {
607                 XERCES_STD_QUALIFIER cerr << "DOMException code is:  " << e.code << XERCES_STD_QUALIFIER
endl;
608                 errorCode = 2;
609             }
610             catch (...)
611             {
612                 XERCES_STD_QUALIFIER cerr << "An error occurred creating the document" <<
XERCES_STD_QUALIFIER endl;
613                 errorCode = 3;
614             }
615         }  // (inpl != NULL)
616         else
617         {
618             XERCES_STD_QUALIFIER cerr << "Requested implementation is not supported" << XERCES_STD_QUALIFIER
endl;
619             errorCode = 4;
620         }
621     }
622
623     XMLPlatformUtils::Terminate();
624     return errorCode;
625  }
```