

CONTENTS INCLUDE:

- About XML
- XML File Sample
- Parsing Techniques
- XML Structure
- XPath
- Hot Tips and more...

Using XML in Java

By Masoud Kalali

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

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <!DOCTYPE publications SYSTEM "publications.dtd">
3 <publications
4 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5 xsi:schemaLocation="http://xml.dzone.org/schema/publications
6 publications.xsd"
7 xmlns="http://xml.dzone.org/schema/publications"
8 xmlns:extras="http://xml.dzone.org/schema/publications">
9   <book id="_001">
10     <title>Beginning XML, 4th Edition </title>
11     <author>David Hunter</author>
12     <copyright>2007</copyright>
13     <publisher>Wrox</publisher>
14     <isbn kind="10">0470114878</isbn>
15   </book>
16   <book id="_002">
17     <title>XML in a Nutshell, Third Edition</title>
18     <author>O'Reilly Media, Inc</author>
19     <copyright>2004</copyright>
20     <publisher>O'Reilly Media, Inc.</publisher>
21     <isbn kind="10">0596007647</isbn>
22   </book>
23   <extras:book id="_003" image="erik_xml.jpg">
24     <title>Learning XML, Second Edition</title>
25     <author>Erik Ray</author>
26     <copyright>2003</copyright>
27     <publisher>O'Reilly Media, Inc.</publisher>
28     <isbn kind="10">0596004206</isbn>
29   </extras:book>
30 </publications>
    
```

Line 1: An XML document always starts with a prolog which describes the XML file. This prolog can be minimal, e.g. `<?xml version="1.0"?>` or can contain other information. For example, the encoding:
`<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>`

Line 2: 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.

Line 3: ROOT ELEMENT: Every well-formed document should have one and only one root element. All other elements reside inside the root element.

Lines 4 – 8: namespace declaration: Line 4 defines the XSI prefix, lines 5 & 6 defines the current URL and XSD file location, line 7 defines the current document default namespace, and line 8 defines a prefix for an XML schema.

Line 20: Element: An element is composed of its start tag, end tag and the possible content which can be text or other nested elements.

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.

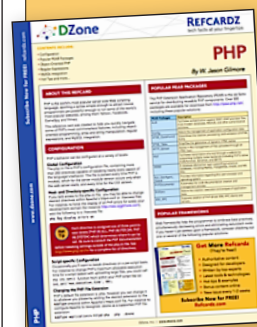
Line 28: Attribute: an attribute is part of an element, consisting of an attribute name and its value.

Capabilities of Element and Attribute

Capability	Attribute	Element
Hierarchical	No – flat	Yes
Ordered	No – undefined	Yes
Complex types	No – string only	Yes
Verbose	Less – usually	More
Readability	Less	More – usually

XML Use Cases

Requirement/Characteristic	Suitable XML Features
Interoperability	XML can be used independent of the target language or platform or target device. 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.



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

XML Use Cases, continued

Searching	<p>There are some technologies for searching in a XML document like XPath (www.w3schools.com/XPath/default.asp) and Xquery (http://www.xquery.com/) but they are relatively young and immature.</p> <p>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.</p>
------------------	---

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 median 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.

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

Parsing Techniques, continued

Best for Applications in need of:	<ul style="list-style-type: none"> • Streaming model • Not modifying the document • Memory efficiency • XML read and XML write • Parsing multiple documents in the same thread • Small devices • Looking for certain tag 	<ul style="list-style-type: none"> • Read only manipulation • Not modifying the document • Memory efficiency • Small devices • Looking for a certain tag 	<ul style="list-style-type: none"> • 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

```

14 DocumentBuilderFactory factory = DocumentBuilderFactory.
15 newInstance();
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("*");
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 set `validation(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 `Dom BuilderFactory` 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:

```

11 public class SimpleErrorHandler implements ErrorHandler {
12
13     public void warning(SAXParseException e) throws SAXException
14     {
15         System.out.println(e.getMessage());
16     }
17     public void error(SAXParseException e) throws SAXException {
18         System.out.println(e.getMessage());
19     }
20
21     public void fatalError(SAXParseException e) throws SAXException {
22         System.out.println(e.getMessage());
23     }
24 }
25 }

```

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:

```
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("id"));
        } else {
            System.out.print(localName + ": ");
        }
    }

    public void characters(char[] ch, int start, int length)
        throws SAXException {
        System.out.print(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:

```
SAXParser saxParser;
SAXParserFactory factory = SAXParserFactory.newInstance();
factory.setNamespaceAware(true);
factory.setValidating(true);
saxParser = factory.newSAXParser();
saxParser.setProperty(
    "http://java.sun.com/xml/jaxp/properties/schemaLanguage",
    "http://www.w3.org/2001/XMLSchema");
XMLReader reader = saxParser.getXMLReader();
reader.setErrorHandler(new SimpleErrorHandler());
reader.setContentHandler(new SimpleHandler());
reader.parse("src/books.xml");
```

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

Cursor API: Best in frameworks and libraries	Iterator API: Best in applications
More memory efficient	XMLEvent subclasses are immutable(Direct use in other part of the application)
Better overall performance	New subclass of XMLEvent can be developed and used when required
Forward only	Applying event filters to reduce event processing costs

A SAMPLE USING StAX PARSER

```
XMLInputFactory inputFactory = XMLInputFactory.newInstance();
InputStream in = new FileInputStream("src/books.xml");
XMLStreamReader eventReader = inputFactory.createXMLStreamReader(in);
while (eventReader.hasNext()) {
    XMLEvent event = eventReader.nextEvent();
    if (event.isEndElement()) {
        if (event.asEndElement().getName().getLocalPart()
            equals("book")) {
            event = eventReader.nextEvent();
            System.out.println("=====");
            continue;
        }
    }
    if (event.isStartElement()) {
        if (event.asStartElement().getName().getLocalPart()
            equals("title")) {
            event = eventReader.nextEvent();
            System.out.println("title: " + event.asCharacters()
                getData());
            continue;
        }
        if (event.asStartElement().getName().getLocalPart()
            equals("author")) {
            event = eventReader.nextEvent();
            System.out.println("author: " + event.asCharacters()
                getData());
            continue;
        }
        if (event.asStartElement().getName().getLocalPart()
            equals("copyright")) {
            event = eventReader.nextEvent();
            System.out.println("copyright: " + event
                asCharacters().getData());
            continue;
        }
        if (event.asStartElement().getName().getLocalPart()
            equals("publisher")) {
            event = eventReader.nextEvent();
            System.out.println("publisher: " + event.asCharacters()
                getData());
            continue;
        }
        if (event.asStartElement().getName().getLocalPart()
            equals("isbn")) {
            event = eventReader.nextEvent();
            System.out.println("isbn: " + event.asCharacters()
                getData());
            continue;
        }
    }
}
```

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.

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

A sample DTD document

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <!ELEMENT publications (book*)>
3 <!ELEMENT book (title, author+, copyright, publisher, isbn,
4 description?)>
5 <!ELEMENT title (#PCDATA)>
6 <!ELEMENT author (#PCDATA)>
7 <!ELEMENT copyright (#PCDATA)>
8 <!ELEMENT publisher (#PCDATA)>
9 <!ELEMENT isbn (#PCDATA)>
10 <!ELEMENT description (#PCDATA)>
11 <!ATTLIST book id ID #REQUIRED image CDATA #IMPLIED>
12 <!ATTLIST isbn kind (10|13) #REQUIRED >

```

Line 2: <i>publications</i> element has 0..unbounded number of <i>book</i> elements inside it.
Line 3: <i>book</i> element has one or more <i>author</i> elements, 0 or 1 <i>description</i> elements and exactly one <i>title</i> , <i>copyright</i> , <i>publisher</i> and <i>isbn</i> elements inside it.
Line 11: <i>book</i> element has two attributes, one named <i>id</i> of type ID which is mandatory, and an <i>image</i> attribute from type CDATA which is optional.
Line 12: <i>isbn</i> element has an attribute named <i>kind</i> which can have 10 or 13 as its value.

DTD Attribute Types

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

What a DTD can validate
Element nesting
Element occurrence
Permitted attributes of an element
Attribute types and default values

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

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.

XSD Advantages:	XSD Disadvantages:
XSD has a much richer language for describing what element or attribute content "looks like." This is related to the type system.	Verbose language, hard to read and write
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.	Provides no mechanism for the user to add more data types.
It is namespace aware and provides the ability to define its own data type from the existing data type.	

A sample XSD document

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
3   xmlns:extras="http://xml.dzone.org/schema/publications"
4   attributeFormDefault="unqualified" elementFormDefault="unqualified"
5   xmlns="http://xml.dzone.org/schema/publications"
6   targetNamespace="http://xml.dzone.org/schema/publications"
7   version="4">
8   <xs:element name="publications">
9     <xs:complexType>
10       <xs:sequence>
11         <xs:element minOccurs="0" maxOccurs="unbounded"
12           ref="book"/>
13       </xs:sequence>
14     </xs:complexType>
15   </xs:element>
16   <xs:element name="book">
17     <xs:complexType>
18       <xs:sequence>
19         <xs:element ref="title"/>
20         <xs:element minOccurs="1" maxOccurs="unbounded"
21           ref="author"/>
22         <xs:element ref="copyright"/>
23         <xs:element ref="publisher"/>
24         <xs:element ref="isbn"/>
25         <xs:element minOccurs="0" ref="description"/>
26       </xs:sequence>
27       <xs:attributeGroup ref="attlist.book"/>
28     </xs:complexType>
29   </xs:element>
30   <xs:element name="title" type="xs:string"/>
31   <xs:element name="author" type="xs:string"/>
32   <xs:element name="copyright" type="xs:string"/>
33   <xs:element name="publisher" type="xs:string"/>
34   <xs:element name="isbn">
35     <xs:complexType mixed="true">
36       <xs:attributeGroup ref="attlist.isbn"/>
37     </xs:complexType>
38   </xs:element>
39   <xs:element name="description" type="xs:string"/>
40   <xs:attributeGroup name="attlist.book">
41     <xs:attribute name="id" use="required" type="xs:ID"/>
42     <xs:attribute name="image"/>
43   </xs:attributeGroup>
44   <xs:attributeGroup name="attlist.isbn">
45     <xs:attribute name="kind" use="required">
46       <xs:simpleType>
47         <xs:restriction base="xs:token">
48           <xs:enumeration value="10"/>
49           <xs:enumeration value="13"/>
50         </xs:restriction>
51       </xs:simpleType>
52     </xs:attribute>
53   </xs:attributeGroup>
54 </xs:schema>

```

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 declared 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

Lines 9 – 14: An element named *publications* has a sequence of an unbounded number of books inside it.

Line 20: the element named *book* has a sequence of multiple elements inside it including *author* which at least should appear as 1, and also an element named *description* with a minimum occurrence of 0. Its maximum occurrence is the default value which is 1.

Lines 34 – 38: the *isbn* element has a group of attributes referenced by a *attlist.isbn*. This attribute group includes one attribute named *kind* (Lines 46 – 51) with a simple value. The value has a restriction which requires it to be one of the enumerated values included in the definition.



The separation of an element type definition and its use. We declared our types separately from where we referenced them (use them). ref attributes point to a declaration with the same name. Using this technique we can have separate XSD files and each of them contains definition and declarations related to one specific package. We can also import or include them in other XSD documents, if needed.



Import and include. The `import` and `include` elements help to construct a schema from multiple documents and namespaces. The `import` element brings in a schema from a different namespace, while the `include` element brings in a schema from the same namespace. When `include` is used, the target namespace of the included schema must be the same as the target namespace of the including schema. In the case of `import`, the target namespace of the included schema must be different.

To validate XML files using external XSD, replace line 17 – 20 of the DOM sample with:

```
1 factory.setValidating(false);
2 factory.setNamespaceAware(true);
3 SchemaFactory schemaFactory = SchemaFactory.newInstance("http://
4 www.w3.org/2001/XMLSchema");
5 factory.setSchema(schemaFactory.newSchema(new Source[]{new
6 StreamSource("src/publication.xsd")}));
```

XML Schema validation factors

Validation factor	Description
Length, minLength, maxLength, maxExclusive, maxInclusive, minExclusive, minInclusive	Enforces a length for the string derived value, either its maximum, minimum, maximum or minimum, inclusive and exclusive.
enumeration	Restricts values to a member of a defined list
TotalDigits, fractionDigits	Enforces total digits in a number; signs and decimal points skipped. Enforces total fractional digits in a fractional number
whiteSpace	Used to preserve, replace, or collapse document white space

XML Schema built-in types

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

XML Schema built-in types, continued

NOTATION, NMTOKEN, NMTOKENS	Same definitions as those in DTD
language	"xml:lang" values from XML 1.0 Recommendation.
name	An XML name

DTD and XSD validation capabilities

W3C XML Schema Features	DTD Features
Namespace-qualified element and attribute declarations	Element nesting
Simple and complex data types	Element occurrence
Type derivation and inheritance	Permitted attributes of an element
Element occurrence constraints	Attribute types and default values

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

XPath Expression	Output
/publications/book[publisher="Wrox"]/copyright	2007
/publications//book[contains(title,"XML")]/author	David Hunter O'Reilly Media, Inc Erik Ray
/publications//book[contains(title,"XML") and position()=3]/@id	_003
/publications//book[contains(title,"XML") and position()=3]/copyright mod 7	1

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

Important XPath Functions

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

Using XPath in a Java Application

```

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

Using XPath in a Java Application, continued

```

31 xmlDoc);
32 System.out.println("Book ID: " + bookid);
    
```

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

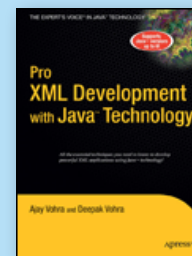
Projects

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!

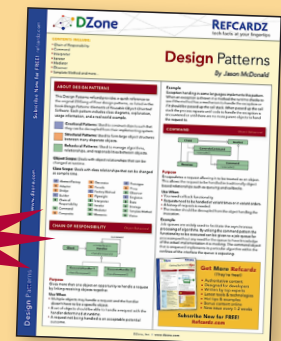
Upcoming Refcardz:

- 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.



Design Patterns
Published June 2008



DZone communities deliver over 4 million pages each month to more than 1.7 million software developers, architects and decision makers. DZone offers something for everyone, including news, tutorials, cheatsheets, blogs, feature articles, source code and more. "DZone is a developer's dream," says PC Magazine.

DZone, Inc.
1251 NW Maynard
Cary, NC 27513
888.678.0399
919.678.0300

Refcardz Feedback Welcome
refcardz@dzone.com
Sponsorship Opportunities
sales@dzone.com



\$7.95