COIN-078B
Internet Programming with XML

Java and XML DOM

Module Objectives

- Create an XML parser in Java
- Parse & display an entire document
- Filter XML documents
- Create a windowed browser
- Navigate & modify XML Documents
- Transform XML Documents with XSLT

Reading Assignments:

- **Textbook:** chapters 10 & 11

Java API for XML Processing (JAXP)

The Java API for XML Processing (JAXP), included in Sun J2SE 1.4 and later, is a fairly complete set of APIs for processing XML data in Java. JAXP leverages the parser standards SAX (Simple API for XML Parsing) and W3C DOM (Document Object Model) so that you can choose to parse your data as a stream of events or to build an object representation of it. JAXP also supports the XSLT (XML Stylesheet Language Transformations) standard, giving you control over the presentation of the data and enabling you to convert the data to other XML documents or to other formats, such as HTML. JAXP also provides namespace support, allowing you to work with DTDs that might otherwise have naming conflicts.

Most current XML parsers for Java support JAXP including Crimson, Ælfred, Xerces, Piccolo, and the Oracle XML Parser for Java.

The following implementations support the transform component of JAXP and also bundle a parser (in alphabetical order):

Name	Parser Implementation	XSLT Processor Implementation	Comment
Apache Xalan-J	Xerces-J 2.x	Xalan-J XSLT	None
JAXP Reference Implementation	Xerces2 or Crimson	Xalan-J XSLT	See JAXP RI questions below.
Java 2 Platform, Standard Edition 1.4	Crimson	Xalan-J XSLT, cvs tag: xalan_2_2_d10	Uses JAXP RI version later than 1.1.2
Saxon	Old fork of Ælfred2	Saxon XSLT	No DOM support

- JAXP tutorial
- Javadoc API documentation
- JAXP samples
- JAXP FAQ

The Document Object Model (DOM) APIs

Interfaces	Uses
Document Object Module (DOM)	• XML to XML • non-XML to XML • XML to non-XML

The DOM APIs is generally an easier API to use. It provides a relatively familiar tree structure of objects. You can use the DOM API to manipulate the hierarchy of application objects it encapsulates. The DOM API is ideal for interactive applications because the entire object model is present in memory, where it can be accessed and manipulated by the user. On the other hand, constructing the DOM requires reading the entire XML structure and holding the object tree in memory, so it is much more CPU and memory intensive.

- DOM tutorial
- DOM samples

Creating a Parser and Parsing a Document

You use the *javax.xml.parsers.DocumentBuilderFactory* class to get a *DocumentBuilder* instance, and use that to produce a *Document* that conforms to the DOM specification. The factory APIs give you the ability to plug in an XML implementation offered by another vendor without changing your source code. The implementation you get depends on the setting of the javax.xml.parsers.SAXParserFactory and javax.xml.parsers.DocumentBuilderFactory system properties. The default values (unless overridden at runtime) point to Sun's implementation.

JAXP parsing uses the following steps:

1. Create a DocumentBuilderFactory instance.
2. Set any parser flags or properties.
3. Create DocumentBuilder to create the parser.
4. Parse the document

```
DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
factory.setNamespaceAware(true);
DocumentBuilder builder = factory.newDocumentBuilder();
Document document = builder.parse("catalog.xml");
```

nodeName, nodeValue, and attributes of Node

The values of `nodeName`, `nodeValue`, and `attributes` vary according to the node type as follows:

Interface	nodeName	nodeValue	attributes
Attr	name of attribute	value of attribute	null
CDATASection	`"#cdata-section"`	content of the CDATA Section	null
Comment	`"#comment"`	content of the comment	null
Document	`"#document"`	null	null
DocumentFragment	`"#document-fragment"`	null	null
DocumentType	document type name	null	null
Element	tag name	null	NamedNodeMap
Entity	entity name	null	null
EntityReference	name of entity referenced	null	null
Notation	notation name	null	null
ProcessingInstruction	target	entire content excluding the target	null
Text	`"#text"`	content of the text node	null

Getting the Element Text Content