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?

[Partly based on tutorial "An Overview of the APIs" at http://java.sun.com/xml/jaxp/dirt/1.1/docs/tutorial/overview /3_apis.html, from which also some graphics are borrowed]

JAXP 1.1

- An interface for "plugging-in" and using XML processors in Java applications
  - Includes packages
    - org.xml.sax: SAX 2.0 interface
    - org.w3c.dom: DOM Level 2 interface
    - java.xml.parsers: initialization and use of parsers
    - javax.xml.transform: initialization and use of transformers (XSLT processors)
  - Included in JDK since version 1.4

JAXP 1.2

- Current version, since June 2002
- Adds property strings for controlling schema-based validation:
  - either on a SAXParser or (DOM) DocumentBuilderFactory (to be discussed)

JAXP: XML processor plugin (1)

- Vendor-independent method for selecting processor implementation at run time
  - principally through system properties
    - java.xml.parsers.DocumentBuilderFactory
    - javax.xml.transform.TransformerFactory
  - Set on command line (for example, to use Apache Xerces as the DOM implementation):
    - java

JAXP: XML processor plugin (2)

- Set during execution (→ Saxon as the XSLT impl):
- By default, reference implementations used
  - Apache Crimson/Xerces as the XML parser
  - Apache Xalan as the XSLT processor
- Currently supported only by a few compliant XML processors:
  - Parsers: Apache Crimson and Xerces, Aelfred, Oracle XML Parser for Java
  - XSLT transformers: Apache Xalan, Saxon

JAXP: Functionality

- Parsing using SAX 2.0 or DOM Level 2
- Transformation using XSLT
  - (We’ll study XSLT in detail 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, by implementing ErrorHandler methods
  - loading and saving of DOM Document objects

JAXP Parsing API

- Included in JAXP package
  - javax.xml.parsers
- Used for invoking and using SAX...
  - SAXParserFactory spf = SAXParserFactory.newInstance();
- and DOM parser implementations:
  - DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();

JAXP: Using a SAX parser (1)
JAXP: Using a SAX parser (2)

- We have already seen this:

  ```java
  SAXParserFactory spf = SAXParserFactory.newInstance();
  try {
    SAXParser saxParser = spf.newSAXParser();
    XMLReader xmlReader = saxParser.getXMLReader();
    ... xmlReader.setContentHandler(handler);
    xmlReader.parse(fileName); ...
  } catch (Exception e) {
    System.err.println(e.getMessage());
    System.exit(1); }
  }
  ```

JAXP: Using a DOM parser (1)

- DOM building in JAXP

JAXP: Using a DOM parser (2)

- Parsing a file into a DOM Document:

  ```java
  DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
  try {
    DocumentBuilder builder = dbf.newDocumentBuilder();
    Document domDoc = builder.parse(fileName);
  } catch (ParserConfigurationException e) {
    e.printStackTrace();
  } catch (SAXException e) {
    e.printStackTrace();
    System.exit(1); }
  }
  ```

JAXP: Controlling parsing (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(myErrorHandler);
    domDoc = builder.parse(fileName);
    ```

- Parser properties can be configured:
  - for both SAXParserFactories and DocumentBuilderFactory properties (before parser/builder creation):

    ```java
    factory.setValidating(true/false);
    factory.setNamespaceAware(true/false);
    ```

JAXP: Controlling parsing (2)

- Further DocumentBuilderFactory configuration methods to control the form of the resulting DOM Document:

  ```java
  setIgnoringComments(true/false)
  setIgnoringElementContentWhitespace(true/false)
  setCoalescing(true/false)
  ```

JAXP Transformation API

- earlier known as TrAX
- Allows application to apply a Transformer to a Source document to get a Result document
- Transformer can be created
  - from XSLT transformation instructions (to be discussed later)
  - without instructions
    - gives an identity transformation, which simply copies the Source to the Result

JAXP: Using Transformers (1)
JAXP Transformation APIs

- javax.xml.transform:
  - Classes Transformer and TransformerFactory; initialization similar to parsers and parser factories
- Transformation Source object can be
  - a DOM tree, a SAX XMLReader or an input stream
- Transformation Result object can be
  - a DOM tree, a SAX ContentHandler or an output stream

JAXP Transformation Packages (2)

- 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 (and the only option in JAXP)
  - I would recommend using the JAXP interfaces until the DOM's own load/save module becomes available
  - Joe Kesselman, IBM & W3C DOM WG

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 encoding="ISO-8859-1" indent="yes"
  doctype="system"="reglist.dtd" />
  <xsl:template match="/">
    <!-- copy whole document: -->
    <xsl:copy-of select="." /></xsl:template>
  </xsl:transform>
  ```

Serializing a DOM Document as XML text

- By an identity transformation to an output stream:
  ```java
  TransformerFactory tFactory = TransformerFactory.newInstance();
  // Create an identity transformer:
  Transformer transformer = tFactory.newTransformer();
  DOMSource source = new DOMSource(myDOMdoc);
  StreamResult result = new StreamResult(System.out);
  transformer.transform(source, result);
  ```

Creating an XSLT Transformer

- Then create a tailored transformer:
  ```java
  StreamSource saveSpecSrc = new StreamSource("saveSpec.xslt");
  Transformer transformer = tFactory.newTransformer(saveSpecSrc);
  // and use it to transform a Source to a Result, as before
  ```

Other Java APIs for XML

- JDOM
  - a Java-specific variant of W3C DOM
  - http://www.jdom.org/
- DOM4J (http://www.dom4j.org/)
  - roughly similar to JDOM; richer set of features:
  - powerful navigation with integrated XPath support
- JAXB (Java Architecture for XML Binding)
  - compiles DTDs to DTD-specific classes for reading, manipulating and writing valid documents
  - http://java.sun.com/xml/jaxb/

Why then stick to DOM?

- Other document models might be more convenient to use, but ...
  "The DOM offers not only the ability to move between languages with minimal releasing, but to move between multiple implementations in a single language – which a specific set of classes such as JDOM can't support"
  - Joe Kesselman, IBM & W3C DOM WG
## JAXP: Summary

- An interface for using XML Processors
  - SAX/DOM parsers, XSLT transformers
- Supports pluggability of XML processors
- Defines means to control parsing, and handling of parse errors (through SAX ErrorHandlers)
- Defines means to write out DOM Documents
- Included in Java 2