Disclaimer and Legal Information

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL® PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER, AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

UNLESS OTHERWISE AGREED IN WRITING BY INTEL, THE INTEL PRODUCTS ARE NOT DESIGNED NOR INTENDED FOR ANY APPLICATION IN WHICH THE FAILURE OF THE INTEL PRODUCT COULD CREATE A SITUATION WHERE PERSONAL INJURY OR DEATH MAY OCCUR.

Intel may make changes to specifications and product descriptions at any time, without notice. Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined." Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The information here is subject to change without notice. Do not finalize a design with this information.

The products described in this document may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Contact your local Intel sales office or your distributor to obtain the latest specifications and before placing your product order.

Copies of documents which have an order number and are referenced in this document, or other Intel literature, may be obtained by calling 1-800-548-4725, or by visiting Intel's Web Site.

Intel processor numbers are not a measure of performance. Processor numbers differentiate features within each processor family, not across different processor families. See http://www.intel.com/products/processor_number for details.

Copyright © 2007-2008 Intel Corporation.

Licensed under the Apache License, Version 2.0 (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.apache.org/licenses/LICENSE-2.0.

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.

See the License for the specific language governing permissions and limitations under the License.

Copyright ©2004 Apache Software Foundation

Revision History

<table>
<thead>
<tr>
<th>Doc Number</th>
<th>Revision</th>
<th>Comments</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>317581</td>
<td>003</td>
<td>Intel® XML Software Suite 1.1 for C/C++ (Initial release)</td>
<td>August 2008</td>
</tr>
</tbody>
</table>
Overview

Contents

1 Overview ................................................................................................................. 4

2 Migrating from Xerces to Intel-based SAX Parsing ................................................. 6
  2.1 Parsing with Apache* Xerces* Parser .............................................................. 6
  2.2 Parsing with Intel® XML Parsing Accelerator ................................................. 8
  2.3 Mapping of Event Handling Interfaces............................................................. 10

3 Migrating from Xerces to Intel-based DOM Parsing .............................................. 14
  3.1 Parsing with Apache* Xerces* Parser .............................................................. 14
  3.2 Parsing with Intel® XML Parsing Accelerator ................................................. 16
  3.3 Mapping of DOM Node Interfaces................................................................. 17

4 Migrating from Xerces to Intel-based XML Schema Validation .......................... 19
  4.1 Validating with Apache* Xerces* Parser ....................................................... 19
  4.2 Validating with Intel® XML Schema Accelerator ......................................... 22

5 Migrating from Xalan to Intel-based XSLT Processing ........................................ 27
  5.1 Migrating a Single-threaded Application ....................................................... 28
    5.1.1 XSL Transformation with Apache* Xalan* Processor ............................. 28
    5.1.2 XSL Transformation with Intel® XSLT Accelerator ................................ 29
  5.2 Migrating a Multi-threaded Application ....................................................... 30

6 Migrating from Xalan to Intel-based XPath Navigation ...................................... 32
  6.1 XPath Navigation with Apache* Xalan* Processor ......................................... 32
  6.2 XPath Navigation with Intel® XPath Accelerator ......................................... 34

Table of Code Examples

The code examples demonstrate XML data manipulation with the analyzed libraries.
Example 1: SAX Parsing with Apache* Xerces* .................................................... 7
Example 2: SAX Parsing with Intel XML Parsing Accelerator ............................... 9
Example 3: DOM Parsing with Apache* Xerces* .................................................. 15
Example 4: DOM Parsing with Intel XML Parsing Accelerator ............................ 17
Example 5: XML Schema Validation with Apache* Xerces* .................................. 20
Example 6: XML Schema Validation with Intel XML Schema Accelerator ............ 23
Example 7: Schema Validation of SAX Source Input ........................................... 24
Example 8: Schema Validation of Stream Source Input ....................................... 24
Example 9: Schema Validation of DOM Source Input .......................................... 25
Example 10: Schema Validation of XMLDocument Source Input ......................... 26
Example 11: XSLT with Apache* Xalan* .............................................................. 28
Example 12: XSLT in a Single-threaded Application with Intel XSLT Accelerator .... 29
Example 13: XSLT in a Multi-threaded Application with Intel XSLT Accelerator ... 31
Example 14: XPath Navigation with Apache* Xalan* .......................................... 33
Example 15: XPath Navigation with Intel XPath Accelerator ................................ 35
1 Overview

Intel® XML Software Suite is a processing library that performs XML data parsing and schema validation, XSLT processing and data navigation with the XML Path (XPath) language. Intel XML Software Suite conforms to W3C* XML specification and XML processing APIs, such as Simple API for XML Processing (SAX) and Document Object Model (DOM) API.

You can now use components of Intel XML Software Suite instead of Apache* Xerces* parser and Apache* Xalan* processor for XML data manipulation, following this metric:

• For SAX or DOM processing of XML data, switch from Apache Xerces to Intel® XML Parsing Accelerator
• For XML schema validation, switch from Apache Xerces to Intel® XML Schema Accelerator
• For XSL transformation, switch from Apache Xalan to Intel® XSLT Accelerator
• For XPath navigation, switch from Apache Xalan to Intel® XPath Accelerator

The migration will allow you to claim a significant gain in performance and efficiency.

The interfaces of Xerces and Intel XML Software Suite for SAX and DOM parsing and schema validation match closely, so that the migration process is straightforward. Unlike these, XSLT processing and XPath navigation APIs differ in a number of ways: Intel XML Software Suite API employs the same paradigm as XSLT and XPath APIs in JAXP specification.

This guide contains an extensive collection of code examples to explain how to migrate from Xerces C++ 2.7.0 or 2.8.0 or from Xalan-C++ 1.10.0. Following instructions of this document, you will adjust your current environment and start using Intel XML Software Suite.

Note: In the document’s code snippets, a number of variables and declarations are omitted for simplicity.

In addition to this guide, Intel XML Software Suite includes a number of sample projects that illustrate major product capabilities. These samples will help you analyze details of the migration code and build a running application.

Please share your experience in migrating your application to Intel XML Software Suite components. You can use your registration information to provide valuable feedback.
Related Information

- *Intel XML Software Suite User Guide*, section *Quick Start* for instructions on preparing your development environment to use components of *Intel XML Software Suite*

- *Intel XML Software Suite API Reference* for details on how to use specific interface functions of *Intel XML Software Suite*

- *Intel XML Software Suite Release Notes* for a definition of standard compliance and conformance of *Intel XML Software Suite*

- *JAXP specification*, https://jaxp.dev.java.net/

- *Apache* *Xerces* *parser samples*, http://xerces.apache.org/xerces-c/samples.html
2 Migrating from Xerces to Intel-based SAX Parsing

This section of the document gives step-by-step instructions for migrating a C++ application from Apache* Xerces* versions 2.7.0 or 2.8.0 to Intel® XML Parsing Accelerator. The process of migrating SAX parsing to Intel XML Parsing Accelerator is quick and straightforward.

To illustrate the migration process step by step, this section uses the sample application SAX2Print shipped with Intel XML Software Suite. This section describes actions of the sample application after initialization and reading command-line parameters.

2.1 Parsing with Apache* Xerces* Parser

To parse XML data in the SAX mode using Apache Xerces, complete the following steps; for an illustration, see Example 1.

1. Create the SAX parser and set it to operate in the validating or non-validating mode, according to the parameters entered on the command line.

2. Enable XML schema validation for the parser: select the type of validation:
   - Val_Never: do not report validation errors
   - Val_Always: always report validation errors
   - Val_Auto: report validation errors only if grammar is specified

3. Set up the SAX2PrintHandlers event handler object and install it as the document and error handler for the parser. This object is inherited from the DefaultHandler object and supports default SAX2 interfaces.

4. Perform the SAX parsing process.

5. Release the objects involved in the parsing process.

These steps are illustrated with code in Example 1.
Example 1: SAX Parsing with Apache* Xerces*

```cpp
/// Create the SAXParser object.
SAX2XMLReader* parser;
SAX2XMLReader* reader = XMLReaderFactory::createXMLReader();
SAX2XMLReader* filter = NULL;
if (sortAttributes)
{
    filter = new SAX2SortAttributesFilter(reader);
    parser = filter;
}
else
    parser = reader;

/// Enable XML schema validation.
if (valScheme == SAX2XMLReader::Val_Auto)
{
    parser->setFeature(XMLUni::fgSAX2CoreValidation, true);
    parser->setFeature(XMLUni::fgXercesDynamic, true);
}
if (valScheme == SAX2XMLReader::Val_Never)
{
    parser->setFeature(XMLUni::fgSAX2CoreValidation, false);
}
if (valScheme == SAX2XMLReader::Val_Always)
{
    parser->setFeature(XMLUni::fgSAX2CoreValidation, true);
    parser->setFeature(XMLUni::fgXercesDynamic, false);
}

/// Set up the handler object.
SAX2PrintHandlers handler(encodingName, unRepFlags, expandNamespaces);
    parser->setContentHandler(&handler);
    parser->setErrorHandler(&handler);

/// Perform the parsing process.
parser->parse(xmlFile);
errorCount = parser->getErrorCount();

/// Release the objects.
delete reader;
if (filter)
    delete filter;
XMLPlatformUtils::Terminate();
```
2.2 Parsing with Intel® XML Parsing Accelerator

To parse XML data in the SAX mode using Intel® XML Parsing Accelerator, complete the following steps; for an illustration, see Example 2.

With, parsing in the SAX mode goes in the following stages:

1. Create a SAX parser: instantiate a factory object and create a default non-validating SAXParser object from it.

2. Enable XML schema validation for the parser.
   Unlike Xerces, Intel implementation of the SAX parser uses the direct approach: when a Schema object is created and set for the parser, it becomes a validating parser ready to validate XML files with this Schema object. You can enable an XML schema for the validating parser with or without an xsd file holding the schema. When no schema file is specified, Intel XML Parsing Accelerator sets an empty schema to initiate the validating process. Example 2 demonstrates both options.

3. Implement a SAX2CoreDumper object and set it as the content and lexical handler.

   The SAX2CoreDumper class is inherited from ContentHandler, DTDHandler, EntityResolver, and LexicalHandler, and is the equivalent of the SAX2PrintHandlers object in Xerces parser, see "...Examples\xpa\SAX2Print\SAX2CoreDumper.h".

4. Perform the SAX parsing process with exception error handling enabled in one of these methods:
   - C++ exception handling: the parser throws an exception of object type Exception, see Example 2.
   - Application-specific error handling: user application code creates its own ErrorHandler object by inheriting from the parser ErrorHandler class and implementing required methods.

5. Sequentially release the Schema, SAXParser, and SAXParserFactory objects.

   Note: Intel XML Parsing Accelerator uses the XMLChar data type throughout the interfaces. This data type is defined as:

   typedef char XMLChar
Example 2: SAX Parsing with Intel XML Parsing Accelerator

```cpp
/// Create a default SAX parser.
SAXParserFactory* saximpl = SAXParserFactory::newInstance();
SAXParser* parser = saximpl->createSAXParser();

/// Create a new instance of a Schema object.
SchemaFactory* psf = SchemaFactory::newInstance(W3C_XML_SCHEMA_NS_URI);
Schema* ps = 0;

/// If a schema file is specified explicitly, create a new object from it.
ps = psf->createSchema(xsdfile);

// or else

/// If no explicit schema file is set, create an empty schema.
ps = psf->createSchema();

/// Enable the validating parser.
parser->setSchema(ps);

/// Implement a SAX2CoreDumper object.
SAX2CoreDumper sax2CoreDumper(fout);
// Set SAX2CoreDumper as ContentHandler.
parser->setContentHandler(&sax2CoreDumper);

/// Perform the SAX parsing process with exception handling.
try {
    parser->parse(xmlFileName);
} catch (Exception& e) {
    cerr<<"SAX parser reports error"<<endl;
    cerr<<e.toString().c_str()<<endl;
}

/// Release the objects involved in the parsing process.
psf->releaseSchema(ps);
SchemaFactory::releaseInstance(psf);
saximpl->releaseSAXParser(parser);
SAXParserFactory::releaseInstance(saximpl);
```
## 2.3 Mapping of Event Handling Interfaces

Table 1 maps Apache Xerces SAX handlers to Intel parser SAX handlers represented by classes `SAX2PrintHandlers` and `SAX2CoreDumper` respectively.

### Table 1: Comparing Event Handling Interface Methods

<table>
<thead>
<tr>
<th>Apache Xerces Parser</th>
<th>Intel XML Parsing Accelerator</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Format target interface</strong></td>
<td><strong>SAX2CoreDumper</strong></td>
</tr>
<tr>
<td>void writeChars {</td>
<td>N/A</td>
</tr>
<tr>
<td>const XMLByte* const toWrite</td>
<td></td>
</tr>
<tr>
<td>}</td>
<td></td>
</tr>
<tr>
<td>void writeChars {</td>
<td></td>
</tr>
<tr>
<td>const XMLByte* const toWrite,</td>
<td></td>
</tr>
<tr>
<td>const unsigned int count,</td>
<td></td>
</tr>
<tr>
<td>XMLFormatter* const formatter</td>
<td></td>
</tr>
<tr>
<td>}</td>
<td></td>
</tr>
<tr>
<td><strong>SAX DocumentHandler interface</strong></td>
<td></td>
</tr>
<tr>
<td>void endDocument();</td>
<td>virtual void endDocument();</td>
</tr>
<tr>
<td>void endElement(</td>
<td>virtual void endElement();</td>
</tr>
<tr>
<td>const XMLCh* const uri,</td>
<td>Note: This endElement() handler returns no</td>
</tr>
<tr>
<td>const XMLCh* const localname,</td>
<td>parameters.</td>
</tr>
<tr>
<td>const XMLCh* const qname</td>
<td></td>
</tr>
<tr>
<td>)</td>
<td></td>
</tr>
<tr>
<td>void characters(</td>
<td>virtual void characters(</td>
</tr>
<tr>
<td>const XMLCh* const chars,</td>
<td>unsigned int count,</td>
</tr>
<tr>
<td>const unsigned int length</td>
<td>const XMLChar* string,</td>
</tr>
<tr>
<td>)</td>
<td>bool more</td>
</tr>
<tr>
<td></td>
<td>);</td>
</tr>
<tr>
<td></td>
<td>Receives notifications of character data.</td>
</tr>
<tr>
<td></td>
<td>Parameters:</td>
</tr>
<tr>
<td></td>
<td>• count - the counter of character data</td>
</tr>
<tr>
<td></td>
<td>that this event reported this time</td>
</tr>
<tr>
<td></td>
<td>• string - the string to character data</td>
</tr>
<tr>
<td></td>
<td>• more - the indicator of character data</td>
</tr>
<tr>
<td></td>
<td>continuity;</td>
</tr>
<tr>
<td></td>
<td>more == true indicates that another</td>
</tr>
<tr>
<td></td>
<td>characters event may follow.</td>
</tr>
<tr>
<td>void ignorableWhitespace (</td>
<td>N/A</td>
</tr>
<tr>
<td>const XMLCh* const chars,</td>
<td></td>
</tr>
<tr>
<td>const unsigned int length</td>
<td></td>
</tr>
<tr>
<td>)</td>
<td></td>
</tr>
</tbody>
</table>
### Apache Xerces Parser

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>void processingInstruction (const XMLCh* const target, const XMLCh* const data)</code>;</td>
<td>Receives notifications of a processing instruction, is invoked once for each instruction.</td>
<td>• <code>targetCount</code> - the counter of the processing instruction target • <code>targetString</code> - the pointer to the string of the processing instruction target • <code>dataCount</code> - the counter of processing instruction data • <code>dataString</code> - the pointer to the string of the processing instruction data or NULL if none has been supplied</td>
</tr>
<tr>
<td><code>void startDocument();</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>void startElement(const XMLCh* const uri, const XMLCh* const localname, const XMLCh* const qname, const Attributes&amp; attributes)</code>;</td>
<td>Receives notifications of the beginning of an element.</td>
<td>• <code>namespaceTag</code> - the prefix and its associated URI or zero if the local name has no associated namespace • <code>elemNameCount</code> - the counter of the local name of this element • <code>elemName</code> - the pointer to the local name string • <code>nAttribute</code> - the counter of attributes that this element contains • <code>attrlist</code> - the pointer to the attribute list</td>
</tr>
</tbody>
</table>

### Intel XML Parsing Accelerator

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>virtual void pi(unsigned int targetCount, const XMLChar* targetString, unsigned int dataCount, const XMLChar* dataString);</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>virtual void startDocument();</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>virtual void startElement(unsigned int namespaceTag, unsigned int elemNameCount, const XMLChar* elemName, unsigned int nAttribute, SAXAttribute** attrlist)</code>;</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### SAX ErrorHandler interface

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>void warning(const SAXParseException&amp; exc)</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>virtual void warning(Exception &amp; e)</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Apache Xerces Parser</td>
<td>Intel XML Parsing Accelerator</td>
<td></td>
</tr>
<tr>
<td>----------------------</td>
<td>-------------------------------</td>
<td></td>
</tr>
<tr>
<td>SAX2PrintHandlers</td>
<td>SAX2CoreDumper</td>
<td></td>
</tr>
<tr>
<td>);</td>
<td>);</td>
<td></td>
</tr>
<tr>
<td>Receives notification of a warning.</td>
<td>Receives notification of an error.</td>
<td></td>
</tr>
<tr>
<td>Parameters:</td>
<td>Parameters:</td>
<td></td>
</tr>
<tr>
<td>e - the Exception object (similar to Xerces SAXParseException objects)</td>
<td>e - the Exception object (similar to Xerces SAXParseException objects)</td>
<td></td>
</tr>
<tr>
<td>void error(</td>
<td>virtual void error(</td>
<td></td>
</tr>
<tr>
<td>const SAXParseException&amp; exc</td>
<td>Exception &amp; e</td>
<td></td>
</tr>
<tr>
<td>)</td>
<td>);</td>
<td></td>
</tr>
<tr>
<td>Receives notification of an error.</td>
<td>Receives notification of a fatal error.</td>
<td></td>
</tr>
<tr>
<td>Parameters:</td>
<td>Parameters:</td>
<td></td>
</tr>
<tr>
<td>e - the Exception object (similar to Xerces SAXParseException objects)</td>
<td>e - the Exception object (similar to Xerces SAXParseException objects)</td>
<td></td>
</tr>
<tr>
<td>void fatalError(</td>
<td>virtual void fatalError(</td>
<td></td>
</tr>
<tr>
<td>const SAXParseException&amp; exc</td>
<td>Exception &amp; e</td>
<td></td>
</tr>
<tr>
<td>)</td>
<td>);</td>
<td></td>
</tr>
<tr>
<td>Receives notification of a fatal error.</td>
<td>Receives notification of a fatal error.</td>
<td></td>
</tr>
<tr>
<td>Parameters:</td>
<td>Parameters:</td>
<td></td>
</tr>
<tr>
<td>e - the Exception object (similar to Xerces SAXParseException objects)</td>
<td>e - the Exception object (similar to Xerces SAXParseException objects)</td>
<td></td>
</tr>
</tbody>
</table>

**SAX DTDHandler interface**

| void notationDecl(   | virtual void notationDecl(  |
| const XMLCh* const name, |     unsigned int nameCount,  |
| const XMLCh* const publicId, |     const XMLChar* name,     |
| const XMLCh* const systemId |     unsigned int publicIdCount, |
| )                     |     const XMLChar* publicId, |
|                       |     unsigned int systemIdCount, |
|                       |     const XMLChar* systemId |
| Receives notifications of a notation declaration event. | Receives notifications of a notation declaration event. |
| Parameters:           | Parameters:                   |
| • nameCount - the counter of the notation name | • nameCount - the counter of the notation name |
| • name - the pointer to the notation name | • name - the pointer to the notation name |
| • publicIdCount - the counter of the notation public identifier or 0 if none was given | • publicIdCount - the counter of the notation public identifier or 0 if none was given |
| • publicId - the pointer to the notation public identifier or NULL if none was given | • publicId - the pointer to the notation public identifier or NULL if none was given |
| • systemIdCount - the counter of the notation system identifier or 0 if none was given | • systemIdCount - the counter of the notation system identifier or 0 if none was given |
| • systemId - the pointer to the notation system identifier or NULL if none was given | • systemId - the pointer to the notation system identifier or NULL if none was given |
### Migrating from Xerces to Intel-based SAX Parsing

<table>
<thead>
<tr>
<th>Apache Xerces Parser</th>
<th>Intel XML Parsing Accelerator</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SAX2PrintHandlers</strong></td>
<td><strong>SAX2CoreDumper</strong></td>
</tr>
<tr>
<td>void unparsedEntityDecl(</td>
<td>virtual void unparsedEntityDecl(</td>
</tr>
<tr>
<td>const XMLCh* const name,</td>
<td>unsigned int nameCount,</td>
</tr>
<tr>
<td>const XMLCh* const publicId,</td>
<td>const XMLChar* name,</td>
</tr>
<tr>
<td>const XMLCh* const systemId,</td>
<td>unsigned int publicIdCount,</td>
</tr>
<tr>
<td>const XMLCh* const notationName</td>
<td>const XMLChar* publicId,</td>
</tr>
<tr>
<td>);</td>
<td>unsigned int systemIdCount,</td>
</tr>
<tr>
<td></td>
<td>const XMLChar* systemId,</td>
</tr>
<tr>
<td></td>
<td>unsigned int notationNameCount,</td>
</tr>
<tr>
<td></td>
<td>const XMLChar* notationName</td>
</tr>
</tbody>
</table>

Receives notifications of an unparsed entity declaration event.

**Parameters:**

- **nameCount** - the counter of the unparsed entity name
- **name** - the pointer to the unparsed entity name
- **publicIdCount** - the counter of the entity public identifier or 0 if none was given
- **publicId** - the pointer to the entity public identifier or NULL if none was given
- **systemIdCount** - the counter of the entity system identifier or 0 if none was given
- **systemId** - the pointer to the entity system identifier or NULL if none was given
- **notationNameCount** - the counter of the name of the associated notation
- **notationName** - the pointer to the name of the associated notation
3 Migrating from Xerces to Intel-based DOM Parsing

This section gives step-by-step instructions for migrating a C++ application for DOM-mode XML data parsing from Xerces versions 2.7.0 or 2.8.0 to Intel® XML Parsing Accelerator. Intel® XML Parsing Accelerator and for Apache* Xerces* parser have a similar DOM parsing interface, so that the migration process is quick and straightforward.

This section relies on code from the sample application DOMTraverse shipped with the product at "..\Examples\xpa\DOMTraverse", and the sample Xerces DOM application DOMPrint, see Xerces Samples.

3.1 Parsing with Apache* Xerces* Parser

To parse XML data in the DOM mode using Apache* Xerces*, complete the following steps; for an illustration, see Example 3.

1. Create the DOM parser.
   Applications of DOM Level 3 have a different way of performing this step, see Example 3.

2. Set the parser features.
   Applications of DOM Level 3 have a different way of performing this step, see Example 3.

3. Perform the DOM parsing process.

4. Serialize the DOM node tree.
   A serializer is an instance of the DOMWriter object. Get a serializer and traverse the document tree using the DOMWriter::writeNode() method.

5. Release the objects involved in the parsing process.
Example 3: DOM Parsing with Apache* Xerces*

/// DOM Level 3 applications: create the parser.
DOMImplementation *impl = DOMImplementationRegistry::getDOMImplementation(gLS);
DOMBuilder *parser = ((DOMImplementationLS*)impl)->createDOMBuilder(DOMImplementationLS::MODE_SYNCHRONOUS, 0);

/// or else

/// Other applications: create the parser.
XercesDOMParser *parser = new XercesDOMParser;

/// DOM Level 3 applications: set parser features.
parsing->setValidationScheme(gValScheme);
parsing->setDoNamespaces(gDoNamespaces);
parsing->setDoSchema(gDoSchema);

/// or else

/// Other applications: set parser features.
parsing->setFeature(XMLUni::fgDOMNamespaces, doNamespaces);
parsing->setFeature(XMLUni::fgXercesSchema, doSchema);

/// Perform the DOM parsing process.
parsing->parse(gXmlFile);

/// Serialize the nodes in the DOM tree.
/// Get a serializer.
DOMImplementation *impl = DOMImplementationRegistry::getDOMImplementation(tempStr);
DOMWriter *theSerializer = ((DOMImplementationLS*)impl)->createDOMWriter();
/// Get the DOM representation.
DOMNode *doc = parsing->getDocument();
/// Perform the serialization.
theSerializer->writeNode(myFormTarget, *doc);

/// Delete the parser object.
delete parser;
3.2 Parsing with Intel® XML Parsing Accelerator

To parse XML data in the DOM mode using Intel® XML Parsing Accelerator, complete the following steps; for an illustration, see Example 4.

1. Create the DOM parser: from the DOMImplementationFactory object get a DOMImplementation object. Do not release it. From DOMImplementation, create a DOMParser and DOMWriter objects.

2. Set the parser features by calling setter methods of the DOMParser class. Intel XML Parsing Accelerator uses a simplified approach to creating the DOMParser object: if a Schema object is set, validation support is enabled; unlike Xerces schema, Intel schema does not require additional features to be set up. User code gets namespace support by default.

3. Perform the DOM parsing process.

4. Serialize the DOM node tree.

   A serializer is an instance of the DOMWriter object. Get a serializer and traverse the document tree using the DOMWriter::writeNode() method.

5. Release the objects involved in the parsing process: DOMParser created from DOMImplementation, the DOM tree created from DOMParser, and the DOMImplementationFactory object from memory.

These steps are demonstrated with code in Example 4.
Example 4: DOM Parsing with Intel XML Parsing Accelerator

```cpp
/// Create the DOM parser.
/// Get DOM implementation factory.
DOMImplementationFactory* domfactory = DOMImplementationFactory::newInstance();
/// Get the DOMImplementation. Do not release this object.
DOMImplementation* domimpl = domfactory->getDOMImplementation();
/// Create a DOMParser object.
DOMParser* parser = domimpl->createDOMParser();
/// Create a DOMWriter object.
DOMWriter* writer = domimpl->createDOMWriter();

/// Set parser features.
parser->setSchema(xsdfile);

/// Perform the DOM parsing process.
Document* doc = parser->parse(xmlFileName);

/// Serialize the DOM node tree.
if(parseSuccess) {
    Writer* output = NULL;
    FileWriter* fout = NULL;
    StdOutWriter* sout = NULL;
    try {
        /// Create the output writer.
        output = fout = FileWriter::createFileWriter(outFileName);
        /// or else
        output = sout = StdOutWriter::createStdOutWriter();
        /// Serialize DOM tree back to XML, set Document object & output writer.
        writer->serializeDocument(output,doc);
        /// Close the output writer when finish using it.
        output->close();
    } catch (Exception& e) {
        cerr<<"DOM writer reports error"<<endl;
        cerr<<e.toString().c_str()<<endl;
    } catch (...) {
        cerr<<"Unknown Exception!"<<endl;
    }
}

/// Release the objects involved in the parsing process.
/// Release output writer.
if(fout != NULL)
    FileWriter::releaseFileWriter(fout);
if(sout != NULL)
    StdOutWriter::releaseStdOutWriter(sout);
/// Release DOMParser, the DOM tree, and DOMImplementationFactory.
if(doc!=NULL)
    domimpl->releaseDocument(doc);
    domimpl->releaseDOMParser(parser);
    DOMImplementationFactory::releaseInstance(domfactory);
```

3.3 Mapping of DOM Node Interfaces

The `Node` interface of the DOM tree is the primary data type for the entire Document Object Mode parsing process. Xerces and Intel XML Parsing Accelerator implement
specific interfaces for node access and traversal. Table 2 shows how Xerces `DOMNode` interface maps to Intel `Node` interface.

### Table 2: Comparing Node Interface Classes

<table>
<thead>
<tr>
<th>Apache Xerces Processor <code>DOMNode</code></th>
<th>Intel XML Parsing Accelerator Node</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>DOMAttr</td>
<td>Attr</td>
<td>Refers to an attribute of an XML element.</td>
</tr>
<tr>
<td>DOMCDATASection</td>
<td>CDATASection</td>
<td>Is used to escape blocks of text containing characters that would otherwise be regarded as markup.</td>
</tr>
<tr>
<td>DOMComment</td>
<td>Comment</td>
<td>Inherits from CharacterData and represents the content of a comment, i.e., all the characters between &quot;&lt;!--&quot; and &quot;--&gt;&quot;.</td>
</tr>
<tr>
<td>DOMDocument</td>
<td>Document</td>
<td>Represents the entire XML document.</td>
</tr>
<tr>
<td>DOMDocumentFragment</td>
<td>DocumentFragment</td>
<td>Is a &quot;lightweight&quot; representation of a document.</td>
</tr>
<tr>
<td>DOMDocumentType</td>
<td>DocumentType</td>
<td>Document type held in the doctype attribute if it is not NULL.</td>
</tr>
<tr>
<td>DOMElement</td>
<td>Element</td>
<td>Represents most objects in a document tree (apart from text).</td>
</tr>
<tr>
<td>DOMEntity</td>
<td>Entity</td>
<td>Represents a parsed or unparsed entity in a document.</td>
</tr>
<tr>
<td>DOMEntityReference</td>
<td>EntityReference</td>
<td>Holds entity references from the source document or inserted by user code in the structure model.</td>
</tr>
<tr>
<td>DOMNotation</td>
<td>Notation</td>
<td>Represents a notation declared in the DTD.</td>
</tr>
<tr>
<td>DOMProcessingInstruction</td>
<td>ProcessingInstruction</td>
<td>Represents a processing instruction to keep processor-specific information in the text of the document.</td>
</tr>
<tr>
<td>DOMText</td>
<td>Text</td>
<td>Represents the textual content (termed character data in XML) of an <code>Element</code> or <code>Attr</code> object.</td>
</tr>
</tbody>
</table>
4  Migrating from Xerces to Intel-based XML Schema Validation

This section of the document gives step-by-step instructions for migrating a C++ application from Apache* Xerces* versions 2.7.0 or 2.8.0 to Intel® XML Schema Accelerator.

Intel® XML Schema Accelerator and for Apache* Xerces* parser have a similar interface for XML schema validation, so that the migration process is quick and straightforward. To migrate a Xerces-based application to Intel XML Schema Accelerator, follow instructions in this section.

To illustrate the migration process step by step, this section uses the sample application SchemaValidator shipped with Intel XML Software Suite and Xerces sample SAX2Count, see Xerces Samples. For additional usage details of SchemaValidator sample, see the directory ".\Examples\xsa\SchemaValidator".

4.1  Validating with Apache* Xerces* Parser

To validate XML data with a schema using Apache* Xerces*, complete the following steps; for an illustration, see Example 5.

1. Specify validation type based on command-line input.

2. Set schema features.

3. Create a SAX parser object and set it to validate according to features

4. Set additional parser validation features based on command–line input, such as reporting validation errors.

5. Create document and error handlers and install them to the parser. You need to create application-specific objects inheriting from DefaultHandler.

6. Parse and validate the specified XML file.

7. Delete objects related to the process.
Example 5: XML Schema Validation with Apache® Xerces®

/// Specify validation type.
if (!strncmp(argV[argInd], "-v=", 3)
    || !strncmp(argV[argInd], "-V=", 3))
{
    const char* const parm = &argV[argInd][3];
    if (!strcmp(parm, "never"))
        valScheme = SAX2XMLReader::Val_Never;
    else if (!strcmp(parm, "auto"))
        valScheme = SAX2XMLReader::Val_Auto;
    else if (!strcmp(parm, "always"))
        valScheme = SAX2XMLReader::Val_Always;
    else
    {
        XERCES_STD_QUALIFIER cerr << "Unknown -v= value: " <<
            parm << XERCES_STD_QUALIFIER endl;
        return 2;
    }
}

/// Set features of schema.
/// Use schema for validation.
doSchema = true;
/// Use full checking feature of schema.
schemaFullChecking = true;
/// Do schema identity constraint checking; default "true".
identityConstraintChecking = false;

/// Create SAXParser and set it to validate based on features.
SAX2XMLReader* parser = XMLReaderFactory::createXMLReader();
parser->setFeature(XMLUni::fgXercesSchema, doSchema);
parser->setFeature(XMLUni::fgXercesSchemaFullChecking, schemaFullChecking);
parser->setFeature(XMLUni::fgXercesIdentityConstraintChecking, identityConstraintChecking);

/// Set parser validation features based on command-line input.
/// Parser to report validation errors only
/// if a grammar is specified.
if (valScheme == SAX2XMLReader::Val_Auto)
{
    parser->setFeature(XMLUni::fgSAX2CoreValidation, true);
    parser->setFeature(XMLUni::fgXercesDynamic, true);
}
/// Parser not to report validation errors.
if (valScheme == SAX2XMLReader::Val_Never)
{
    parser->setFeature(XMLUni::fgSAX2CoreValidation, false);
}
/// Parser to always report validation errors.
if (valScheme == SAX2XMLReader::Val_Always)
{
    parser->setFeature(XMLUni::fgSAX2CoreValidation, true);
    parser->setFeature(XMLUni::fgXercesDynamic, false);
}

/// Create document and error handlers, set them to the parser.
/// Application-specific object inherited from DefaultHandler
SAX2CountHandlers handler;
parser->setContentHandler(&handler);
parser->setErrorHandler(&handler);

/// Parse and Validate XML File.
/// Reset error count.
handler.resetErrors();
parser->parse(xmlFile);
/// Delete the parser itself and terminate Xerces system.
delete parser;
XMLPlatformUtils::Terminate();
4.2 Validating with Intel® XML Schema Accelerator

To validate XML data with a schema using Intel® XML Schema Accelerator, complete the following steps; for an illustration, see Example 6.

1. Set up the Schema environment:
   a. Define schema variables and features defined in "..\Include\common\constants.h":
      - #define FEATURE_VALIDATOR_SCHEMA_FULL_CHECKING http://xml.intel.com/validator/schema_full_checking - sets the full source schema check
      - #define FEATURE_VALIDATOR_IDENTITY_CONSTRAINT_CHECKING http://xml.intel.com/validator/identity_constraint_checking - represents the identity constraint check
   b. Optionally set a customized error handler and resource resolver as shown in Example 6.

2. Create a Schema instance from:
   - An xsd file
   - Several xsd files
   - No xsd file (when the XML file defines the schema location as <xsi:schemaLocation> or <xsi:NoNamespaceSchemaLocation>)

3. Create a Validator instance.

4. Perform the validation process depending on the input type. Examples 7 to 10 use the same Validator instance to validate SAX, DOM and stream source types, and Intel-specific source format XMLDocument. For details on the XMLDocument object purpose, please refer to the user guide and the API reference for the product.

5. Release the objects involved in the validation process.
Example 6: XML Schema Validation with Intel XML Schema Accelerator

```c++
/// Set up the Schema environment.
/// Define schema variables used in sample
SchemaFactory* schemaFactory = NULL;
Schema* schema = NULL;
Validator* validator = NULL;
CustomErrorHandler schemaErrorHandler;
CustomResourceResolver resourceResolver;
/// Create a SchemaFactory instance.
try{
    schemaFactory = SchemaFactory::newInstance(schemaLanguage);
} catch(IllegalArgumentException &e){
    fprintf(stderr, "\nError:getSchemaFactory fail\n%s\n", e.getErrorMessage().c_str());
    return 1;
}
/// (optional) Set the customized ErrorHandler.
schemaFactory->setErrorHandler(&schemaErrorHandler);
/// (optional) Set the customized ResourceResolver.
if(isResourceResolverOpen)
    schemaFactory->setResourceResolver(&resourceResolver);
/// Set the full checking feature.
schemaFactory->setFeature(schema_full_checking_feature_id,schemaFullCheck);
/// setting identity constraint check
schemaFactory->setFeature(identity_constraint_checking_feature_id,identityConstraint);

/// Create an instance of a Schema object.
if(xsdArraySize != 0) {
    if(xsdArraySize == 1)//one file
        schema = schemaFactory->createSchema(xsdfile[0]);
    else  schema = schemaFactory->createSchema(xsdfile,xsdArraySize);
    /// multiple files – array of URI locations to schema files
}
else
    /// No xsd file
    schema = schemaFactory->createSchema();

/// Create the Validator instance.
try{
    validator = schema->createValidator();
} catch(ValidationException&e){

/// Perform validation depending on source type.
/// See examples 7 to 10 for sample validation of different
/// source types.
[...]

/// Release the Validator and Schema objects.
```
schema->releaseValidator(validator);
schemaFactory->releaseSchema(schema);
SchemaFactory::releaseInstance(schemaFactory);

Example 7: Schema Validation of SAX Source Input

if(strcmp(validationSource,"sax")==0){
  errorFound = false;
  SAXSource saxsource(xmlfile[i]);
  try{
    validator->validate(&saxsource);
  }catch(ValidationException&e){
    fprintf(stderr,"\nValidation Failed!
\n%s\n",e.getErrorMessage().c_str());
    schemaFactory->releaseSchema(schema);
    SchemaFactory::releaseInstance(schemaFactory);
    return 1;
  }
  if(errorFound)
    printf("\nValidation Failed!\n");
  else
    printf("\nValidation Succeeded!\n");
}

Example 8: Schema Validation of Stream Source Input

else if(strcmp(validationSource,"str")==0){
  errorFound = false;
  StreamSource streamsource(xmlfile[i]);
  try{
    validator->validate(&streamsource);
  }catch(ValidationException&e){
    fprintf(stderr,"\nValidation Failed!
\n%s\n",e.getErrorMessage().c_str());
    schemaFactory->releaseSchema(schema);
    SchemaFactory::releaseInstance(schemaFactory);
    return 1;
  }
  if(errorFound)
    printf("\nValidation Failed!\n");
  else
    printf("\nValidation Succeeded!\n");
}
Example 9: Schema Validation of DOM Source Input

```c
else if(strcmp(validationSource,"dom") == 0){
    DOMImplementationFactory* domfac = NULL;
    DOMImplementation* domimpl = NULL;
    Document* doc = NULL;
    DOMParser* parser = NULL;
    /// Create a document.
    try{
        domfac = DOMImplementationFactory::newInstance();
        domimpl = domfac->getDOMImplementation();
        //Create a DOMParser form DOM Implementation.
        parser = domimpl->createDOMParser();
        doc = parser->parse(xmlfile[i]);
    }catch(Exception& e){
        fprintf(stderr, "\nCreate document Failed!\n%s\n", e.getErrorMessage().c_str());
        if(domimpl != NULL){
            domimpl->releaseDocument(doc);
            domimpl->releaseDOMParser(parser);
        }
    }
    /// Validate the document.
    errorFound = false;
    try{
        DOMSource domsource(doc, xmlfile[i]);
        validator->validate(&domsource);
    }catch(ValidationException& e){
        fprintf(stderr, "\nValidation Failed!\n%s\n", e.getErrorMessage().c_str());
        if(domimpl != NULL){
            domimpl->releaseDocument(doc);
            domimpl->releaseDOMParser(parser);
        }
    }
    if(errorFound)
        printf("\nValidation Failed!\n");
    else
        printf("\nValidation Succeeded!\n");
    /// Release the document, parser and factory.
    domimpl->releaseDocument(doc);
    domimpl->releaseDOMParser(parser);
    DOMImplementationFactory::releaseInstance(domfac);
}
```
Example 10: Schema Validation of XMLDocument Source Input

```c
else if(strcmp(validationSource,"xdoc") == 0){
    XMLDocFactory* docFactory = NULL;
    XMLDocParser* parser = NULL;
    idoc::XMLDocument* doc = NULL;
    // Create a XML document.
    try{
        docFactory = XMLDocFactory::newInstance();
        parser = docFactory->createXMLDocParser();
        doc = parser->parse(xmlfile[i]);
    }
    catch (Exception& e)
    {        fprintf(stderr,"XmlDocparser reports
                    error:%s\n",e.toString().c_str());
            if(docFactory != NULL){
                docFactory->releaseXMLDocParser(parser);
                docFactory->releaseXMLDocument(doc);
                XMLDocFactory::releaseInstance(docFactory);
            }
            schemaFactory->releaseSchema(schema);
            SchemaFactory::releaseInstance(schemaFactory);
            return 1;
        }
    // Validate the XML document.
    errorFound = false;
    try{
        XMLDocSource xmldocsource(doc);
        validator->validate(&xmldocsource);
    }
    catch(ValidationException& e){
        fprintf(stderr,\n"Validation Failed!
                    %s\n",e.getErrorMessage().c_str());
        schemaFactory->releaseSchema(schema);
        SchemaFactory::releaseInstance(schemaFactory);
        return 1;
    }
    if(errorFound)
        printf("Validation Failed!\n");
    else
        printf("Validation Succeeded!\n");
    // Release the XML document, parser and factory.
    docFactory->releaseXMLDocParser(parser);
    docFactory->releaseXMLDocument(doc);
    XMLDocFactory::releaseInstance(docFactory);
}
```
5 Migrating from Xalan to Intel-based XSLT Processing

This section of the document gives step-by-step instructions for migrating XSL transformation from the Apache* Xalan* processor version 1.10.0 to Intel® XSLT Accelerator. The XSLT API of Intel XSLT Accelerator is modeled after the Java API for XML Processing (JAXP) interface and differs from that of Apache* Xalan* C++. However, initial migration of the XSL transformation process is straightforward and limited in scope.

The section shows how Apache Xalan processor and Intel XSLT Accelerator perform XSL transformation, and provides recommendations on tuning XSL transformation for multi-threaded applications. To make full use of the Intel XSLT Accelerator interface, familiarize yourself with the API concepts defined in Intel XML Software Suite User Guide and API Reference.
5.1 Migrating a Single-threaded Application

5.1.1 XSL Transformation with Apache* Xalan* Processor

To perform XSL transformation over your XML data using Apache* Xalan*, complete the following steps; for an illustration, see Example 11.

1. Initialize Apache Xalan processor.

2. Create the XalanTransformer object.

3. Perform the transformation.

4. Release the objects involved in the transformation, and terminate the processor.

These steps are demonstrated in Example 11.

Example 11: XSLT with Apache* Xalan*

```
/// Initialize Apache Xalan.
XalanTransformer::initialize( memoryManager );

/// Create the XalanTransformer object.
XalanTransformer theXalanTransformer( memoryManager ); Initialize Apache Xalan.

/// Perform the transformation.
theResult = theXalanTransformer.transform("foo.xml", "foo.xsl", "foo.out");

/// Release the Xalan transformer and the Xerces parser.
XalanTransformer::terminate();
XMLPlatformUtils::Terminate();
```
5.1.2 XSL Transformation with Intel® XSLT Accelerator

To perform XSL transformation over your XML data using Intel® XSLT Accelerator, complete the following steps; for an illustration, see Example 12.

1. Initialize the objects required for XSL transformation: the input and output objects and the stylesheet object.

2. Create the transformer.
   Use TransformFactory to load and compile the stylesheet into a Templates object, and create the transformer from this object.

3. Perform the transformation.

4. Release the objects involved in the transformation.

These steps are demonstrated in Example 12.

Example 12: XSLT in a Single-threaded Application with Intel XSLT Accelerator

```cpp
/// Initialize the objects required for XSL transformation.
/// Create the input source object.
StreamSource sourceStream(sourceFileName);
/// Create the input stylesheet object.
StreamSource stylesheetStream(stylesheetFileName);
/// Create the output stream.
StreamResult outStream(outFileName);

/// Create the transformer.
/// Get the TransformFactory instance.
TransformFactoy* factory = TransformFactory::newInstance();
/// Load and compile the stylesheet into a Templates object.
templates = factory->newTemplates(&stylesheetStream);
/// Create the transformer from the Templates object.
transformer = templates->newTransformer();

/// Perform the transformation.
transformer->transform(&sourceStream, &outStream);

/// Release the Transformer, Templates, and TransformFactory objects.
templates->releaseTransformer(transformer);
factory->releaseTemplates(templates);
TransformFactory::releaseInstance(factory);
```
5.2 Migrating a Multi-threaded Application

For applications that require the best performance and scalability during transformation, operating in the multithread environment can be a solution.

To illustrate the process of migrating to multithreaded transformation step by step, this section uses the sample application MultiThread shipped with Intel® XML Software Suite. For additional usage details, see the directory "..\Examples\xslta\MultiThread".

To migrate your application to the new model, complete the following steps; for an illustration, see Example 13.

1. Perform initialization.
   Initialize the transformer factory, source XML file and stylesheet in the main application thread. This application uses standard output `std::cout` as a result location. To direct the transformation output into a file or another object, initialize the output stream as well.

2. Create required fields.
   Create the threads with the thread entry procedure pointer pointing to the transform function.

3. Start each transformation thread from the function in the main thread.

4. Perform the transformation in each thread:
   a. Set the input and output of the transformation based on the input arguments of the thread initialization function.
   b. Create a new transformer.
   c. Perform the transformation.
   d. Release the transformer.

5. Release the resource in the main thread.

These steps are demonstrated in Example 13.
Example 13: XSLT in a Multi-threaded Application with Intel XSLT Accelerator

/// Perform initialization.
/// Get the TransformFactory instance.
TransformFactory* factory = TransformFactory::newInstance();
StreamSource xml(stylesheetFileName);

/// Load and compile the stylesheet.
Templates* templates = factory->newTemplates(&xsl);
/// Wrap the input source.
StreamSource xml(sourceFileName);

/// Create required threads.
/// Create input arguments for the transform function of the thread.
void* args[2];
args[0] = (void*)templates;
args[1] = (void*)(&xml);
/// Create threads with templates and input source pointers
/// in the second argument:
run_threads(thread_count,args);
/// Start each thread.
void run_threads(int thread_count,void** args)
{
    int i;
    HANDLE* threads = new HANDLE[thread_count];
    for(i=0; i<thread_count; i++)
    { HANDLE thread = CreateThread(NULL,0,(LPTHREAD_START_ROUTINE)transform_func,(void*)args,0, NULL);
        if (thread == NULL)
          ExitProcess((UINT)1);
        threads[i]= thread;
    }
    WaitForMultipleObjects(thread_count, threads, TRUE, INFINITE);
    /// Release thread resources.
    for(i=0; i<thread_count; i++)
    {
        CloseHandle(threads[i]);
    }
    delete[] threads;
}
/// Perform the transformation in each thread.
/// Set the input and output of the transformation.
Templates* templates = (Templates*)arguments[0];
Source* s = (Source*)arguments[1];
StreamResult out;  ///< Default output to stdout
/// Create a new transformer.
Transformer *transformer = templates->newTransformer();
/// Perform the transformation.
try {
    transformer->transform(s,&out);
} catch(...) {
    std::cerr << "transform error." << std::endl;
}
/// Release the transformer.
templates->releaseTransformer(transformer);
/// Release the resource in the main thread.
factory->releaseTemplates(templates);
TransformFactory::releaseInstance(factory);
6 Migrating from Xalan to Intel-based XPath Navigation

This section of the document gives step-by-step instructions for migrating XPath data manipulation to Intel XPath Accelerator from the Apache* Xalan* processor version 1.10.0. The XPath API of Intel® XPath Accelerator is modeled after the Java API for XML Processing (JAXP) interface and differs from that of Apache* Xalan* C++. However, initial migration of the XPath navigation process is straightforward and limited in scope.

To illustrate the migration process step by step, this section uses the sample application ApplyXPath shipped with Intel XML Software Suite. To make full use of the Intel XPath Accelerator interface, familiarize yourself with the API concepts defined in Intel XML Software Suite User Guide and API Reference.

6.1 XPath Navigation with Apache* Xalan* Processor

This typical Xalan-C XPath processing example uses the XPathEvaluator interface to evaluate an XPath expression from the specified context node of an XML file and displays the nodeset returned by the expression. To perform XPath processing with Xalan, complete the following steps; for an illustration, see Example 14.

1) **Initialize** the XPathEvaluator system and the processing part of the interface.

2) **Define** the XPathEvaluator object.

3) **Find** the context node.

4) **Evaluate** the expression for the node.

5) **Terminate** the XPathEvaluator object.

These steps are demonstrated in Example 14.
Migrating from Xalan to Intel-based XPath Navigation

Example 14: XPath Navigation with Apache* Xalan*

```cpp
/// Initialize the XPathEvaluator system.
XPathEvaluator::initialize();

/// Initialize the other part of the processing and parsing.
/// Initialize the XalanSourceTree subsystem.
XalanSourceTreeInit theSourceTreeInit;

/// Use the following to parse the XML file.
XalanSourceTreeDOMSupport theDOMSupport;
XalanSourceTreeParserLiaison theLiaison(theDOMSupport);

/// Hook the two together.
theDOMSupport.setParserLiaison(&theLiaison);

const XalanDOMString theFileName(argv[1]);

/// Create an input source that represents a local file.
const LocalFileInputSource theInputSource(theFileName.c_str());

/// Parse the document.
XalanDocument* const theDocument =
    theLiaison.parseXMLStream(theInputSource);

/// Define the XPathEvaluator object.
XPathEvaluator theEvaluator;

/// Find the context node.
XalanNode* const theContextNode =
    theEvaluator.selectSingleNode( //invoking XPathEvaluator::evaluate method
        theDOMSupport,
        theDocument,
        XalanDOMString(argv[2]).c_str(),
        thePrefixResolver);

/// Evaluate the expression.
constXObjectPtr theResult( //invoking XPathEvaluator::evaluate method
    theEvaluator.evaluate( //invoking XPathEvaluator::evaluate method
        theDOMSupport,
        theContextNode,
        XalanDOMString(argv[3]).c_str(),
        thePrefixResolver));

/// Result of evaluation is a generalized object type XObjectPtr.

/// Terminate the XPathEvaluator object.
XPathEvaluator::terminate();
```
6.2 XPath Navigation with Intel® XPath Accelerator

The Intel XPath Accelerator interface for evaluating XPath expressions consists of the following major classes:

- The XPathFactory class creates instances of the XPath object.
- The XPath class provides access to the XPath evaluation environment and expressions.
- The XPathExpression class encapsulates XPath expressions and allows operations on them.
- The XPathvariableResolver class provides access to the set of user-defined XPath variables

To evaluate an XPath expression for a context node, do the following; for an illustration see Example 15:

1) Initialize the XPathFactory object.
Example 15 uses the Intel-specific XML document model XPATH_INTEL_XML_DOCUMENT_MODEL for efficiency and better memory utilization.

2) Evaluate the XPath object, in two runs:
   a) Get the string result and output it.
   b) Compile the evaluation expression and get the XObject result. XObject enables access to results of different types similar to Xalan XObjectPtr.

3) Print out the result of the evaluation.

4) Release the objects involved in the evaluation process.

These steps are illustrated by Example 15. For the complete code listing, see the ApplyXPath example shipped with the product.
Example 15: XPath Navigation with Intel XPath Accelerator

```c++
#include "include/xpatha/xpath.h"
#include "include/xpatha/xpathfactory.h"
#include "include/xpatha/xpathexception.h"

/// Initialize the XPathFactory object.
XPathFactory* xpathFactory = NULL;
if(useDOMModel){
xpathFactory = XPathFactory::newInstance();
} else {
  /// using Intel-specific XML document model
  xpathFactory = XPathFactory::newInstance(XPATH_INTEL_XML_DOCUMENT_MODEL);
}
XPath* xpath = xpathFactory->newXPath();

/// Evaluate the XPath object.
try{
  /// First run: get the string result and output it.
  string stringResult = xpath->evaluateAsString(expr, source);
  printf("The string result is \n    %s\n", stringResult.c_str());

  /// Second run: get the XObject result with compile process.
  /// Compile an XPath expression for later evaluation.
  XPathExpression* xpathExpression = xpath->compile(expr);
  /// Get XObject result from XPath evaluation
  XObject* objResult = xpathExpression->evaluate(source);
  printf("The object result is \n    ");

  switch(objResult->getType()){  
    case X_BOOLEAN:{
      if(objResult->getBooleanValue()){
        printf("true\n");
      } else{
        printf("false\n");
      }
      break;
    }
    case X_NUMBER:{
      std::ostringstream ss;
      ss << (objResult->getNumberValue());
      printf("%s\n", ss.str().c_str());
      break;
    }
    case X_STRING:{
      printf("%s\n", objResult->getStringValue());
      break;
    }
    case X_USER_DEFINED:{
      printf("%p\n", objResult->getUserValue());
      break;
    }
    case X_DOMNODESET:{
      DOMNodesetIterator* it = objResult->getDOMNodeIterator();
      int count = 0;
      while(!it->end()) {
        Node* node = it->element();
        /// Do operations on node as you like
        count++;
        it->next();
      }
      printf("Total node number is %d\n", count);
```

```
break;
}

case X.XMLDOCNODESET:
    XMLDocNodesetIterator* it = objResult->getXMLDocIterator();
    int count=0;
    while(!it->end()) {
        XMLDocNode* node = it->element();
        // Do operations on node as you like
        count++;
        it->next();
    }
    printf("Total node number is %d\n", count);
    break;
}

/// Release XPath objects and input source.
xpathFactory->releaseXPath(xpath);
xPathFactory::releaseInstance(xpathFactory);

/// Release the input source.
if(useDOMModel) {
    domImpl->releaseDocument(domDoc);
    domImpl->releaseDOMParser(domParser);
    DOMImplementationFactory::releaseInstance(domFactory);
} else {
    docFactory->releaseXMLDocument(xmldoc);
    docFactory->releaseXMLDocParser(docParser);
    XMLDocFactory::releaseInstance(docFactory);
}