ABOUT XML

XML is a general-purpose specification for creating custom mark-up languages. It is classified as an extensible language because it allows its users to define their own elements. Its primary purpose is to help information systems share structured data, particularly via the Internet, and it is used both to encode documents and to serialize data. In the latter context, it is comparable with other text-based serialization languages such as JSON and YAML.

As a diverse platform, Java has several solutions for working with XML. This refcard provides developers a concise overview of the different XML processing technologies in Java, and a use case of each technology.

XML FILE SAMPLE

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<!-- DOCTYPE : DTD definitions can either be embedded in the XML document or referenced from a DTD file. Using the System keyword means that the DTD file should be in the same folder our XML file resides. -->
  <book id="_001">
    <title>XML in a Nutshell, Third Edition</title>
    <author>O'Reilly Media, Inc.</author>
    <copyright>2004</copyright>
    <isbn kind="10">0596004206</isbn>
  </book>
  <book id="_002">
    <title>XML in a Nutshell, Second Edition</title>
    <author>Erik Ray</author>
    <copyright>2003</copyright>
    <isbn kind="10">0596004206</isbn>
  </book>
  <book id="_003" image="erik_xml.jpg">
    <title>Learning XML, Second Edition</title>
    <author>O'Reilly Media, Inc.</author>
    <copyright>2007</copyright>
    <isbn kind="10">0470114878</isbn>
  </book>
</publications>
```

The above XML file defines a simple document containing information about several books. This file can be used as an input to various XML processors. It is structured with elements such as `<book>`, `<title>`, `<author>`, and `<isbn>`, which represent the book, title, author, and ISBN respectively.

XML USE CASES

### Interoperability
XML can be used independent of the target language or platform. Use XML when you need to support or interact with multiple platforms.

### Multiple output format for multiple devices
XML Transformation can help you get a required format from plain XML files. Use XML as the preferred output format when multiple output formats are required.

### Content size
Use XML when messaging and processing efficiency is less important than interoperability and availability of standard tools. Large content can create a big XML document. Use compression for XML documents or use other industry standards like ASN.1.

### Project size
For using XML, you need at least XML parsing libraries and helper classes to measure the project size and XML related required man/ hour before using XML. For small projects with simple requirements, you might not want to incur the overhead of XML.

## XML FILE SAMPLE, continued

### Line 23: namespace prefixed tag. A start tag prefixed by a namespace. End tag must be namespace prefixed in order to get a document, the end tag is line 29.

```
  <extras:book id="_003" image="erik_xml.jpg">
```

### Line 28: Attribute: an attribute is part of an element, consisting of an attribute name and its value.

```
  <isbn kind="10">0470114878</isbn>
```

### Capabilities of Element and Attribute

<table>
<thead>
<tr>
<th>Capability</th>
<th>Attribute</th>
<th>Element</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hierarchical</td>
<td>No – flat</td>
<td>Yes</td>
</tr>
<tr>
<td>Ordered</td>
<td>No – undefined</td>
<td>Yes</td>
</tr>
<tr>
<td>Complex types</td>
<td>No – string only</td>
<td>Yes</td>
</tr>
<tr>
<td>Verbose</td>
<td>Less – usually</td>
<td>More</td>
</tr>
<tr>
<td>Readability</td>
<td>Less</td>
<td>More – usually</td>
</tr>
</tbody>
</table>

## XML Use Cases

### Requirement/ Characteristic

- **Interoperability**
  - XML can be used independent of the target language or platform.
  - Use XML when you need to support or interact with multiple platforms.

- **Multiple output format for multiple devices**
  - XML Transformation can help you get a required format from plain XML files.
  - Use XML as the preferred output format when multiple output formats are required.

- **Content size**
  - Use XML when messaging and processing efficiency is less important than interoperability and availability of standard tools.
  - Large content can create a big XML document. Use compression for XML documents or use other industry standards like ASN.1.

- **Project size**
  - For Using XML, you need at least XML parsing libraries and helper classes to measure the project size and XML related required man/ hour before using XML.
  - For small projects with simple requirements, you might not want to incur the overhead of XML.

---

**Using XML in Java**

By Masoud Kalali

---

Get More Refcardz

(They're free!)

- Authoritative content
- Designed for developers
- Written by top experts
- Latest tools & technologies
- Hot tips & examples
- Bonus content online
- New issue every 1-2 weeks

Subscribe Now for FREE!

Refcardz.com
There are some technologies for searching in a XML document like XPath (http://www.w3schools.com/XPath/default.asp) and Xquery (http://www.xquery.com/) but they are relatively young and immature. Don’t use XML documents when searching is important. Instead, store the content in a traditional database, use XML databases or use XML-aware databases.

PARSING TECHNIQUES

In order to use a XML file or a XML document inside an application, it will be required to read it and tokenize it. For the XML files, this is called XML Parsing and the piece of software which performs this task is called a Parser.

There are two general parsing techniques:

- In Memory Tree: The entire document is read into memory as a tree structure which allows random access to any part of the document by the calling application.
- Streaming (Event processing): A Parser reads the document and fires corresponding event when it encounters XML entities.

Two types of parsers use streaming techniques:

- Push parsers: Parsers are in control of the parsing and the parser client has no control over the parsing flow.
- Pull parsers: The parser client is in control of the parsing and the parser goes forward to the next infoset element when it is asked to.

Following are parsers generally available in the industry:

- DOM: DOM is a tree-based parsing technique that builds up an entire parse tree in memory. It allows complete dynamic access to a whole XML document.
- SAX: SAX is an event-driven push model for processing XML. It is not a W3C standard, but it’s a very well-recognized API that most SAX parsers implement in a compliant way. Rather than building a tree representation of an entire document as DOM does, a SAX parser fires off a series of events as it reads through the document.
- StAX (JSR 173): StAX was designed as a middle ground between DOM and SAX. In StAX, the application moves the cursor forward ‘pulling’ the information from the parser as it needs. So there is no event firing by the parser or huge memory consumption. You can use 3rd party libraries for Java SE 5 and older or bundled StAX parser of Java SE 6 and above.

<table>
<thead>
<tr>
<th>Feature</th>
<th>StAX</th>
<th>SAX</th>
<th>DOM</th>
</tr>
</thead>
<tbody>
<tr>
<td>API Type</td>
<td>Pull, streaming</td>
<td>Push, streaming</td>
<td>In memory tree</td>
</tr>
<tr>
<td>Ease of Use</td>
<td>High</td>
<td>Medium</td>
<td>High</td>
</tr>
<tr>
<td>XPath Capability</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>CPU and Memory Efficiency</td>
<td>Good</td>
<td>Good</td>
<td>Varies</td>
</tr>
<tr>
<td>Forward Only</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Read XML</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Write XML</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Create, Read, Update or Delete Nodes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Best for Applications in need of:

- Streaming model
- Not modifying the document
- Memory efficiency
- XML read and XML write
- Parsing multiple documents in the same thread
- Small devices
- Looking for a certain tag
- Modifying the XML Document
- XPath, XSLT
- XML tree traversing and random access to any section
- Merging documents

All of these parsers fall under JAXP implementation. The following sample codes show how we can utilize Java SE 6 XML processing capabilities for XML parsing.

PARSING XML USING DOM

```java
15 new Instance();
16 factory.setValidating(true);
17 factory.setNamespaceAware(true);
18 factory.setAttribute("http://java.sun.com/xml/jaxp/properties
19 /schemaLanguage", "http://www.w3.org/2001/XMLSchema");
20 DocumentBuilder builder = factory.newDocumentBuilder();
21 builder.setErrorHandler(new SimpleErrorHandler());
22 Document doc = builder.parse("src/books.xml");
23 NodeList list = doc.getElementsByTagName("author");
24 for (int i = 0; i < list.getLength(); i++) {
25     Element element = (Element) list.item(i);
26     System.out.println(element.getNodeName() + " " +
27         element.getTextContent());
28     if (element.getNodeName().equalsIgnoreCase("book") {
29         System.out.println("Book ID= " + element
30             getAttribute("id");
31     }
32     if (element.getNodeName().equalsIgnoreCase("isbn") {
33         System.out.println("ISBN Kind= " + element
34             getAttribute("kind");
35     }

Line 16: In order to validate the XML using internal DTD we need only to setValidation(true).
To validate a document using DOM, ensure that there is no schema in the document, and no element prefix for our start and end tags.

Line 17: The created parser is namespace aware (the namespace prefix will be dealt with as a prefix, and not a part of the element).

Lines 18 – 19: The created parser uses internal XSD to validate the document. DomBuilderFactory instances accept several features which let developers enable or disable a functionality, one of them is validating against the internal XSD.

Line 21: Although DOM can use some default error handler, it’s usually better to set our own error handler to handle different levels of possible errors in the document. The default handler has different behaviors based on the implementation that we use. A simple error handler might be:

```java
11 public class SimpleErrorHandler implements ErrorHandler {
12         public void warning(SAXParseException e) throws SAXException {
13                 System.out.println(e.getMessage());
14             }
15             public void error(SAXParseException e) throws SAXException {
16                 System.out.println(e.getMessage());
17             }
18             public void fatalError(SAXParseException e) throws SAXException {
19                 System.out.println(e.getMessage());
20             }
21         }

```
PARSING XML USING SAX

For using SAX, we need the parser and an event handler that should respond to the parsing events. Events can be a start element event, end element event, and so forth.

A simple event handler might be:

```java
public class SimpleHandler extends DefaultHandler {
    public void startElement(String namespaceURI, String localName, String qName, Attributes atts) throws SAXException {
        if ("book".equals(localName)) {
            System.out.print("Book details:  Book ID: "+atts.getValue("")"
        } else {
            System.out.print(localName + ": ");
        }
    }

    public void characters(char[] ch, int start, int length) throws SAXException {
        System.out.println(new String(ch, start, length));
    }

    public void endElement(String namespaceURI, String localName, String qName) throws SAXException {
        if ("book".equals(localName)) {
            System.out.println("=");
        }
    }
}
```

The parser code that uses the event handler to parse the book.xml document might be:

```java
XMLInputFactory inputFactory = XMLInputFactory.newInstance();
XMLInputSource in = new XMLInputSource("src/books.xml");
SAXParserFactory factory = SAXParserFactory.newInstance();
saxParser = factory.newSAXParser();
```

```java
saxParser.setProperty("http://www.w3.org/2001/XMLSchema",
"http://java.sun.com/xml/jaxp/properties/schemaLanguage",
"http://java.sun.com/xml/jaxp/properties/schemaLocation",
"http://java.sun.com/xml/jaxp/properties/docontentHandler",
"http://java.sun.com/xml/jaxp/properties/isXMLSchema",
"http://java.sun.com/xml/jaxp/properties/preserve whitespace"
); saxParser.setProperty("http://java.sun.com/xml/jaxp/properties/validating",
"true"); saxParser.setProperty("http://java.sun.com/xml/jaxp/properties/namespaceAware",
"true");
```

```java
SAXParser saxParser = saxParserFactory.newSAXParser(); saxParser.setEventHandler(new SimpleHandler()); saxParser.parse("src/books.xml");
```

A SAMPLE USING StAX PARSER

```java
XMLInputFactory inputFactory = XMLInputFactory.newInstance(); InputStreamReader eventReader = inputFactory.createXMLStreamReader(in); saxParser = SAXParserFactory.newInstance(); saxParser.setValidating(true); saxParser.setNamespaceAware(true);
```

```java
SAXParserFactory factory = SAXParserFactory.newInstance(); saxParser = factory.newSAXParser(); saxParser.setProperty("http://www.w3.org/2001/XMLSchema",
"http://java.sun.com/xml/jaxp/properties/schemaLanguage",
"http://java.sun.com/xml/jaxp/properties/schemaLocation",
"http://java.sun.com/xml/jaxp/properties/doContentHandler",
"http://java.sun.com/xml/jaxp/properties/isXMLSchema",
"http://java.sun.com/xml/jaxp/properties/preserveWhitespace"
); saxParser.setProperty("http://java.sun.com/xml/jaxp/properties/validating",
true); saxParser.setProperty("http://java.sun.com/xml/jaxp/properties/namespaceAware",
true);
```

```java
while (eventReader.hasNext()) {
    XMLEvent event = eventReader.nextEvent();
    if (event.isStartElement()) {
        if (event.asStartElement().getName().getLocalPart().equals("book")) {
            System.out.println("=");
            event = eventReader.nextEvent();
        } else {
            System.out.print(localName + ": ");
        }
    }
    if (event.isEndElement()) {
        if (event.asEndElement().getName().getLocalPart().equals("book")
            System.out.println("=");
            event = eventReader.nextEvent();
    }
    if (event.isStartElement()) {
        if (event.asStartElement().getName().getLocalPart().equals("author")
            System.out.print("=");
            event = eventReader.nextEvent();
    }
    if (event.isStartElement()) {
        if (event.asStartElement().getName().getLocalPart().equals("isbn")
            System.out.print("=");
            event = eventReader.nextEvent();
    }
    if (event.isStartElement()) {
        if (event.asStartElement().getName().getLocalPart().equals("copyright")
            System.out.print("=");
            event = eventReader.nextEvent();
    }
    if (event.isStartElement()) {
        if (event.asStartElement().getName().getLocalPart().equals("publisher")
            System.out.print("=");
            event = eventReader.nextEvent();
    }
}
```

PARSING XML USING StAX

StAX is a streaming pull parser. It means that the parser client can ask the parser to go forward in the document when it needs.

StAX provides two sets of APIs:
- The cursor API methods return XML information as strings, which minimizes object allocation requirements.
- Iterator-based API which represents the current state of the parser as an Object. The parser client can get all the required information about the element underlying the event from the object.

Differences and features of StAX APIs

<table>
<thead>
<tr>
<th>Cursor API</th>
<th>Itertor API</th>
</tr>
</thead>
<tbody>
<tr>
<td>Best in frameworks and libraries</td>
<td>Best in applications</td>
</tr>
<tr>
<td>More memory efficient</td>
<td>XMLEvent subclasses are immutable (Direct use in other part of the application)</td>
</tr>
<tr>
<td>Better overall performance</td>
<td>New subclass of XMLEvent can be developed and used when required</td>
</tr>
<tr>
<td>Forward only</td>
<td>Applying event filters to reduce event processing costs</td>
</tr>
</tbody>
</table>

XML STRUCTURE

There are two levels of correctness of an XML document:

1. Well-formed-ness. A well-formed document conforms to all of XML’s syntax rules. For example, if a start-tag appears without a corresponding end-tag, it is not well-formed. A document that is not well-formed is not considered to be XML.

Sample characteristics:
- XML documents must have a root element
- XML elements must have a closing tag
- XML tags are case sensitive
- XML elements must be properly nested
- XML attribute values must always be quoted

2. Validity. A valid document conforms to semantic rules. The rules are included as XML schema, especially DTD. Examples of invalid documents include: if a required attribute or element is not present in the document; if the document contains an undefined element; if an element is meant to be repeated once, and appears more than once; or if the value of an attribute does not conform to the defined pattern or data type.
**XML Structure, continued**

XML validation mechanisms include using DTD and XML schema like XML Schema and RelaxNG.

**Document Type Definition (DTD)**

A DTD defines the tags and attributes used in a XML or HTML document. Elements defined in a DTD can be used, along with the predefined tags and attributes of each markup language. DTD support is ubiquitous due to its inclusion in the XML 1.0 standard.

<table>
<thead>
<tr>
<th>DTD Advantages</th>
<th>DTD Disadvantages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Easy to read and write (plain text file with a simple semicolon format).</td>
<td>No type definition system.</td>
</tr>
<tr>
<td>Can be used as an in-line definition inside the XML documents.</td>
<td>No means of element and attribute content definition and validation.</td>
</tr>
<tr>
<td>Includes #define, #include, and #ifdef, the ability to define shorthand abbreviations, external content, and some conditional parsing.</td>
<td></td>
</tr>
</tbody>
</table>

**A sample DTD document**

1. `<!DOCTYPE book SYSTEM "book.dtd">`
2. `<ELEMENT book (title, author, copyright, publisher, isbn, description)>`
3. `<ELEMENT title (#PCDATA)>`
4. `<ELEMENT author (#PCDATA)>`
5. `<ELEMENT copyright (#PCDATA)>`
6. `<ELEMENT publisher (#PCDATA)>`
7. `<ELEMENT isbn (#PCDATA)>`
8. `<ELEMENT image CDATA #IMPLIED>`
9. `<ATTLIST isbn kind (10|13) #REQUIRED>`
10. `<ATTLIST book id ID #REQUIRED image CDATA #IMPLIED>`

**XML Schema Definition (XSD)**

XSD provides the syntax and defines a way in which elements and attributes can be represented in a XML document. It also advocates the XML document should be of a specific format

**XSD Advantages:**  | **XSD Disadvantages:**
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>XSD has a much richer language for describing what element or attribute content &quot;looks like.&quot; This is related to the type system.</td>
<td>Verbose language, hard to read and write</td>
</tr>
<tr>
<td>XSD Schema supports Inheritance, where one schema can inherit from another schema. This is a great feature because it provides the opportunity for re-usability.</td>
<td>Provides no mechanism for the user to add more data types.</td>
</tr>
</tbody>
</table>

**XML Schema Definition (XSD), continued**

and specific data type. XSD is fully recommended by the W3C consortium as a standard for defining a XML Document. XSD documents are written in XML format.

**A sample XSD document**

1. `<?xml version="1.0" encoding="UTF-8"?>`
2. `<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" version="4">`
4. `<!ATTLIST isbn kind (10|13) #REQUIRED>`
5. `<!ATTLIST book id ID #REQUIRED image CDATA #IMPLIED>`
6. `<!ATTLIST isbn kind (10|13) #REQUIRED >`

**DTD Attribute Types**

<table>
<thead>
<tr>
<th>DTD Attribute Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CDATA</td>
<td>Any character string acceptable in XML</td>
</tr>
<tr>
<td>NMTOKEN</td>
<td>Close to being a XML name, first character is looser</td>
</tr>
<tr>
<td>NMTOKENS</td>
<td>One or more NMTOKEN tokens separated by white space enumeration list of the only allowed values for an attribute</td>
</tr>
<tr>
<td>ENTITY</td>
<td>Associates a name with a macro-like replacement</td>
</tr>
<tr>
<td>ENTITIES</td>
<td>White-space-separated list of ENTITY names</td>
</tr>
<tr>
<td>ID</td>
<td>XML name unique within the entire document</td>
</tr>
<tr>
<td>IDREF</td>
<td>Reference to an ID attribute within the document</td>
</tr>
<tr>
<td>IDREFS</td>
<td>White-space-separated list of IDREF tokens</td>
</tr>
<tr>
<td>NOTATION</td>
<td>Associates a name with information used by the client</td>
</tr>
</tbody>
</table>

**What a DTD can validate**

- Element nesting
- Element occurrence
- Permitted attributes of an element
- Attribute types and default values

**XML Schema Definition (XSD)**

Lines 2 – 7: Line 2 defines XML Schema namespace. Line 3 defines available schemas where it can use its vocabulary. Line 4 specifies whether locally declared elements and attributes are namespace qualified or not. A locally declared element is an element declaration directly inside a complexType (not by reference). Line 5 declares the default namespace for this schema document. Lines 6 and 7 define the namespace that a XML document can use in order to make it possible to validate it with this schema.
XML Schema Definition (XSD), continued

To validate XML files using external XSD, replace line 17 – 20 of the DOM sample with:
1 factory.setValidating(false);
2 factory.setNamespaceAware(true);
4 www.w3.org/2001/XMLSchema");
5 factory.setSchema(schemaFactory.newSchema(new Source[]{new
6 StreamSource("src/publication.xsd")});

XML Schema validation factors

<table>
<thead>
<tr>
<th>Validation factor</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Length, minLength, maxLength, maxExclusive, maxInclusive, enumeration</td>
<td>Enforces a length for the string derived value, either its maximum, minimum, maximum or minimum, inclusive and exclusive.</td>
</tr>
<tr>
<td>TotalDigits, fractionDigits</td>
<td>Restricts values to a member of a defined list</td>
</tr>
<tr>
<td>whitespace</td>
<td>Used to preserve, replace, or collapse document white space</td>
</tr>
</tbody>
</table>

XML Schema built-in types

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>anyURI</td>
<td>Uniform Resource Identifier</td>
</tr>
<tr>
<td>base64Binary</td>
<td>base64 encoded binary value</td>
</tr>
<tr>
<td>Boolean, byte, dateTime, integer, string</td>
<td>True, false or 0, 1; Signed quantity &gt;= 128 and &lt; 127; An absolute date and time; Signed integer; Unicode string</td>
</tr>
<tr>
<td>ID, IDREF, IDREFS, ENTITY, ENTITIES</td>
<td>Used to preserve, replace, or collapse document white space</td>
</tr>
</tbody>
</table>

XML Schema built-in types, continued

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NOTATION, NMTOKEN, NMTOKENS</td>
<td>Same definitions as those in DTD</td>
</tr>
<tr>
<td>language</td>
<td>&quot;xml:lang&quot; values from XML 1.0 Recommendation.</td>
</tr>
<tr>
<td>name</td>
<td>An XML name</td>
</tr>
</tbody>
</table>

DTD and XSD validation capabilities

<table>
<thead>
<tr>
<th>W3C XML Schema Features</th>
<th>DTD Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>Namespace-qualified element and attribute declarations</td>
<td>Element nesting</td>
</tr>
<tr>
<td>Simple and complex data types</td>
<td>Element occurrence</td>
</tr>
<tr>
<td>Type derivation and inheritance</td>
<td>Permitted attributes of an element</td>
</tr>
<tr>
<td>Element occurrence constraints</td>
<td>Attribute types and default values</td>
</tr>
</tbody>
</table>

XPATH

XPath is a declarative language used for referring to sections of XML documents. XPath expressions are used for locating a set of nodes in a given XML document. Many XML technologies, like XSLT and XQuery, use XPath extensively. To use these technologies, you’ll need to understand the basics of XPath. All samples in this section assume we are working on a XML document similar to the XML document on page 1.

Sample XPath Expressions and Output

<table>
<thead>
<tr>
<th>XPath Expression</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>/publications//book[contains(title, &quot;XML&quot;)]/author</td>
<td>David Hunter O’Reilly Media, Inc Erik Ray</td>
</tr>
<tr>
<td>/publications//book[title=&quot;XML&quot;] and position()=3/@id</td>
<td>303</td>
</tr>
<tr>
<td>/publications//book[title=&quot;XML&quot;] and position()=3/copyright mod 7</td>
<td>1</td>
</tr>
</tbody>
</table>

As you can see, contains and positions functions are two widely used XPath functions.

Important XPath Functions

<table>
<thead>
<tr>
<th>Operate On</th>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Node set</td>
<td>count(node-set)</td>
<td>Returns the number of nodes that are in the node set.</td>
</tr>
<tr>
<td>Node set</td>
<td>last()</td>
<td>Returns the position of the last node in the node set.</td>
</tr>
<tr>
<td>Numbers</td>
<td>ceiling(number)</td>
<td>Returns an integer value equal to or greater than the specified number.</td>
</tr>
<tr>
<td>Numbers</td>
<td>sum(node-set)</td>
<td>Returns the sum of the numerical values in the specified node set.</td>
</tr>
<tr>
<td>Boolean</td>
<td>lang(language)</td>
<td>Checks to see if the given language matches the language specified by the xsl:lang element.</td>
</tr>
<tr>
<td>Boolean</td>
<td>boolean(argument)</td>
<td>Converts the argument to Boolean.</td>
</tr>
<tr>
<td>String</td>
<td>substring-after(string1, string2)</td>
<td>Returns the portion of string1 that comes after the occurrence of string2 (which is a subset of string1).</td>
</tr>
<tr>
<td>String</td>
<td>normalize-space(string)</td>
<td>Returns the given string with no leading or trailing whitespaces, and removes sequences of whitespaces by replacing them with a single whitespace.</td>
</tr>
<tr>
<td>String</td>
<td>concat(string1, string2, stringN)</td>
<td>Returns a string containing the concatenation of the specified string arguments.</td>
</tr>
</tbody>
</table>
Using XPath in a Java Application

17 Document document;
18 DocumentBuilderFactory dbFactory = DocumentBuilderFactory.newInstance();
19 DocumentBuilder builder = dbFactory.newDocumentBuilder();
20 document = builder.parse("src/books.xml");
21 XPathFactory factory = XPathFactory.newInstance();
22 XPath xPath = factory.newXPath();
23 String copyright = xPath.evaluate("/publications/book[publisher= 'Wrox']/copyright", document);
24 System.out.println("Copyright: "+ copyright);
25 NodeList nodes = (NodeList) xPath.evaluate("//book", document, XPathConstants.NODESET);
26 String bookid = xPath.evaluate("/publications//book[contains(title,'XML') and position()=3]/@id", document);
27 System.out.println("Book ID: "+ bookid);

Using XPath in a Java Application, continued

Line 21: Prepares the XML document object to feed the XPath parser. We can use other types of InputSources.
Lines 22 – 23: Creates a XPath factory. The factory is a heavyweight object that needs to be re-used often.
Line 24: Evaluates a simple expression which returns a primary type (String).
Lines 25: The double quotation is replaced with a single quotation to make the string easy to create and read.
Line 27: An expression which returns multiple nodes. The QName is determined for the return type, and later cast to NodeList.
Lines 28: Using XPathConstants, we can determine the evaluation result type for being either a NodeList or a String.

ABOUT THE AUTHOR

Masoud Kalali

Masoud Kalali holds a software engineering degree and has been working on software development projects since 1999. He is experienced with Net but his platform of choice is Java. His experience is in software architecture, design and server side development. Masoud’s main area of research and interest is XML Web Services and Service Oriented Architecture. He has several published articles and on-going book.

Publications
- GlassFish in Action, Manning Publications

Projects
- Netbeans contributor
- GlassFish contributor

Blog
http://weblogs.java.net/blog/kalali

RECOMMENDED BOOK

Pro XML Development with Java Technology covers all the essential XML topics, including XML Schemas, addressing of XML documents through XPath, transformation of XML documents using XSLT style-sheets, storage and retrieval of XML content in native XML and relational databases, web applications based on Ajax, and SOAP/HTTP and WSDL based Web Services.

BUY NOW
books.dzone.com/books/pro-xml

Get More FREE Refcardz. Visit refcardz.com now!

Upcoming Refcardz:
- Core Mule
- Getting Started with Equinox and OSGi
- SOA Patterns
- Getting Started with EMF

Available:
- Core CSS: Part III
- Getting Started with Hibernate Search
- Core Seam
- Essential Ruby
- Essential MySQL
- JUnit and EasyMock
- Getting Started with MyEclipse
- Spring Annotations
- Core Java
- Core CSS: Part II
- PHP
- Getting Started with JPA
- JavaServer Faces
- Core CSS: Part I
- Struts2
- Core .NET
- Very First Steps in Flex

Visit refcardz.com for a complete listing of available Refcardz.