Java and XML - Tutorial

Lars Vogel

Version 1.4

20.04.2012

Revision History			
Revision 0.1	26.02.2008	Lars Vogel	created
Revision 0.2 - 1.4	21.10.2008 - 20.04.2012	Lars Vogel	bug fixes and enhancements

Java and XML

This article give an introduction into to XML and its usage with Java. The Java Streaming API for XML (Stax) and the Java XPath library are explained and demonstrated.

Table of Contents

1. XML Introduction
    1.1. XML Overview
    1.2. XML Example
    1.3. XML Elements
2. Java XML Overview
3. Streaming API for XML (StaX)
    3.1. Overview
    3.2. Event Iterator API
    3.3. XMLEventReader - Read XML Example
    3.4. Write XML File- Example
4. XPath
    4.1. Overview
    4.2. Using XPath
5. Thank you
6. Questions and Discussion
7. Links and Literature
    7.1. Source Code
    7.2. Links and Literature
    7.3. vogella Resources

1. XML Introduction

1.1. XML Overview

XML stands for Extensible Markup Language and was defined 1998 by the World Wide Web Consortium (W3C).

A XML document consists out of elements, each element has a start tag, content and an end tag. A XML document must have exactly one root element, e.g. one tag which encloses the remaining tags. XML differentiates between capital and non-capital letters.

A XML file is required to be "well-formated".

A well-formated XML file must apply to the following conditions:

- A XML document always starts with a prolog (see below for an explanation of what a prolog is)
- Every opening tag has a closing tag.
- All tags are completely nested.

A XML file is called valid, if it is well-formated and if it is contains a link to an XML schema and is valid according to the schema.

Using XML has the following advantages vs. using a binary or unstructured format:

- XML is plain text.
- XML represents data without defining how the data should be displayed.
- XML can be transformed into other formats via XSL.
- XML can be easily processed via standard parsers.
- XML files are hierarchical.

1.2. XML Example

The following is a valid, well-formated XML file.

```
<?xml version="1.0"?>
<!-- This is a comment -->
<address>
  <name>Lars </name>
  <street> Test </street>
  <telephone number= "0123"/>
</address>
```

1.3. XML Elements

A 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, e.g. the encoding, e.g. <?xml version="1.0" encoding="UTF-8" standalone="yes" ?>

A tag which does not enclose any content is know as an "empty tag". For example: <flag/>

Comments in XML are defined as: <! COMMENT>

2. Java XML Overview

Java contains several methods to access XML. The following is a short overview of the available methods.

Java provides the DOM API (Document Object Model). In DOM your access the XML document over an object tree. DOM can be used to read and write XML files.

SAX (Simple API for XML) is an Java API to sequential reading of XML files. SAX can only read XML documents. SAX provides an Event-Driven XML Processing following the Push-Parsing Model. What this model means is that in SAX, Applications will register Listeners in the form of Handlers to the Parser and will get notified through Call-back methods. Here the SAX Parser takes the control over Application thread by Pushing Events to the Application. Both DOM and Sax are older API's and I recommend not to use them anymore.

Stax (Streaming API for XML) is an API for reading and writing XML Documents. Introduced in Java 6.0 and considered as superior to SAX and DOM.

Java Architecture for XML Binding (**JAXB**) is a Java standard that defines how Java objects are converted to XML and vice versa(specified using a standard set of mappings. JAXB defines a programmer API for reading and writing Java objects to / from XML documents and a service provider which allows the selection of the JAXB implementation JAXB applies a lot of defaults thus making

reading and writing of XML via Java very easy.

The following will explain the Stax interface, for an introduction into JAXB please see **JAXB tutorial**.

3. Streaming API for XML (StaX)

3.1. Overview

Streaming API for XML, called StaX, is an API for reading and writing XML Documents.

StaX is a Pull-Parsing model. Application can take the control over parsing the XML documents by pulling (taking) the events from the parser.

The core StaX API falls into two categories and they are listed below. They are

- Cursor API
- Event Iterator API

Applications can any of these two API for parsing XML documents. The following will focus on the event iterator API as I consider it more convenient to use.

3.2. Event Iterator API

The event iterator API has two main interfaces: XMLEventReader for parsing XML and XMLEventWriter for generating XML.

3.3. XMLEventReader - Read XML Example

This example is stored in project "de.vogella.xml.stax.reader".

Applications loop over the entire document requesting for the Next Event. The Event Iterator API is implemented on top of Cursor API.

In this example we will read the following XML document and create objects from it. file.

```
<?xml version="1.0" encoding="UTF-8"?>
<config>
  <item date="January 2009">
    <mode>1</mode>
    <unit>900</unit>
    <current>1</current>
    <interactive>1</interactive>
  </item>
  <item date="February 2009">
    <mode>2</mode>
    <unit>400</unit>
    <current>2</current>
    <interactive>5</interactive>
  </item>
  <item date="December 2009">
    <mode>9</mode>
    <unit>5</unit>
    <current>100</current>
    <interactive>3</interactive>
  </item>
</config>
```

Define therefore the following class to store the individual entries of the XML file.

```
package de.vogella.xml.stax.model;

public class Item {
  private String date;
  private String mode;
  private String unit;
  private String current;
```

```
  private String interactive;

  public String getDate() {
    return date;
  }

  public void setDate(String date) {
    this.date = date;
  }
  public String getMode() {
    return mode;
  }
  public void setMode(String mode) {
    this.mode = mode;
  }
  public String getUnit() {
    return unit;
  }
  public void setUnit(String unit) {
    this.unit = unit;
  }
  public String getCurrent() {
    return current;
  }
  public void setCurrent(String current) {
```



```
    return interactive;
  }
  public void setInteractive(String interactive) {
    this.interactive = interactive;
  }

  @Override
  public String toString() {
    return "Item [current=" + current + ", date=" + date + ", interactive="
        + interactive + ", mode=" + mode + ", unit=" + unit + "]";
  }
}
```

The following reads the XML file and creates a List of object Items from the entries in the XML file.

```
package de.vogella.xml.stax.read;

import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.InputStream;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

import javax.xml.stream.XMLEventReader;
import javax.xml.stream.XMLInputFactory;
import javax.xml.stream.XMLStreamException;
import javax.xml.stream.events.Attribute;
import javax.xml.stream.events.EndElement;
import javax.xml.stream.events.StartElement;
import javax.xml.stream.events.XMLEvent;

import de.vogella.xml.stax.model.Item;

public class StaXParser {
  static final String DATE = "date";
  static final String ITEM = "item";
  static final String MODE = "mode";
  static final String UNIT = "unit";
  static final String CURRENT = "current";
  static final String INTERACTIVE = "interactive";

  @SuppressWarnings({ "unchecked", "null" })
  public List<Item> readConfig(String configFile) {
    List<Item> items = new ArrayList<Item>();
    try {
      // First create a new XMLInputFactory
      XMLInputFactory inputFactory = XMLInputFactory.newInstance();
      // Setup a new eventReader
      InputStream in = new FileInputStream(configFile);
      XMLEventReader eventReader = inputFactory.createXMLEventReader(in);
      // Read the XML document
      Item item = null;

      while (eventReader.hasNext()) {
        XMLEvent event = eventReader.nextEvent();

        if (event.isStartElement()) {
          StartElement startElement = event.asStartElement();
          // If we have a item element we create a new item
          if (startElement.getName().getLocalPart() == (ITEM)) {
```

```
            item = new Item();
            // We read the attributes from this tag and add the date
            // attribute to our object
            Iterator<Attribute> attributes = startElement
                .getAttributes();
            while (attributes.hasNext()) {
              Attribute attribute = attributes.next();
              if (attribute.getName().toString().equals(DATE)) {
                item.setDate(attribute.getValue());
              }

            }
          }

          if (event.isStartElement()) {
            if (event.asStartElement().getName().getLocalPart()
                .equals(MODE)) {
              event = eventReader.nextEvent();
              item.setMode(event.asCharacters().getData());
              continue;
            }
          }
          if (event.asStartElement().getName().getLocalPart()
              .equals(UNIT)) {
            event = eventReader.nextEvent();
            item.setUnit(event.asCharacters().getData());
            continue;
          }

          if (event.asStartElement().getName().getLocalPart()
              .equals(CURRENT)) {
            event = eventReader.nextEvent();
            item.setCurrent(event.asCharacters().getData());
            continue;
          }

          if (event.asStartElement().getName().getLocalPart()
              .equals(INTERACTIVE)) {
            event = eventReader.nextEvent();
            item.setInteractive(event.asCharacters().getData());
            continue;
          }
        }
        // If we reach the end of an item element we add it to the list
        if (event.isEndElement()) {
          EndElement endElement = event.asEndElement();
          if (endElement.getName().getLocalPart() == (ITEM)) {
            items.add(item);
          }
        }

      }
    } catch (FileNotFoundException e) {
      e.printStackTrace();
    } catch (XMLStreamException e) {
      e.printStackTrace();
    }
    return items;
  }

}
```

You can test the parser via the following test program. Please note that the file config.xml must exist in the Java project folder.

```
package de.vogella.xml.stax.read;

import java.util.List;

import de.vogella.xml.stax.model.Item;

public class TestRead {
  public static void main(String args[]) {
    StaXParser read = new StaXParser();
    List<Item> readConfig = read.readConfig("config.xml");
    for (Item item : readConfig) {
      System.out.println(item);
    }
  }
}
```

3.4. Write XML File- Example

This example is stored in project "de.vogella.xml.stax.writer".

Lets assume you would like to write the following simple XML file.

```
<?xml version="1.0" encoding="UTF-8"?>
<config>
  <mode>1</mode>
  <unit>900</unit>
  <current>1</current>
  <interactive>1</interactive>
</config>
```

StaX does not provide functionality to format the XML file automatically. So you have to add end-of-lines and tab information to your XML file.

```
package de.vogella.xml.stax.writer;

import java.io.FileOutputStream;

import javax.xml.stream.XMLEventFactory;
import javax.xml.stream.XMLEventWriter;
import javax.xml.stream.XMLOutputFactory;
import javax.xml.stream.XMLStreamException;
import javax.xml.stream.events.Characters;
import javax.xml.stream.events.EndElement;
import javax.xml.stream.events.StartDocument;
import javax.xml.stream.events.StartElement;
import javax.xml.stream.events.XMLEvent;

public class StaxWriter {
  private String configFile;

  public void setFile(String configFile) {
    this.configFile = configFile;
  }

  public void saveConfig() throws Exception {
    // Create a XMLOutputFactory
    XMLOutputFactory outputFactory = XMLOutputFactory.newInstance();
    // Create XMLEventWriter
    XMLEventWriter eventWriter = outputFactory
        .createXMLEventWriter(new FileOutputStream(configFile));
    // Create a EventFactory
    XMLEventFactory eventFactory = XMLEventFactory.newInstance();
    XMLEvent end = eventFactory.createDTD("\n");
    // Create and write Start Tag
    StartDocument startDocument = eventFactory.createStartDocument();
    eventWriter.add(startDocument);

    // Create config open tag
    StartElement configStartElement = eventFactory.createStartElement("",
        "", "config");
    eventWriter.add(configStartElement);
    eventWriter.add(end);
    // Write the different nodes
    createNode(eventWriter, "mode", "1");
    createNode(eventWriter, "unit", "901");
    createNode(eventWriter, "current", "0");
    createNode(eventWriter, "interactive", "0");

    eventWriter.add(eventFactory.createEndElement("", "", "config"));
    eventWriter.add(end);
    eventWriter.add(eventFactory.createEndDocument());
    eventWriter.close();
  }

  private void createNode(XMLEventWriter eventWriter, String name,
      String value) throws XMLStreamException {

    XMLEventFactory eventFactory = XMLEventFactory.newInstance();
    XMLEvent end = eventFactory.createDTD("\n");
    XMLEvent tab = eventFactory.createDTD("\t");
    // Create Start node
    StartElement sElement = eventFactory.createStartElement("", "", name);
    eventWriter.add(tab);
    eventWriter.add(sElement);
    // Create Content
    Characters characters = eventFactory.createCharacters(value);
    eventWriter.add(characters);
    // Create End node
    EndElement eElement = eventFactory.createEndElement("", "", name);
    eventWriter.add(eElement);
    eventWriter.add(end);

  }

}
```

And a little test.

```
package de.vogella.xml.stax.writer;

public class TestWrite {

  public static void main(String[] args) {
    StaxWriter configFile = new StaxWriter();
    configFile.setFile("config2.xml");
    try {
      configFile.saveConfig();
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
}
```

For another (more complex example of using Stax please see **Reading and creating RSS feeds via Java (with Stax)**

4. XPath

4.1. Overview

XPath (XML Path Language) is a language for selecting / searching nodes from an XML document. Java 5 introduced the javax.xml.xpath package which provides a XPath library.

The following explains how to use XPath to query an XML document via Java.

4.2. Using XPath

The following explains how to use XPath. Create a new Java project called UsingXPath.

Create the following xml file.

```
<?xml version="1.0" encoding="UTF-8"?>
<people>
  <person>
    <firstname>Lars</firstname>
    <lastname>Vogel</lastname>
    <city>Heidelberg</city>
  </person>
  <person>
    <firstname>Jim</firstname>
    <lastname>Knopf</lastname>
    <city>Heidelberg</city>
  </person>
  <person>
    <firstname>Lars</firstname>
    <lastname>Strangelastname</lastname>
    <city>London</city>
  </person>
  <person>
    <firstname>Landerman</firstname>
    <lastname>Petrelli</lastname>
    <city>Somewhere</city>
  </person>
  <person>
    <firstname>Lars</firstname>
    <lastname>Tim</lastname>
    <city>SomewhereElse</city>
  </person>
</people>
```

Create a new package "myxml" and a new Java class "QueryXML".

```
package myxml;

import java.io.IOException;

import javax.xml.parsers.DocumentBuilder;
```

```
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;
import javax.xml.xpath.XPath;
import javax.xml.xpath.XPathConstants;
import javax.xml.xpath.XPathExpression;
import javax.xml.xpath.XPathExpressionException;
import javax.xml.xpath.XPathFactory;

import org.w3c.dom.Document;
import org.w3c.dom.NodeList;
import org.xml.sax.SAXException;

public class QueryXML {
  public void query() throws ParserConfigurationException, SAXException,
      IOException, XPathExpressionException {
    // Standard of reading a XML file
    DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
    factory.setNamespaceAware(true);
    DocumentBuilder builder;
    Document doc = null;
    XPathExpression expr = null;
    builder = factory.newDocumentBuilder();
    doc = builder.parse("person.xml");

    // Create a XPathFactory
    XPathFactory xFactory = XPathFactory.newInstance();

    // Create a XPath object
    XPath xpath = xFactory.newXPath();

    // Compile the XPath expression
    expr = xpath.compile("//person[firstname='Lars']/lastname/text()");
    // Run the query and get a nodeset
    Object result = expr.evaluate(doc, XPathConstants.NODESET);

    // Cast the result to a DOM NodeList
    NodeList nodes = (NodeList) result;
    for (int i=0; i<nodes.getLength();i++){
      System.out.println(nodes.item(i).getNodeValue());
    }

    // New XPath expression to get the number of people with name lars
    expr = xpath.compile("count(//person[firstname='Lars'])");
    // Run the query and get the number of nodes
    Double number = (Double) expr.evaluate(doc, XPathConstants.NUMBER);
    System.out.println("Number of objects " +number);

    // Do we have more then 2 people with name lars?
    expr = xpath.compile("count(//person[firstname='Lars']) >2");
    // Run the query and get the number of nodes
    Boolean check = (Boolean) expr.evaluate(doc, XPathConstants.BOOLEAN);
    System.out.println(check);

  }

  public static void main(String[] args) throws XPathExpressionException, ParserConfi
gurationException, SAXException, IOException {
    QueryXML process = new QueryXML();
    process.query();
  }
}
```

5. Thank you

Please help me to support this article:

6. Questions and Discussion

If you find errors in this tutorial please notify me (see the top of the page). Please note that due to the high volume of feedback I receive, I cannot answer questions to your implementation. Ensure you have read the **vogella FAQ**, I also don't answer questions answered in the FAQ.

7. Links and Literature

7.1. Source Code

Source Code of Examples

7.2. Links and Literature

http://www.vogella.com/articles/RSSFeed/article.html Read and write RSS feeds via Java (Stax)

http://java.sun.com/developer/technicalArticles/WebServices/jaxb/ JAXB Overview

http://www.ibm.com/developerworks/library/x-javaxpathapi.html The Java XPath API (by Elliotte Rusty Harold)

7.3. vogella Resources

vogella Training Android and Eclipse Training from the vogella team

Android Tutorial Introduction to Android Programming

GWT Tutorial Program in Java and compile to JavaScript and HTML

Eclipse RCP Tutorial Create native applications in Java

JUnit Tutorial Test your application

Git Tutorial Put everything you have under distributed version control system