3.3 JAXP: Java API for XML Processing

- How can applications use XML processors?
  - A Java-based answer: through JAXP
  - An overview of the JAXP interface
    - What does it specify?
    - What can be done with it?
    - How do the JAXP components fit together?

[java.sun.com/xml/jaxp/properties/schemaSource]

3.3: (XML APIs) JAXP

Later Versions: 1.2

- JAXP 1.2 added property-strings for setting the language and source of a scheme used for validation

- in JAXP 1.3 through the setSchema() method of the Factory classes (used to initialize SAXParsers or DOM DocumentBuilders)

JAXP: XML processor plugin (1)

- Vendor-independent method for selecting processor implementations at run time
  - principally through system properties
  - in JAXP 1.3 through the setSchema(Schema) method of the Factory classes (used to initialize SAXParsers or DOM DocumentBuilders)

JAXP: Basic Functionality

- Parsing using SAX 2.0 or DOM Level 2
- Transformation using XSLT
  - (more about XSLT later)
- Adds functionality missing from SAX 2.0 and DOM Level 2:
  - controlling validation and handling of parse errors
  - error handling can be controlled in SAX
  - loading and saving of DOM Document objects

JAXP: XML processor plugin (2)

- By default, reference implementations used
  - Apache Xerces as the XML parser
  - Xalan (JAXP 1.2) / XSLTC (JAXP 1.3) as the XSLT processor

JAXP Parsing API

- Included in JAXP package
  - javax.xml.parsers

- Used for invoking and using SAX ...

```java
SAXParserFactory spf = SAXParserFactory.newInstance();
```

- and DOM parser implementations:
  ```java
  DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
  ```
**JAXP: Using a SAX parser (1)**

- We have already seen this:
  ```java
  try {
      SAXParserFactory spf = SAXParserFactory.newInstance();
      SAXParser saxParser = spf.newSAXParser();
      XMLReader xmlReader = saxParser.getXMLReader();
      ContentHandler handler = new myHandler();
      xmlReader.setContentHandler(handler);
      xmlReader.parse(systemIdOrInputSrc);
  } catch (Exception e) {
      System.err.println(e.getMessage());
      System.exit(1);
  }
  ```

**JAXP: Using a SAX parser (2)**

- Parsing a file into a DOM Document:
  ```java
  try {
      DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
      DocumentBuilder builder = dbf.newDocumentBuilder();
      Document domDoc = builder.parse(fileNameOrURIetc);
      builder.setErrorHandler(new myErrHandler());
  } catch (ParserConfigurationException e) {
      e.printStackTrace();
      System.exit(1);
  } catch (SAXException e) {
      System.err.println(e.getMessage());
      System.exit(1);
  }
  ```

**JAXP: Using a DOM parser (1)**

- Errors of DOM parsing can be handled
  - by creating a SAX ErrorHandler
  - to implement error, fatalError and warning methods
  and passing it to the DocumentBuilder:
  ```java
  builder.setErrorHandler(new myErrHandler());
  Document domDoc = builder.parse(fileName);  // to get a new DocumentBuilder
  DocumentBuilder builder = newDocumentBuilder();
  DocumentBuilderFactory dbf = dbf.newDocumentBuilder();
  DOMDocument domDoc = domDoc.parse(fileNameOrURIetc);
  ```

**JAXP: Using a DOM parser (2)**

- Parser properties can be configured:
  - for both SAXParserFactories and DocumentBuilderFactory:
  ```java
  factory.setValidating(true/false);
  factory.setNamespaceAware(true/false);
  ```

**DOM building in JAXP**

- DOM on top of SAX - So what?

**JAXP: Controlling parsing (1)**

- Further DocumentBuilderFactory configuration methods to control the form of the resulting DOM Document:
  ```java
dbf.setIgnoringComments(true/false)
dbf.setIgnoringElementContentWhitespace(true/false)
dbf.setCoalescing(true/false)
  ```

**JAXP: Controlling parsing (2)**

- Errors of DOM parsing can be handled
  - for multiple implementations move between languages
  - which a specific set of classes each class file
  - which allow move between languages with minimal relearning, but
  - with minimal relearning, but...

**DOM vs. Other Java/XML APIs**

- The others may be more convenient to use, but ...
  ```java
  SAXParserFactory saxFactory = SAXParserFactory.newInstance();
  ```

- JDOM (www.jdom.org), DOM4J (www.dom4j.org), JAXB (java.sun.com/xml/jaxb)

- "The DOM offers not only the ability to move between languages with minimal relearning, but to move between multiple implementations in a single language - which a specific set of classes such as JDOM can't support"
JAXP Transformation API

- **Package javax.xml.transform**
  - TransformerFactory and Transformer classes; initialization similar to parser factories and parsers
  - Allows application to apply a Transformer to a Source document to get a Result document
  - Transformer can be created
    - from an XSLT script
    - without instructions → an identity transformation from a Source to the Result

JAXP Transformation Packages

- Classes to create Source and Result objects from DOM, SAX and I/O streams defined in packages
  - javax.xml.transform.dom, javax.xml.transform.sax, and javax.xml.transform.stream
- Identity transformation to an output stream is a vendor-neutral way to serialize DOM documents
  - as an alternative to DOM3 Save

JAXP: Using Transformers (1)

- Transformation Source & Result
  - Transformation Source object can be
    - (a node of) a DOM tree
    - a SAX XMLReader or
    - an input stream
  - Transformation Result object can be
    - (a node of) a DOM tree
    - a SAX ContentHandler or
    - an output stream

Source-Result combinations

- Serializing a DOM Document as XML text
  - By an identity transformation to an output stream:
    - TransformerFactory tfFactory = TransformerFactory.newInstance();
    - Transformer transformer = tfFactory.newTransformer();
    - DOMSource source = new DOMSource(myDOMDoc);
    - StreamResult result = new StreamResult(System.out);
    - transformer.transform(source, result);

Controlling the form of the result?

- We could specify the requested form of the result by an XSLT script, say, in file saveSpec.xslt:

  ```xml
  <xsl:transform version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:output method="xml" indent="yes" doctype-system="reglist.dtd" />
    <xsl:template match="/">
      <!-- copy whole document: -->
      <xsl:copy-of select="." />
    </xsl:template>
  </xsl:transform>
  ```

Creating an XSLT Transformer

- Create a tailored transformer:
  ```java
  StreamSource saveSpecSrc = new StreamSource(
    new File("saveSpec.xslt"));
  Transformer transformer = TransformerFactory.newTransformer(saveSpecSrc);
  // and use it to transform a Source to a Result,
  // as before
  ```
- The Source of transformation instructions could be given also as a DOMSource or SAXSource
Transformation OutputProperties

Transformer myTr = tFactory.newTransformer();
myTr.setOutputProperty(OutputKeys.INDENT, "yes");
myTr.setOutputProperty(OutputKeys.DOCTYPE_SYSTEM, "square6 square6");
myTr.setOutputProperty(OutputKeys.INDENT, "yes");

// Then use it as before

Equivalent to the previous "saveSpec.xslt" Transformer

Stylesheet Parameters

- Can also pass parameters to a transformer created from a script like this:
  <xsl:transform ...>
  <xsl:output method="text"/>
  <xsl:variable name="In">10</xsl:variable>
  <xsl:template match="/">
    <xsl:value-of select="2*$In/>
  </xsl:template>
  </xsl:transform>

  myTr.setParameter("In", 10)

JAXP Validation

- JAXP 1.3 introduced also a Validation framework
  - based on familial Factory pattern, to provide independence of schema language and implementation
    » SchemaFactory → Schema → Validator
  - separates validation from parsing
    » say, to validate an in-memory DOM subtree

Validation Example: "Xeditor"

- Xeditor, an experimental XML editor
  - to experiment and demonstrate JAXP-based on-the-fly multi-schema validation


Look & Feel of "Xeditor"

- A Validator created when the user selects "Xeditor"
  - if WF check as XML or DTD
    validate using DTD or against schema

Different Schemas and Schema Languages

- A Validator created when the user selects Schema

  - externalFactory (XML schema)
    externalFactory (result)
    Schema
    Validator
    ExternalErrorHandler
    Invalid DTD Handler

Efficiency of In-Memory Validation

- Is brute-force re-validation too inefficient?
- No: Delays normally unnoticeable

Event-driven document validation

- Modified document passed to the Validator
  - errors caught as SAX parse exceptions
JAXP: Summary

- An interface for using XML Processors
  - SAX/DOM parsers, XSLT transformers
  - schema-based validators (in JAXP 1.3)
- Supports pluggability of XML processors
- Defines means to control parsing, and handling of parse errors (through SAX ErrorHandlers)
- Defines means to create and save DOM Documents