JDOM and XML Parsing, Part 2

JDOM makes XML manipulation in Java easier than ever.

In the first article of this series, I introduced JDOM, an open source library for Java-optimized XML data manipulations. I explained how this alternative document object model was not built on DOM or modeled after DOM but was created to be Java-specific and thereby take advantage of Java's features, including method overloading, collections, reflection, and familiar programming idioms. I covered the important classes and started examining how to use JDOM in your applications. In this article, I'll take a look at XML Namespaces, ResultSetBuilder, XSLT, and XPath.

WORKING WITH NAMESPACES

JDOM provides robust, native support for XML Namespaces. JDOM was created after the namespace recommendation was published, so unlike other APIs there's no pre-namespace and deprecated leftovers. (See the sidebar "XML Namespaces" for more on namespaces.) In JDOM, namespaces are represented by a `Namespace` class:

```
Namespace xhtml = Namespace.getNamespace(
  "xhtml", "http://www.w3.org/1999/xhtml");
```

During construction, an object is given a name and can optionally be given a namespace:

```
elt.addContent(new Element("table", xhtml));
```

If no namespace is given, the element is constructed in "no namespace." An element's namespace is an intrinsic part of its type, so JDOM ensures that its namespace doesn't change when it moves around the document. If an element has no namespace and moves under an element that has a namespace, it explicitly does not inherit the namespace. Sometimes that causes confusion until you learn to separate the textual representation from the semantic structure.

The `XMLOutputter` class sorts out the namespace issues and ensures placement of all the "xmlns" declarations into the appropriate locations, even after a document's elements have been heavily shuffled around. By default, the class places the declarations where they're first necessary. If you want them declared further up the tree (in other words, all declarations at the root), you can use the `element.addNamespaceDeclaration()` method to provide that guidance.

All JDOM element or attribute accessor methods support an optional `namespace` argument indicating the namespace in which to look. This example points to the `xhtml` namespace:

```
List kids = html.getChildren("title", xhtml);
Element kid = html.getChild("title", xhtml);
Attribute attr = kid.getAttribute("id", xhtml);
```

When calling accessor methods it's only the Uniform Resource Identifiers (URI) that matters. That's because of how XML Namespaces work.

If no namespace instance is provided to the accessor methods, the search looks for elements without a namespace. JDOM uses a very literal representation. No namespace means no namespace, not "the parent's namespace" or anything fancy that might be subject to later subtle bugs.

MORE ABOUT RESULTSETBUILDER

ResultSetBuilder is an extension to JDOM created for people who need to treat a SQL result as an XML document. Look for it in the jdom-contrib repository in the `org.jdom.contrib.input` package.

The `ResultSetBuilder` constructor accepts a `java.sql.ResultSet` as input and returns an `org.jdom.Document` from its `build()` method.

```
Statement stmt = connection.createStatement();
ResultSet rs = stmt.executeQuery("select id, name from
registry");
ResultSetBuilder builder = new ResultSetBuilder(rs);
Document doc = builder.build();
```

GETTY ONE/EYEWIRE

BY JASON HUNTER

If you don't provide any special configuration information, the above code constructs a document similar to the following:

```
<result>
  <entry>
    <id>1</id>
    <name>Alice</name>
  </entry>
  <entry>
    <id>2</id>
    <name>Bob</name>
  </entry>
</result>
```

The ResultSetBuilder class uses the query's `ResultSet MetaData` to determine the column names and uses them as the element names. By default, the root element has the name "result," and each row has the name "entry." These names can be changed with `setRootName(String name)` and `setRowName(String name)`. You can also assign a namespace for the document with `setNamespace (Namespaces)`.

If you want a resultset column represented as an XML attribute instead of an element, you can call `setAsAttribute (StringcolumnName)` or `setAsAttribute(StringcolumnName, String attribName)`—the latter method changes the name of the attribute. You can also rename elements using `setAsElement (String columnName, String elemName)`. Both of these calls accept numbered indexes instead of names, if you prefer. The following code snippet uses these calls to reformat the generated document:

```
ResultSetBuilder builder = new ResultSetBuilder(rs);
builder.setAsAttribute("id");
builder.setAsElement("name", "fname");
Document doc = builder.build();
```

```
<result>
  <entry id="1">
    <fname>Alice</fname>
  </entry>
  <entry id="2">
    <fname>Bob</fname>
  </entry>
</result>
```

The class doesn't provide any mechanism to store XML documents in a database for later retrieval or to make XQuery calls against data in the database. To accomplish these tasks, you would need a native XML database, such as Oracle9i's feature set, XML DB.

BUILT-IN XSLT

Now that we've covered the basics of the core library, let's look at some higher-level features, such as eXtensible Stylesheet Language Transformation (XSLT) language.

XSLT provides a standard way to convert XML content from one format into another, using an XML file to handle the conversion. It's commonly used in presenting XML as an XHTML Web page, or in changing XML between one schema and another. JDOM provides built-in support for in-memory XSLT transformations, using the JAXP standard interface to XSLT engines. The key classes are `JDOMSource` and `JDOMResult` in the org.jdom.transform package. `JDOMSource` provides a JDOM document as input to the translation; `JDOMResult` catches the results as a JDOM document. Listing 1 demonstrates a complete program that

XML NAMESPACES

Namespaces can be a tricky business in XML. The goal of namespaces is similar to that of Java packages, where two classes with the same name can be differentiated by their package. Yet few people really understand what the following means:

```
<xhtml:html xmlns:xhtml="http://www.w3.org/1999/ xhtml">
    <xhtml:title>Home Page</xhtml:title>
</xhtml:html>
```

Here's the short summary. In the <html> element, the "xmlns:xhtml=" attribute name is special. It's a namespace declaration. It indicates that "xhtml" (or whatever comes after "xmlns:") should be an alias for the URI that appears in the attribute's value. Specifically, it means that for that element and any content under the declaring element, the "xhtml" prefix on a name means the item is in the "http://www.w3.org/1999/xhtml" namespace.

That namespace looks like a Web address, but it's not. It's just a string. And it's important to note that "xhtml" is an alias and not the namespace itself. There's a special namespace called the "default namespace." It's the namespace without a prefix. By using a default namespace, the earlier xhtml content could be written yet another way:

```
<html xmlns="http://www.w3.org/1999/xhtml">
  <title>Home Page</title>
</html>
```

If the "xmlns=" declaration did not exist in this XML, the elements would reside in "no namespace." In many ways, it's like the default package in Java.

performs an in-memory transformation.

You can mix and match various source and result implementations. For example, if you know you're only going to output the document and don't need the in-memory JDOM representation, you could use an import `javax.xml.transform.stream.StreamResult` instead:

```
JDOMSource source = new JDOMSource(doc);
StreamResult result = new StreamResult(System.out);
transformer.transform(source, result);
```

The downside is you lose the nice XMLOutputter formatting.

XPATH INCLUDED

XPath provides a mechanism for referring to parts of an XML document by using a string lookup path. Using XPath, you can avoid walking a document and extract just the information you want based on simple path expressions. In code releases after Beta 8, JDOM provides built-in support for XPath using the `org.jdom.xpath.XPath` class. To use it, you first construct an XPath instance by calling `XPath.newInstance()` with an expression to compile:

```
XPath xpath = XPath.newInstance("/some/xpath");
```

Then you call `selectNodes()` to get a List of answers based at a given context. The context can, for example, be the document or an element within the document.

```
List results = xpath.selectNodes(doc);
```

There are other methods for retrieving single nodes, number values, string values, and such. The default XPath implementation uses Jaxen from http://jaxen.org.

The code in Listing 2 (online at otn.oracle.com/oraclemagazine) uses XPath to pull information from a servlet deployment descriptor web.xml file. Given a web.xml file, as shown in Listing 3 (online at otn.oracle.com/oraclemagazine), the output looks like this:

```
This WAR has 2 registered servlets:
        snoop for SnoopServlet (it has 0 init params)
        file for ViewFile (it has 1 init params)
This WAR contains 3 roles:
        manager
        director
        president
```

NEW FEATURES OF JDOM BETA 9

JDOM is at Beta 8, with the next beta release producing a near-final API. Some API changes for the Beta 9 release are:

- Expose a mechanism to easily iterate over the entire document tree.
- Expose the Filter interface currently used internally so a programmer can walk only the parts of the tree that satisfy a given programmatic filter rule.
- Include a base interface to aid in these traversal models, to allow for detaching content without knowing its exact type.
- Provide a class to simplify XSLT transformations and hide the JAXP calls.
- Finalize a mechanism to handle outputting of characters not actually available in the programmer-selected output charset.

JDOM allows Java programmers to easily and effectively interact with XML. It was created to be easy to learn, powerful, and natural for Java programmers. It's released under open source with a commercial-friendly license, so it's free to use; it integrates well with JAXP, DOM, and SAX; it is an official JSR overseen by the Java Community Process. ■

codeLISTING 1: XSL Transform

```
import org.jdom.*;
import org.jdom.input.*;
import org.jdom.output.*;
import org.jdom.transform.*;
import javax.xml.transform.*;
import javax.xml.transform.stream.*;

public class XSLTransform {

  public static void main(String[] args) throws Exception {
    // Build the base document, just assume we get 2 params
    String docname = args[0];
    String sheetname = args[1];
    SAXBuilder builder = new SAXBuilder();
    Document doc = builder.build(docname);

    // Use JAXP to get a transformer
    Transformer transformer = TransformerFactory.newInstance()
      .newTransformer(new StreamSource(sheetname));

    // Run the transformation
    JDOMSource source = new JDOMSource(doc);
    JDOMResult result = new JDOMResult();
    transformer.transform(source, result);
    Document doc2 = result.getDocument();

    // Display the results
    XMLOutputter outp = new XMLOutputter();
    outp.setTextNormalize(true);
    outp.setIndent("  ");
    outp.setNewlines(true);
    outp.output(doc2, System.out);
  }
}
```

Jason Hunter *is publisher of Servlets.com and vice president of the Apache Software Foundation, and he holds a seat on the JCP Executive Committee.*

nextSTEPS

LEARN more about JDOM

Visit the JDOM project home to download software, read documentation, read the FAQ, sign up for mailing lists, and join the development discussion. Remember to read the FAQ before joining the discussion. Find out more at **http://jdom.org**.

MEET Jason Hunter

Meet Jason Hunter at OTN's OracleWorld booth. For more information and to register, go to **oracle.com/start** and enter keyword **OW2002**.