JAXP: Beyond XML Processing

Bonnie B. Ricca
Sun Microsystems
bonnie.ricca@sun.com
bonnie@bobrow.net
Contents

- Review of SAX, DOM, and XSLT
  - JAXP Overview
  - Bootstrapping
  - Source and Result
  - Validation
  - Transformation Performance
  - Conclusion
What Is SAX?

- “Simple API for XML” parsing
- “De facto” standard
  - current version is 2.0.1 (SAX2 r2)
  - developed and maintained collaboratively by SourceForge online community
- Event-based parser
  - reads XML as stream
    - can’t navigate or search document
  - low memory usage, fast performance
  - ideal for server-side apps
SAX Event Model

XMLReader

DefaultHandler

ContentHandler

ErrorHandler

DTDHandler

EntityResolver

XML Doc
Some Handler Methods

- startDocument()
- startElement(String namespaceURI, String localName, String qName, Attributes atts)
- endElement(String namespaceURI, String localName, String qName)
- endDocument()
What Is DOM?

- “Document Object Model”
- API for parsing/modifying/creating XML
  - W3C specification
    - current version is 2 (DOM2)
- Represents XML doc as tree structure
  - entire document in memory at once
    - memory-intensive
  - can traverse back and forth
  - ideal for interactive applications
DOM Nodes

- All tree objects subclass from Node
- Whitespace is parsed as a Text node

```
<order>
  <item>
    <id>388</id>
    <used /></item>
  <number>3</number>
</order>
```
What Is XSLT?

- “XSL Transformations”
  - Part of XSL (eXtensible Stylesheet Language)
- W3C Recommendation
  - 1.0 finalized in 1999
  - 2.0 currently in Working Draft stage
- Enables transformation of XML doc structure
  - stylesheets define how to reorganize data
  - can output XML, HTML, plain text
Transforming XML

```xml
<order custId="77834">
  <id>101899</id>
  <item>
    <id>421</id>
    <price currency="dollars">21.99</price>
    <number>3</number>
  </item>
  <item>
    <id>325-A</id>
    <price currency="dollars">12.49</price>
    <number>21</number>
  </item>
</order>
```

```html
<html>
  Order ID:101899
  <table border="2">
    <tr><th>Item</th><th>Price</th><th>Number</th></tr>
    <tr><td>325-A</td><td><em>12.49</em></td><td>21</td></tr>
    <tr><td>421</td><td><b>21.99</b></td><td>3</td></tr>
  </table>
</html>
```
### Drawbacks

- **DOM2 does not address:**
  - validation
  - bootstrapping
    - no vendor offers this
  - XML serialization

- **XSLT does not address:**
  - Programmatic transformation

- **SAX**
  - no major drawbacks
    - addresses portability, validation, bootstrapping
    - JAXP bases these features on SAX
Contents

- Review of SAX, DOM, and XSLT
- JAXP Overview
  - Bootstrapping
  - Source and Result
  - Validation
  - Transformation Performance
  - Conclusion
What Is JAXP?

- “Java API for XML Processing”
  - part of the Java Web Services Developer Pack
- By Sun Microsystems
  - version 1.2 released summer 2002
  - version 1.1 built in to Java 2 Platform 1.4
- Supports DOM, SAX, XSLT, DTD
  - XML-Schema support added in 1.2
- An abstraction layer
  - Can plug in any Java XML parser or XSLT processor
What Is JAXP Not?

- Does not replace SAX and DOM APIs
- Is not a new way to parse
- Does not add new features to SAX, DOM, XSLT
- Does not provide parsing or transformation functionality
  - Sun’s JAXP *distribution* includes parsers and processors
    - JAXP itself does not
## Advantages of JAXP

<table>
<thead>
<tr>
<th>Without JAXP</th>
<th>With JAXP</th>
</tr>
</thead>
</table>
| ✗ Modify and recompile when switching implementation | ✔ Portability  
  ➢ change impl at runtime |
| ✗ Must combine SAX, DOM, XSLT yourself | ✔ Flexibility  
  ➢ input/output encapsulation  
  ➢ smooth flow betw/technologies |
| ✗ Different APIs between parsers and vendors | ✔ Consistency  
  ➢ between implementations  
  ➢ between SAX, DOM, XSLT |
| ✗ Features vary from parser to parser | ✔ Power  
  ➢ SAX validation  
  ➢ XML serialization |
## JAXP Parsing Classes

<table>
<thead>
<tr>
<th>PACKAGE</th>
<th>CLASSES</th>
<th>NOTES</th>
</tr>
</thead>
</table>
| javax.xml.parsers | DocumentBuilder
              DocumentBuilderFactory
              SAXParser
              SAXParserFactory
              ParserConfigurationException
              FactoryConfigurationException
              FactoryConfigurationError | - Contains bootstrapping and JAXP configuration and Throwable classes for DOM and SAX parsers.
- The exception and error apply to both parsers. |
JAXP SAX Parsing Overview

Features
- namespaces?
- validation?

Properties
- schema
- vendor-specific

XML Doc
DefaultHandler

SAXParserFactory
SAXParser
XMLReader
JAXP DOM Parsing Overview

Attributes
- namespaces?
- validation?
- schema
- vendor-specific

DocumentBuilderFactory

XML Doc

Document-Builder

Document
JAXP DOM Creation Overview

Attributes
- namespaces?
- validation?
- schema
- vendor-specific

DocumentBuilderFactory

Document-Builder

Empty Document
Transformation with JAXP

- XSLT itself has no API
- JAXP 1.1 added the Standardization of Transformation API
  - more necessary than for parsing
- Accommodates as many inputs/outputs as possible
  - doesn’t cater to any one kind
- No effect on XSLT itself
# JAXP Transformation Packages

<table>
<thead>
<tr>
<th>PACKAGE</th>
<th>SOME CLASSES AND INTERFACES</th>
<th>NOTES</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>javax.xml.transform</code></td>
<td><code>Result Source Templates OutputKeys Transformer TransformerFactory TransformerException</code></td>
<td>Includes everything except Result/Source implementations and SAX/DOM specialty classes/interfaces</td>
</tr>
<tr>
<td><code>javax.xml.transform.dom</code></td>
<td><code>DOMResult DOMSource</code></td>
<td>DOM input/output classes</td>
</tr>
<tr>
<td><code>javax.xml.transform.sax</code></td>
<td><code>SAXResult SAXSource</code></td>
<td>SAX input/output classes</td>
</tr>
<tr>
<td><code>javax.xml.transform.stream</code></td>
<td><code>StreamResult StreamSource</code></td>
<td>Stream input/output classes</td>
</tr>
</tbody>
</table>
JAXP Transformation Overview

Attributes
- vendor-specific

TransformationFactory

Transformer

Source

StyleSheet

Result
Contents

- Review of SAX, DOM, and XSLT
- JAXP Overview

Bootstrapping
- Source and Result
- Validation
- Transformation Performance
- Conclusion
Bootstrapping Defined

- Bootstrapping
  - generic way to access your vendor’s impl of the object you need to interact with
    - SAX
      - the parser itself (XMLReader)
    - DOM
      - a DOM tree (Document)
    - Transformation
      - the XSLT processor (Transformer)
SAX Bootstrapping

- Vendor-independent coding

```java
XMLReader xmlReader = XMLReaderFactory.createXMLReader();
```

- Set property to vendor’s parser class

```java
org.xml.sax.driver=org.apache.xerces.parsers.SAXParser
```

- Can set at runtime

  ➢ no recompilation to change vendors
DOM Bootstrapping

- Use DOMImplementation factory

```java
DOMImplementation domImpl = new org.apache.xerces.dom.DOMImplementationImpl();
Document doc = domImpl.createDocument("", "rootElementName", docType);
```

- Problem:
  - literal reference to vendor’s class
    - need to import vendor’s class
    - parser change requires modification and recompile
Bootstrapping in JAXP

- Based on SAX model
  - can set system property to indicate third-party implementation
- To use built-in (“reference”) parser:
  - no configuration necessary
  - instantiate factory and main class for API
    - SAXParser
    - DocumentBuilder
    - Transformer
Getting the SAX Parser

1. Get the factory.
2. Get the parser from the factory.

```java
// Get the default factory
SAXParserFactory factory =
    SAXParserFactory.newInstance();

try {
    // Get the parser
    SAXParser saxParser = factory.newSAXParser();
    ...
}
```
Getting the DOM Parser

1. Get the factory.
2. Get the parser from the factory.

```java
Document doc;
// Get the default factory
DocumentBuilderFactory factory =
    DocumentBuilderFactory.newInstance();
try{
    // Get the parser
    DocumentBuilder builder =
        factory.newDocumentBuilder();
    ...
}
```
3. Get the new Document from the parser.

```java
Document doc;
// Get the default factory
DocumentBuilderFactory factory =
    DocumentBuilderFactory.newInstance();
try{
    // Get the parser
    DocumentBuilder builder =
        factory.newDocumentBuilder();
    // Get a new, empty Document
    doc = builder.newDocument();
    ...
}
```
Getting a Transformer

1. Get the factory.

2. Get the processor for this stylesheet from the factory.

```java
// Get the default factory
TransformerFactory factory = TransformerFactory.newInstance();

try {
    // Get the transformer for this stylesheet
    Transformer xformer = factory.newTransformer(myStylesheetStreamSource);
}
```
Bootstrap Example

- Bootstrap.java
Changing Parsers

- Change parser factory, not parser
- To override reference parser’s factory:
  1. Set system property to desired factory.
     - `javax.xml.parsers.SAXParserFactory`
     - `javax.xml.parsers.DocumentBuilderFactory`
     - `javax.xml.transform.TransformerFactory`
  2. Modify CLASSPATH to include new library.
Portability Example

- VendorChange.java
  
  to change vendor, set system property on command line:

```
java -DfactoryClass=newFactoryClass VendorChange
```
Contents

- Review of SAX, DOM, and XSLT
- JAXP Overview
- Bootstrapping

Source and Result
- Validation
- Transformation Performance
- Conclusion
Source and Result

Source

- DOMSource
  - DOM Tree
- StreamSource
  - XML Doc
- SAXSource
  - XMLReader

Transformer

- Style-sheet

Result

- DOMResult
  - DOM Tree
- SAXResult
  - XMLReader
- StreamResult
  - stdout
Transforming Your Document

1. Create Source and Result.
2. Transform the Source to the Result.

```
Document doc = builder.parse(myXMLFile);
...

// Create a DOMSource
DOMSource source = new DOMSource(doc);
...

// Create a StreamResult
StreamResult result = new StreamResult(System.out);

// Transform the source into the result
xformer.transform(source, result);
```
FEATURE String

- Can ask TransformerFactory if implementation supports Source and Result subclasses

```java
TransformerFactory tfactory = TransformerFactory.newInstance();
if (tfactory.getFeature(DOMSource.FEATURE) &&
   tfactory.getFeature(StreamResult.FEATURE))
{
    // transformation code
    ...
}
```
Serializing a DOM to XML

- DOM 2 does not address serialization
  - available through vendor-specific methods only

- Use Transformer with:
  - DOMSource
  - StreamResult
  - “identity transformation”
    - empty stylesheet
    - transforms data format, not data structure
Identity Transformation


2. Create Transformer with no stylesheet.

```java
Transformer xformer = xformerFactory.newTransformer();
```

3. Create DOMSource and StreamResult.

```java
DOMSource source = new DOMSource(doc);
StreamResult result =
    new StreamResult(FileWriter(new File("out.xml")));
```


```java
xformer.transform(source, result);
```
Example

- DOMSerializer.java
Contents

- Review of SAX, DOM, and XSLT
- JAXP Overview
- Bootstrapping
- Source and Result

Validation
- Transformation Performance
- Conclusion
**SAX ErrorHandler**

1. **Fatal Errors**
   - generated by default error event handler

2. **Non-Fatal Errors**
   - only from validating parsers
   - ignored by default

3. **Warnings**
   - usually from validating parsers
   - ignored by default
Validating Against DTD

1. Turn on validation in factory.
2. Register ErrorHandler with parser.
   - SAX: use DefaultHandler and pass to parse()
   - DOM: use setErrorHandler() on builder

```java
SAXParserFactory factory =
    SAXParserFactory.newInstance();
// set parser to validate (this is all that's needed for DTD)
factory.setValidating(true);
try {
    // Parse the input
    SAXParser saxParser = factory.newSAXParser();
saxParser.parse( xmlFile, handler );
}
```
Validating Against Schema

DTD steps, plus:

3. Set factory to use namespaces

4. Set two properties:
   - schema file (if not declared in XML doc)
   - schema language
Schema Validation in SAX

```java
SAXParserFactory factory = SAXParserFactory.newInstance();
// set parser to validate (this is all that’s needed for DTD)
factory.setValidating(true);
// enable namespaces (required for XML-Schema)
factory.setNamespaceAware(true);
try {
    SAXParser saxParser = factory.newSAXParser();
    // ** set schema properties on parser **
                        new File("mySchema.xsd"));
    saxParser.setProperty("http://java.sun.com/xml/jaxp/properties/schemaLanguage",
                        "http://www.w3.org/2001/XMLSchema");
}  
```
Schema Validation in DOM

```java
DocumentBuilderFactory factory =
    DocumentBuilderFactory();
// set parser to validate (this is all that's needed for DTD)
factory.setValidating(true);
// enable namespaces (required for XML-Schema)
factory.setNamespaceAware(true);
try {
    // ** set schema attributes on factory **
    factory.setAttribute("http://java.sun.com/xml/jaxp/properties/schemaSource",
                      new File("mySchema.xsd"));
    factory.setAttribute("http://java.sun.com/xml/jaxp/properties/schemaLanguage",
                      "http://www.w3.org/2001/XMLSchema");
    DocumentBuilder builder = factory.newDocumentBuilder();
    builder.setErrorHandler(errorHandler);
}
```
Transformation with Validation

- Standardization of Transformation API does not address validation

- Use a SAXSource
  1. Configure an XMLReader to validate.
  2. Create a SAXSource with reader and input file.
  3. Give it to a Transformer.
     - Transformer configures itself as a ContentHandler
     - calls parse() on XMLReader
Creating a SAXSource

1. Create a SAXParserFactory.
   - configure it for namespaces and validation

   ```java
   SAXParserFactory factory = SAXParserFactory.newInstance();
   factory.setNamespaceAware(true);
   factory.setValidating(true);
   ```

2. Create a SAXParser.

   ```java
   SAXParser parser = factory.newSAXParser();
   ```
Creating a SAXSource (Continued)

3. Get parser’s XMLReader.
   - register an ErrorHandler

   ```java
   XMLReader reader = parser.getXMLReader();
   reader.setErrorHandler(eHandler);
   ```

4. Create a SAXSource with:
   - XMLReader
   - source XML file

   ```java
   SAXSource source = new SAXSource(reader, sourceFile);
   ```
Example

- XformValidator.java
Contents

- Review of SAX, DOM, and XSLT
- JAXP Overview
- SAX and JAXP
- DOM and JAXP
- Changing Parsers

Transformation Performance

- Conclusion
Caching with Templates

- XSLT uses tons of CPU and memory
- TransformerFactory not good for high-volume, **concurrent** processing
  - not thread-safe; need one per thread
  - loads, parses and processes stylesheet for each Transformer
- Use a Templates object
  - thread-safe
  - can create multiple Transformers concurrently
  - loads, parses, and processes stylesheet only once
Using Templates

- Get Templates from factory
- Get Transformer from Templates

```java
//Transformer xformer =
//    xformerFactory.newTransformer(stylesheet);

Templates templates =
    xformerFactory.newTemplates(stylesheet);
Transformer xformer = templates.newTransformer();
```

- Create one Templates for each stylesheet
  - need only one TransformerFactory
  - greatly improves performance for multi-threaded creation of Transformers
Xalan-Java Reference Processor

- Apache’s Xalan-Java (Xalan-J)
  - JAXP 1.2 reference XSLT processor

- Has two components
  - Xalan-J Interpretive
    - interprets stylesheets
    - JAXP 1.2 default component
  - XSLTC
    - compiles stylesheets into Java byte code (“translets”)
    - uses less memory
Interpreted vs. Compiled

**Xalan**
- Interprets stylesheets
- Slower than XSLTC (except first time thru)
- Supports SQL

Perfect for one-time transformations and working with databases.

**XSLTC**
- Compiles and caches stylesheets for subsequent use
- A little slow first time thru
- Much faster and uses less memory after that

Perfect for repeated transformations in a low-resource environment.
Using XSLTTC with JAXP

1. Make sure you’re using JAXP 1.2
   - $JAVA_HOME/jre/lib/endorsed should contain:
     - xalan.jar
     - xsltc.jar

2. Set system property
   - javax.xml.transform.TransformerFactory=
     org.apache.xalan.xslt.trax.TransformerFactoryImpl

- No code changes necessary!
XSLTC and JAXP Transformation

- Style-sheet
- Transformer Factory
- Transformer
- Template byte code
- Client
- XML Source
- XML Result
- XSLT compiler
- Translet
- XSLTC Runtime Library
Example

- XformTimer.java
Smart Transformer Switch

- A third Xalan-J implementation
  - uses Interpretive for TransformFactory/Transformer
  - uses XSLTC for Templates
- Ideal for applications that mix one-time transformations with repetitive ones
- Set system property to
  org.apache.xalan.xslt.trax.SmartTransformerFactoryImpl
Contents

- Review of SAX, DOM, and XSLT
- JAXP Overview
- SAX and JAXP
- DOM and JAXP
- Changing Parsers
- Transforming with JAXP

Conclusion
Conclusion

JAXP gives you:

- **consistency**
  - between technologies
  - between vendors

- **flexibility**
  - to change implementations
  - to vary input and output formats

- **streamlined processing**
  - to seamlessly pass data between technologies

- **power**
  - to combine the strengths of all the technologies
JAXP 1.3 Preview

- Some features to look forward to:
  - DOM Level 3 support
  - XPath 1.0 support
- Will be part of J2SE 1.5
Resources

- **JAXP 1.2 specification**

- **JAXP 1.2 installation (WSDP 1.3)**

- **JAXP tutorial**

- **Apache Xalan-J and XSLTC**