



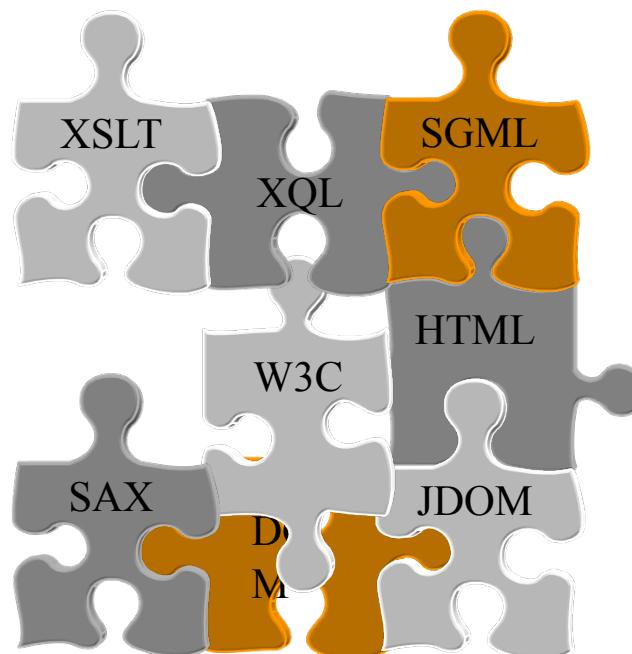
# **Session XML Lectures: 12 & 13**

# Introduction to XML

- **XML overview**
- **XML components**
- **Document Type Definition**
- **Specifying data elements (tags)**
- **Defining attributes and entities**
- **A look at XML schema**

# XML Overview

- When people refer to XML, they typically are referring to XML and related technologies



- XSL: Extensible Stylesheet Language; XQL (XML Query Language)
- XSLT: XSL Transformations; JDOM: Java-based Document Object Model
- SAX: Simple API for XML
- SGML: Standard Generalized Markup Language

# **XML Resources**

- **XML 1.0 Specification**
  - <http://www.w3.org/TR/REC-xml>
- **WWW consortium's Home Page on XML**
  - <http://www.w3.org/XML/>
- **Sun Page on XML and Java**
  - <http://java.sun.com/xml/>
- **Apache XML Project**
  - <http://xml.apache.org/>
- **XML Resource Collection**
  - <http://xml.coverpages.org/>
- **O'Reilly XML Resource Center**
  - <http://www.xml.com/>

# XML Overview

- **EXtensible Markup Language (XML) is a meta-language that describes the content of the document (self-describing data)**

Java = Portable Programs

XML = Portable Data

- **XML does not specify the tag set or grammar of the language**
  - Tag Set – markup tags that have meaning to a language processor
  - Grammar – defines correct usage of a language's tag

# Applications of XML

- **Configuration files**
  - Used extensively in J2EE architectures
- **Media for data interchange**
  - A better alternative to proprietary data formats
- **B2B transactions on the Web**
  - Electronic business orders (ebXML)
  - Financial Exchange (IFX)
  - Messaging exchange (SOAP)

# XML versus HTML

- XML fundamentally separates content (data and language) from presentation; HTML specifies the presentation
- HTML explicitly defines a set of legal tags as well as the grammar (intended meaning)

<TABLE> ... </TABLE>

- XML allows any tags or grammar to be used (hence, eXtensible)

<BOOK> ... </BOOK>

- Note: Both are based on Standard Generalized Markup Language (SGML)

# Simple XML Example

```
<?xml version="1.0"?>
<authors>
  <name>
    <firstname>Larry</firstname>
    <lastname>Brown</lastname>
  </name>
  <name>
    <firstname>Marty</firstname>
    <lastname>Hall</lastname>
  </name>
  ...
</authors>
```

# XML Components

- **Prolog**
  - Defines the xml version, entity definitions, and DOCTYPE
- **Components of the document**
  - Tags and attributes
  - CDATA (character data)
  - Entities
  - Processing instructions
  - Comments

# XML Prolog

- **XML Files always start with a prolog**

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
```

- The `version` of XML is required
- The `encoding` identifies character set (default UTF-8)
- The value `standalone` identifies if an *external* document is referenced for DTD or entity definition
- Note: the prolog can contain entities and DTD definitions

# Prolog Example

```
<?xml version="1.0" standalone="yes"?>
<!DOCTYPE authors [
<!ELEMENT authors (name*)
<!ELEMENT name (firstname, lastname)
<!ELEMENT firstname (#PCDATA)
<!ELEMENT lastname (#PCDATA)
]>
<authors>
  <name>
    <firstname>James</firstname>
    <lastname>Gosling</lastname>
  </name>
  ...
</authors>
```

# XML DOCTYPE

- **Document Type Definition**

- Specifies the location of the DTD defining the syntax and structure of elements in the document
- Common forms:

```
<!DOCTYPE root [DTD]>
<!DOCTYPE root SYSTEM URL>
<!DOCTYPE root PUBLIC FPI-identifier URL>
```

- The **root** identifies the starting element (root element) of the document
- The DTD can be external to the XML document, referenced by a **SYSTEM** or **PUBLIC** URL
  - **SYSTEM** URL refers to a private DTD
    - Located on the local file system or HTTP server
  - **PUBLIC** URL refers to a DTD intended for public use

# DOCTYPE Examples

```
<!DOCTYPE book "DTDs/CWP.dtd">
```

Book must be the root element  
of the XML document

DTD located in subdirectory  
below XML document

```
<!DOCTYPE book SYSTEM  
"http://www.corewebprogramming.com/DTDs/CWP.dtd">
```

DTD located HTTP server:  
[www.corewebprogramming.com](http://www.corewebprogramming.com)

# XML DOCTYPE, cont.

- **Specifying a PUBLIC DTD**

```
<!DOCTYPE root PUBLIC FPI-identifier URL>
```

- The Formal Public Identifier (FPI) has four parts:
  1. Connection of DTD to a formal standard
    - if defining yourself
    - + nonstandards body has approved the DTD
    - ISO if approved by formal standards committee
  2. Group responsible for the DTD
  3. Description and type of document
  4. Language used in the DTD

# PUBLIC DOCTYPE Examples

```
<!DOCTYPE Book  
PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"  
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">  
  
<!DOCTYPE CWP  
PUBLIC "-//Prentice Hall//DTD Core Series 1.0//EN"  
"http://www.prenticehall.com/DTD/Core.dtd">
```

# XML Comments

- **Comments are the same as HTML comments**

```
<!-- This is an XML and HTML comment -->
```

# Processing Instructions

- Application-specific instruction to the XML processor

```
<?processor-instruction?>
```

- Example

```
<?xml version="1.0" ?>
<?xmlstylesheet type="text/xml" href="orders.xsl" ?>
<orders>
  <order>
    <total-price>1849.63</total-price>
    <book>
      <count>37</count>
      <price>49.99</price>
      <isbn>0130897930</isbn>
      <title>Core Web Programming Second Edition</title>
      <authors>
        <author>Marty Hall</author>
        <author>Larry Brown</author>
      </authors>
    </book>
  </order>
</orders>
```

# XML Root Element

- Required for XML-aware applications to recognize beginning and end of document
- Example

```
<?xml version="1.0" ?>
<book>
    <title>Core Web Programming</title>
    <contents>
        <chapter number="1">
            Designing Web Pages with HTML
        </chapter>
        <chapter number="2">
            Block-level Elements in HTML 4.0
        </chapter>
        <chapter number="3">
            Text-level Elements in HTML 4.0
        </chapter>
        ...
    </contents>
</book>
```

# XML Tags

- **Tag names:**
  - Case sensitive
  - Start with a letter or underscore
  - After first character, numbers, – and . are allowed
  - Cannot contain whitespaces
  - Avoid use of colon expect for indicating namespaces
- **For a well-formed XML documents**
  - Every tag must have an end tag

```
<elementOne> ... </elementOne>
<elementTwo />
```
  - All tags are completely nested (tag order cannot be mixed)

# XML Tags, cont.

- Tags can also have attributes

```
<message to="Gates@microsoft.com" from="Gosling@sun.com">
  <priority/>
  <text>We put the . in .com.
    What did you do?
  </text>
</message>
```

# XML Attributes

- **Element Attributes**

- Attributes provide metadata for the element
- Every attribute must be enclosed in "" with no commas in between
- Same naming conventions as elements

# Using Tag Attributes

- **The general rule is:**
  - Use elements for *presentable* data and attributes for *system* data
    - Case I (preferred design)

```
<chapter number="23" focus="Server-side programming">  
    XML Processing with Java  
</chapter>
```
    - Case II
      - ```
<chapter>  
    <number>23</number>  
    <focus>Server-side programming</focus>  
    <title>XML Processing with Java</title>  
</chapter>
```
  - Note, however, not all XML technologies handle attributes well
    - A document with no attributes diminishes clarity and meaning, but is easier to process

# Document Entities

- **Entities refer to a data item, typically text**
  - General entity references start with & and end with ;
  - The entity reference is replaced by its true value when parsed
  - The characters < > & ' " require entity references to avoid conflicts with the XML application (parser)  
    &lt;   &gt;   &amp;   &quot;   &apos;
- **Entities are user definable**

```
<?xml version="1.0" standalone="yes" ?>
<!DOCTYPE book [
  <!ELEMENT book (title)>
  <!ELEMENT title (#PCDATA)>
  <!ENTITY COPYRIGHT "2001, Prentice Hall">
]>
<book>
  <title>Core Web Programming, &COPYRIGHT;</title>
</book>
```

# Document Entities (Aside)

- CDATA (character data) is not parsed

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<server>
  <port status="accept">
    <! [CDATA[8001 <= port < 9000]]>
  </port>
</server>
```

# Well-Formed versus Valid

- An XML document can be *well-formed* if it follows basic syntax rules
- An XML document is *valid* if its structure matches a Document Type Definition (DTD)

# Document Type Definition (DTD)

- **Defines Structure of the Document**
  - Allowable tags and their attributes
  - Attribute values constraints
  - Nesting of tags
  - Number of occurrences for tags
  - Entity definitions

# DTD Example

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!ELEMENT perennials (daylily)*>
<!ELEMENT daylily (cultivar, award*, bloom, cost)+>
<!ATTLIST daylily
    status (in-stock | limited | sold-out) #REQUIRED>
<!ELEMENT cultivar (#PCDATA)>
<!ELEMENT award (name, year)>
<!ELEMENT name (#PCDATA)>
<!ATTLIST name note CDATA #IMPLIED>
<!ELEMENT year (#PCDATA)>
<!ELEMENT bloom (#PCDATA)>
<!ATTLIST bloom code (E | EM | M | ML | L | E-L) #REQUIRED>
<!ELEMENT cost (#PCDATA)>
<!ATTLIST cost discount CDATA #IMPLIED>
<!ATTLIST cost currency (US | UK | CAN) "US">
```

# Defining Elements

- **<!ELEMENT name definition/type>**

```
<!ELEMENT daylily (cultivar, award*, bloom, cost)+>
<!ELEMENT cultivar (#PCDATA)>
<!ELEMENT id (#PCDATA | catalog_id)>
```

- **Types**

|            |                                                                                                           |
|------------|-----------------------------------------------------------------------------------------------------------|
| – ID       | Identifier                                                                                                |
| – IDREF    |                                                                                                           |
| – IDREFS   | ID reference                                                                                              |
| – ANY      | Any well-formed XML data                                                                                  |
| – EMPTY    | Element cannot contain any text or child elements                                                         |
| – PCDATA   | Character data only (should not contain markup)                                                           |
| – elements | List of legal child elements (no character data)                                                          |
| – mixed    | May contain character data and/or child elements<br>(cannot constrain order and number of child elements) |

# Defining Elements, cont.

- **Cardinality**
  - [none] Default (one and only one instance)
  - ? 0, 1
  - \* 0, 1, ..., N
  - + 1, 2, ..., N
- **List Operators**
  - , Sequence (in order)
  - | Choice (one of several)

# Grouping Elements

- Set of elements can be grouped within parentheses
  - $(\text{Elem1?}, \text{Elem2?})^+$ 
    - Elem1 can occur 0 or 1 times followed by 0 or 1 occurrences of Elem2
    - The group (sequence) must occur 1 or more times
- OR
  - $((\text{Elem1}, \text{Elem2}) \mid \text{Elem3})^*$ 
    - Either the group of Elem1, Elem2 is present (in order) or Elem3 is present, 0 or more times

# Element Example

```
<?xml version="1.0" standalone="yes"?>
<!DOCTYPE Person [
  <!ELEMENT Person ( (Mr|Ms|Miss)?, FirstName,
                     MiddleName*, LastName, (Jr|Sr)? )>
  <!ELEMENT FirstName (#PCDATA)>
  <!ELEMENT MiddleName (#PCDATA)>
  <!ELEMENT LastName (#PCDATA)>
  <!ELEMENT Mr EMPTY>
  <!ELEMENT Ms EMPTY>
  ...
  <!ELEMENT Sr EMPTY>
]>
<Person>
  <Mr/>
  <FirstName>Lawrence</FirstName>
  <LastName>Brown</LastName>
</Person>
```

# Defining Attributes

- **<!ATTLIST element attrName type modifier>**
- **Examples**

```
<!ELEMENT Customer (#PCDATA )>
<!ATTLIST Customer id CDATA #IMPLIED>
```

```
<!ELEMENT Product (#PCDATA )>
<!ATTLIST Product
    cost CDATA #FIXED "200"
    id    CDATA #REQUIRED>
```

# Attribute Types

- **CDATA**
  - Essentially anything; simply unparsed data

```
<!ATTLIST Customer id CDATA #IMPLIED>
```
- **Enumeration**
  - attribute (value1|value2|value3) [Modifier]
- **Eight other attribute types**
  - ID, IDREF, IDREFS, NMTOKEN, NMTOKENS, ENTITY, ENTITIES, NOTATION

# Attribute Modifiers

- **#IMPLIED**
  - Attribute is not required

```
<!ATTLIST cost discount CDATA #IMPLIED>
```
- **#REQUIRED**
  - Attribute must be present

```
<!ATTLIST account balance CDATA #REQUIRED>
```
- **#FIXED "value"**
  - Attribute is present and always has this value

```
<!ATTLIST interpreter language CDATA #FIXED "EN">
```
- **Default value (applies to enumeration)**
  - ```
<!ATTLIST car color (red | white | blue) "white" >
```

# Defining Entities

- `<!ENTITY name "replacement">`  
`<!ENTITY & "amp;">`  
`<!ENTITY copyright "Copyright 2001">`

# Limitations of DTDs

- DTD itself is not in XML format – more work for parsers
- Does not express data types (weak data typing)
- No namespace support
- Document can override external DTD definitions
- No DOM support
- XML Schema is intended to resolve these issues but ... DTDs are going to be around for a while

# XML Schema

- **W3C recommendation released May 2001**
  - <http://www.w3.org/TR/xmlschema-0/>
  - <http://www.w3.org/TR/xmlschema-1/>
  - <http://www.w3.org/TR/xmlschema-2/>
  - Depends on following specifications
    - XML-Infoset, XML-Namespace, XPath
- **Benefits:**
  - Standard and user-defined data types
  - Express data types as patterns
  - Higher degree of type checking
  - Better control of occurrences
  - Clearly the future ... but limited support

# XML Schema, Example

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="perennials" type="PerennialType"/>

  <xsd:complexType name="PerennialType" >
    <xsd:element name=daylily" type="DaylilyType"
                  maxOccurs="unbounded"/>
  </xsd:complexType>

  <xsd:complexType name="DaylilyType" >
    <xsd:sequence>
      <xsd:element name="cultivar" type="xsd:string"/>
      <xsd:element name="award"       type="AwardType"
                    maxOccurs="unbounded"/>
      <xsd:element name="bloom"      type="xsd:string"/>
      <xsd:element name="cost"       type="xsd:decimal"/>
    </xsd:sequence>
    <xsd:attribute name="status" type="StatusType"
                   use="required"/>
  </xsd:complexType>
```

# XML Schema, Example, cont.

```
<xsd:simpleType name="StatusType">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="in-stock"/>
    <xsd:enumeration value="limited"/>
    <xsd:enumeration value="sold-out"/>
  </xsd:restriction>
</xsd:simpleType>
...
</xsd:schema>
```

# Summary

- **XML is a self-describing meta data**
- **DOCTYPE defines the *root* element and location of DTD**
- **Document Type Definition (DTD) defines the grammar of the document**
  - Required to *validate* the document
  - Constrains grouping and cardinality of elements
- **DTD processing is expensive**
- **Schema uses XML to specify the grammar**
  - More complex to express but easier to process

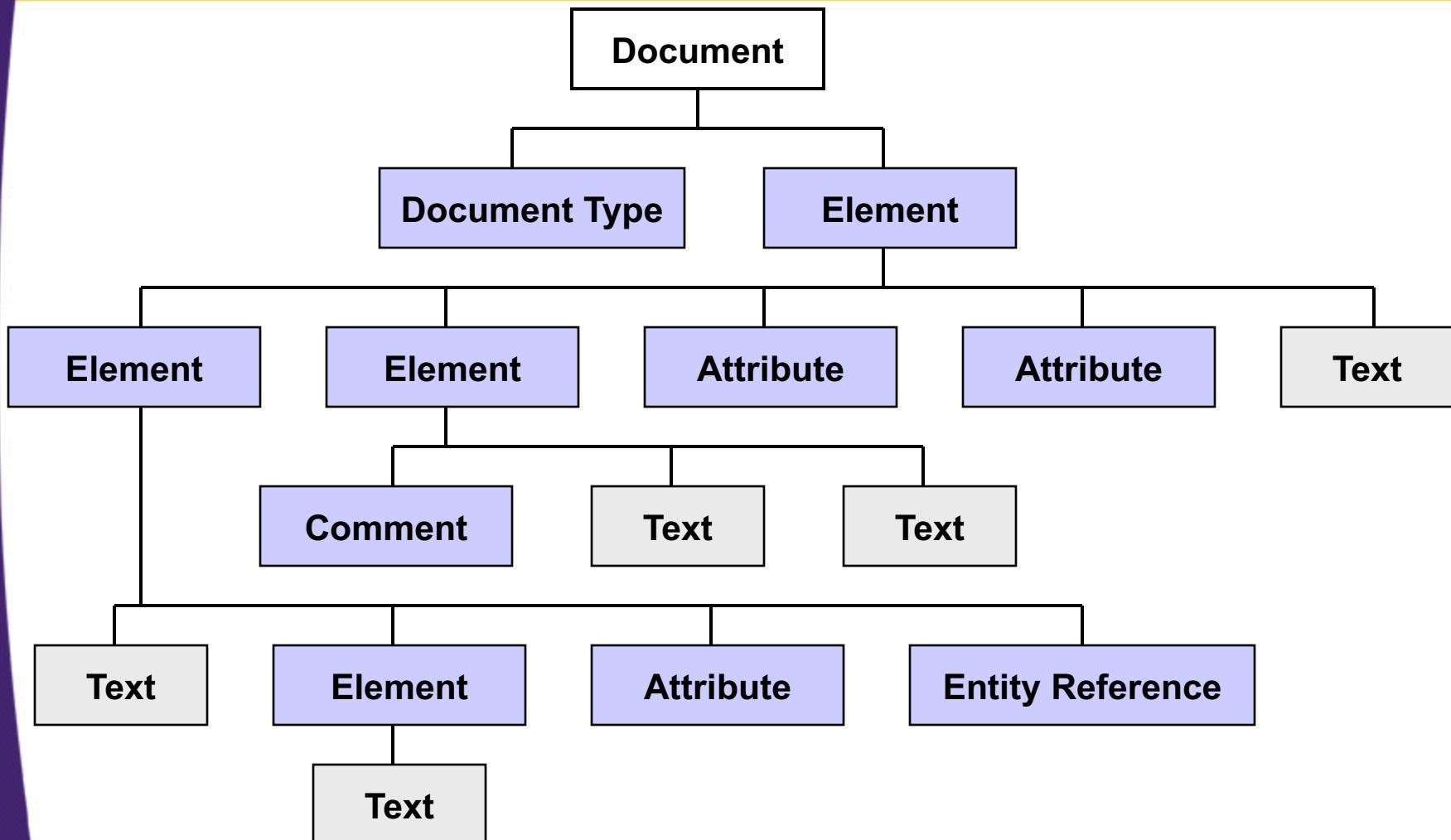
# Document Object Model

- **Introduction to DOM**
- **Java API for XML Parsing (JAXP)**
- **Installation and setup**
- **Steps for DOM parsing**
- **Example**
  - Representing an XML Document as a JTree
- **DOM or SAX?**

# Document Object Model (DOM)

- **DOM supports navigating and modifying XML documents**
  - Hierarchical tree representation of document
    - Tree follows standard API
    - Creating tree is vendor specific
- **DOM is a language-neutral specification**
  - Bindings exists for Java, C++, CORBA, JavaScript
- **DOM Versions**
  - DOM 1.0 (1998)
  - DOM 2.0 Core Specification (2000)
  - Official Website for DOM
    - <http://www.w3c.org/DOM/>

# DOM Tree



# Exercise

- **Given some API for DOM**

Node is object for all elements in XML file. Node has some methods as follows:

- getChildNodes(): return NodeList
- getNodeValue(): return value of Node
- getNodeType(): return ELEMENT\_NODE,  
DOCUMENT\_NODE, TEXT\_NODE
- getNodeName(): return name of current node

# Solution

- ```
void InOrder(Node root) {
    if (root.getNodeType() != Node.TEXT_NODE) {
        NodeList list = root.getChildNodes();
        if (list.getLength() > 0)
            InOrder(list.item(0));
        System.out.print(root.getNodeName());
        for(int i = 1; i < list.getLength(); i++)
            InOrder(list.item(i));
    }
    else
        System.out.print(root.getNodeValue());
}
```

# DOM Advantages and Disadvantages

- **Advantages**
  - Robust API for the DOM tree
  - Relatively simple to modify the data structure and extract data
- **Disadvantages**
  - Stores the entire document in memory
  - As DOM was written for any language, method naming conventions don't follow standard Java programming conventions

# Java API for XML Parsing (JAXP)

- JAXP provides a vendor-neutral interface to the underlying DOM or SAX parser

`javax.xml.parsers`

DocumentBuilderFactory  
DocumentBuilder

SAXParserFactory  
SAXParser

ParserConfigurationException  
FactoryConfigurationError

# **DOM Installation and Setup**

## **1. Download a DOM-compliant parser**

- Java-based DOM parsers at  
[http://www.xml.com/pub/rg/Java\\_Parsers](http://www.xml.com/pub/rg/Java_Parsers)
- Recommend Apache Xerces-J parser at  
<http://xml.apache.org/xerces-j/>

## **2. Download the Java API for XML Processing (JAXP)**

- JAXP is a small layer on top of DOM which supports specifying parsers through system properties versus hard coded
- See <http://java.sun.com/xml/>
- Note: Apache Xerces-J already incorporates JAXP

# DOM Installation and Setup (continued)

## 3. Set your CLASSPATH to include the DOM (and JAXP) classes

```
set CLASSPATH=xerces_install_dir\xerces.jar;  
          %CLASSPATH%
```

or

```
setenv CLASSPATH xerces_install_dir/xerces.jar:  
                      $CLASSPATH
```

- For servlets, place xerces.jar in the server's lib directory
  - Note: Tomcat 4.0 is prebundled with xerces.jar
  - Xerces-J already incorporates JAXP
    - For other parsers you may need to add jaxp.jar to your classpath and servlet lib directory

# Aside: Using Xerces with Tomcat 3.2.x

- **Problem**
  - Tomcat 3.2.x may load the provided DOM Level 1 parser first (`parser.jar`) before `xerces.jar`, effectively eliminating namespace support required in DOM Level 2
- **Solutions**
  1. Set up a static CLASSPATH and place `xerces.jar` *first* in the list
  2. As the files are loaded alphabetically, rename `parser.jar` to `z_parser.jar` and `xml.jar` to `z_xml.jar`

# DOM Installation and Setup (continued)

## 4. Bookmark the DOM Level 2 and JAXP APIs

- DOM Level 2
  - <http://www.w3.org/TR/DOM-Level-2-Core/>
- JAXP
  - <http://java.sun.com/xml/jaxp/dist/1.1/docs/api/index.html>

# Steps for DOM Parsing

- 1. Tell the system which parser you want to use**
- 2. Create a JAXP document builder**
- 3. Invoke the parser to create a Document representing an XML document**
- 4. Normalize the tree**
- 5. Obtain the root node of the tree**
- 6. Examine and modify properties of the node**

# Step 1: Specifying a Parser

- **Approaches to specify a parser**
  - Set a system property for `javax.xml.parsers.DocumentBuilderFactory`
  - Specify the parser in `jre_dir/lib/jaxp.properties`
  - Through the J2EE Services API and the class specified in `META-INF/services/javax.xml.parsers.DocumentBuilderFactory`
  - Use system-dependant default parser (check documentation)

# Specifying a Parser, Example

- **The following example:**
  - Permits the user to specify the parser through the command line -D option

```
java -Djavax.xml.parser.DocumentBuilderFactory =
      com.sun.xml.parser.DocumentBuilderFactoryImpl ...
```
  - Uses the Apache Xerces parser otherwise

```
public static void main(String[] args) {
    String jaxpPropertyName =
        "javax.xml.parsers.DocumentBuilderFactory";
    if (System.getProperty(jaxpPropertyName) == null) {
        String apacheXercesPropertyValue =
            "org.apache.xerces.jaxp.DocumentBuilderFactoryImpl";
        System.setProperty(jaxpPropertyName,
                           apacheXercesPropertyValue);
    }
    ...
}
```

# Step 2: Create a JAXP Document Builder

- First create an instance of a builder factory, then use that to create a `DocumentBuilder` object

```
DocumentBuilderFactory builderFactory =  
    DocumentBuilderFactory.newInstance();  
DocumentBuilder builder =  
    builderFactory.newDocumentBuilder();
```

- A builder is basically a wrapper around a specific XML parser
- To set up namespace awareness and validation, use

```
builderFactory.setNamespaceAware(true)  
builderFactory.setValidating(true)
```

# Step3: Invoke the Parser to Create a Document

- Call the `parse` method of the `DocumentBuilder`, supplying an XML document (input stream)

```
Document document = builder.parse(someInputStream) ;
```

- The `Document` class represents the parsed result in a tree structure
- The XML document can be represented as a:
  - URI, represented as a string
  - InputStream
  - org.xml.sax.InputSource

# Step 4: Normalize the Tree

- **Normalization has two affects:**
  - Combines textual nodes that span multiple lines
  - Eliminates empty textual nodes

```
document.getDocumentElement().normalize();
```

# Step 5: Obtain the Root Node of the Tree

- Traversing and modifying the tree begins at the root node

```
Element rootElement = document.getDocumentElement();
```

- An Element is a subclass of the more general Node class and represents an XML element
- A Node represents all the various components of an XML document
  - Document, Element, Attribute, Entity, Text, CDATA, Processing Instruction, Comment, etc.

# Step 6: Examine and Modify Properties of the Node

- **Examine the various node properties**
  - `getnodeName`
    - Returns the name of the element
  - `getNodeType`
    - Returns the node type
    - Compare to `Node` constants
      - `DOCUMENT_NODE`, `ELEMENT_NODE`, etc.
  - `getAttributes`
    - Returns a `NamedNodeMap` (collection of nodes, each representing an attribute)
      - Obtain particular attribute node through `getNamedItem`
  - `getChildNodes`
    - Returns a `NodeList` collection of all the children

# Step 6: Examine and Modify Properties of the Node (cont)

- **Modify the document**
  - setNodeValue
    - Assigns the text value of the node
  - appendChild
    - Adds a new node to the list of children
  - removeChild
    - Removes the child node from the list of children
  - replaceChild
    - Replace a child with a new node

# DOM Example: Representing an XML Document as a JTree

- **Approach**
  - Each XML document element is represented as a tree node (in the JTree)
  - Each tree node is either the element name or the element name followed by a list of attributes

# DOM Example: Representing an XML Document as a JTree

- **Approach (cont.)**
  - The following steps are performed:
    1. Parse and normalize the XML document and then obtain the root node
    2. Turn the root note into a JTree node
      - The element name (`getnodeName`) is used for the tree node label
      - If attributes are present (`node.getAttributes`), then include them in the label enclosed in parentheses
    3. Look up child elements (`getChildNodes`) and turn them into JTree nodes, linking to their parent tree node
    4. Recursively apply step 3 to all child nodes

# DOM Example: XMLTree

```
import java.awt.*;
import javax.swing.*;
import javax.swing.tree.*;
import java.io.*;
import org.w3c.dom.*;
import javax.xml.parsers.*;

/** Given a filename or a name and an input stream,
 *  this class generates a JTree representing the
 *  XML structure contained in the file or stream.
 *  Parses with DOM then copies the tree structure
 *  (minus text and comment nodes).
 */

public class XMLTree extends JTree {
    public XMLTree(String filename) throws IOException {
        this(filename, new FileInputStream(new File(filename)));
    }

    public XMLTree(String filename, InputStream in) {
        super(makeRootNode(in));
    }
}
```

# DOM Example: XMLTree (continued)

```
private static DefaultMutableTreeNode
    makeRootNode(InputStream in) {
    try {
        // Use the system property
        // javax.xml.parsers.DocumentBuilderFactory (set either
        // from Java code or by using the -D option to "java").
        DocumentBuilderFactory builderFactory =
            DocumentBuilderFactory.newInstance();
        DocumentBuilder builder =
            builderFactory.newDocumentBuilder();
        Document document = builder.parse(in);
        document.getDocumentElement().normalize();
        Element rootElement = document.getDocumentElement();
        DefaultMutableTreeNode rootTreeNode =
            buildTree(rootElement);
        return(rootTreeNode);
    } catch(Exception e) {
        String errorMessage = "Error making root node: " + e;
        System.err.println(errorMessage);
        e.printStackTrace();
        return(new DefaultMutableTreeNode(errorMessage));
    }
}
```

# DOM Example: XMLTree (continued)

```
...
private static DefaultMutableTreeNode
        buildTree(Element rootElement) {

    // Make a JTree node for the root, then make JTree
    // nodes for each child and add them to the root node.
    // The addChildren method is recursive.

    DefaultMutableTreeNode rootTreeNode =
        new DefaultMutableTreeNode(treeNodeLabel(rootElement));
    addChildren(rootTreeNode, rootElement);
    return (rootTreeNode);
}

...
```

# DOM Example: XMLTree (continued)

```
private static void addChildren
    (DefaultMutableTreeNode parentTreeNode, Node parentXMLElement) {
    // Recursive method that finds all the child elements and adds
    // them to the parent node. Nodes corresponding to the graphical
    // JTree will have the word "tree" in the variable name.
    NodeList childElements =
        parentXMLElement.getChildNodes();
    for(int i=0; i<childElements.getLength(); i++) {
        Node childElement = childElements.item(i);
        if (!(childElement instanceof Text ||
              childElement instanceof Comment)) {
            DefaultMutableTreeNode childTreeNode =
                new DefaultMutableTreeNode
                    (treeNodeLabel(childElement));
            parentTreeNode.add(childTreeNode);
            addChildren(childTreeNode, childElement);
        }
    }
}
```

# DOM Example: XMLTree (continued)

```
...
private static String treeNodeLabel(Node childElement) {
    NamedNodeMap elementAttributes =
        childElement.getAttributes();
    String treeNodeLabel = childElement.getNodeName();
    if (elementAttributes != null &&
        elementAttributes.getLength() > 0) {
        treeNodeLabel = treeNodeLabel + " (";
        int numAttributes = elementAttributes.getLength();
        for(int i=0; i<numAttributes; i++) {
            Node attribute = elementAttributes.item(i);
            if (i > 0) {
                treeNodeLabel = treeNodeLabel + ", ";
            }
            treeNodeLabel =
                treeNodeLabel + attribute.getNodeName() +
                "=" + attribute.getNodeValue();
        }
        treeNodeLabel = treeNodeLabel + ")";
    }
    return(treeNodeLabel);
}
```

# DOM Example: XMLFrame

```
import java.awt.*;
import javax.swing.*;
import java.io.*;

public class XMLFrame extends JFrame {
    public static void main(String[] args) {
        String jaxpPropertyName =
            "javax.xml.parsers.DocumentBuilderFactory";
        // Pass the parser factory in on the command line with
        // -D to override the use of the Apache parser.
        if (System.getProperty(jaxpPropertyName) == null) {
            String apacheXercesPropertyValue =
                "org.apache.xerces.jaxp.DocumentBuilderFactoryImpl";
            System.setProperty(jaxpPropertyName,
                               apacheXercesPropertyValue);
        }
        ...
    }
}
```

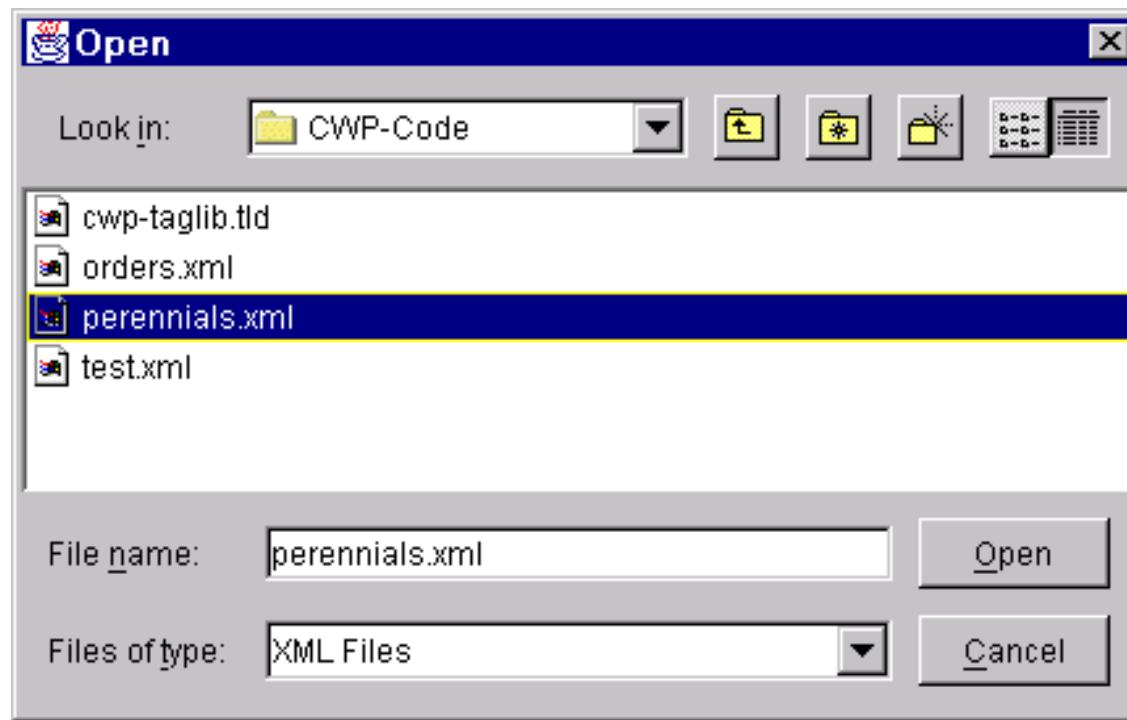
# DOM Example: XMLFrame (continued)

```
String[] extensions = { "xml", "tld" };  
WindowUtilities.setNativeLookAndFeel();  
String filename = ExtensionFileFilter.getFileName(".",  
        "XML Files", extensions);  
new XMLFrame(filename);  
}  
  
public XMLFrame(String filename) {  
    try {  
        WindowUtilities.setNativeLookAndFeel();  
        JTree tree = new XMLTree(filename);  
        JFrame frame = new JFrame(filename);  
        frame.addWindowListener(new ExitListener());  
        Container content = frame.getContentPane();  
        content.add(new JScrollPane(tree));  
        frame.pack();  
        frame.setVisible(true);  
    } catch(IOException ioe) {  
        System.out.println("Error creating tree: " + ioe);  
    }  
}
```

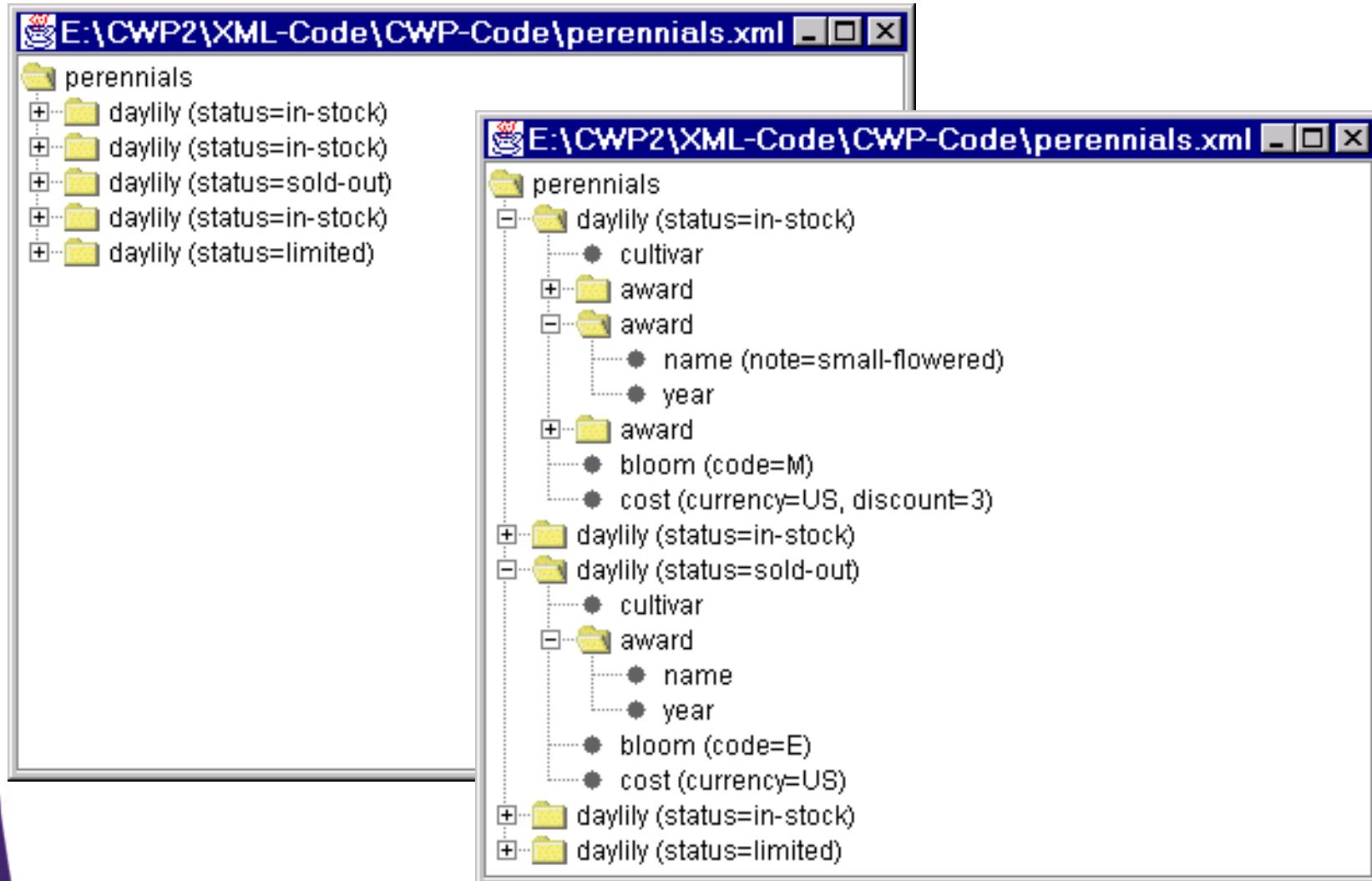
# DOM Example: perennials.xml

```
<?xml version="1.0"?>
<!DOCTYPE perennials SYSTEM "dtds/perennials.dtd">
<perennials>
  <daylily status="in-stock">
    <cultivar>Luxury Lace</cultivar>
    <award>
      <name>Stout Medal</name>
      <year>1965</year>
    </award>
    <award>
      <name note="small-flowered">Annie T. Giles</name>
      <year>1965</year>
    </award>
    <award>
      <name>Lenington All-American</name>
      <year>1970</year>
    </award>
    <bloom code="M">Midseason</bloom>
    <cost discount="3" currency="US">11.75</cost>
  </daylily>
  ...
<perennials>
```

# DOM Example: Results



# DOM Example: Results (continued)



# **DOM or SAX?**

- **DOM**
  - Suitable for small documents
  - Easily modify document
  - Memory intensive; load the complete XML document
- **SAX**
  - Suitable for large documents; saves significant amounts of memory
  - Only traverse document once, start to end
  - Event driven
  - Limited standard functions

# Summary

- **DOM is a tree representation of an XML document in memory**
  - DOM provides a robust API to easily modify and extract data from an XML document
- **JAXP provides a vendor-neutral interface to the underlying DOM or SAX parser**
- **Every component of the XML document is represent as a Node**
- **Use normalization to combine text elements spanning multiple lines**

# Simple API for XML

- **Introduction to SAX**
- **Installation and setup**
- **Steps for SAX parsing**
- **Defining a content handler**
- **Examples**
  - Printing the Outline of an XML Document
  - Counting Book Orders
- **Defining an error handler**
- **Validating a document**

# Simple API for XML (SAX)

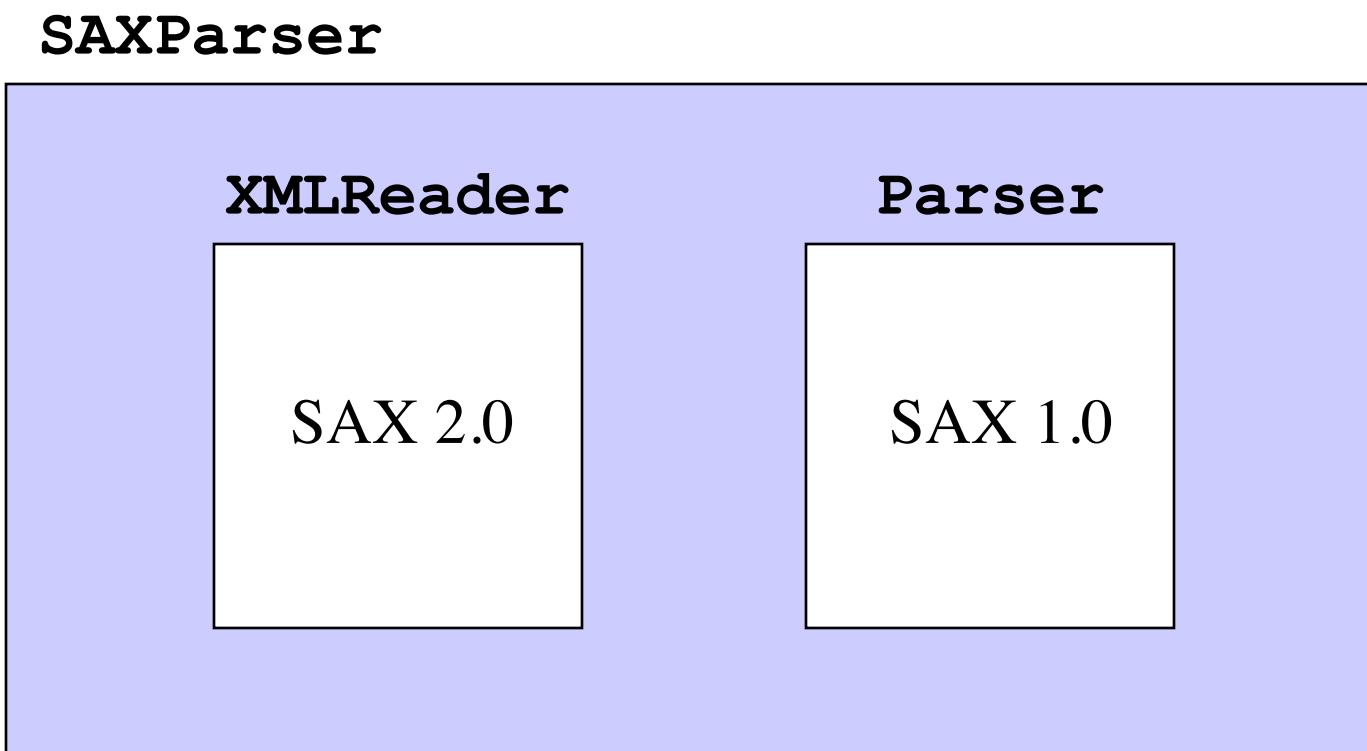
- Parse and process XML documents
- Documents are read sequentially and callbacks are made to handlers
- Event-driven model for processing XML content
- SAX Versions
  - SAX 1.0 (May 1998)
  - SAX 2.0 (May 2000)
    - Namespace addition
  - Official Website for SAX
    - <http://sax.sourceforge.net/>

# SAX Advantages and Disadvantages

- **Advantages**
  - Do not need to process and store the entire document (low memory requirement)
    - Can quickly skip over parts not of interest
  - Fast processing
- **Disadvantages**
  - Limited API
    - Every element is processed through the same event handler
    - Need to keep track of location in document and, in cases, store temporary data
  - Only traverse the document once

# Java API for XML Parsing (JAXP)

- JAXP provides a vendor-neutral interface to the underlying SAX 1.0/2.0 parser



# SAX Installation and Setup

## 1. Download a SAX 2-compliant parser

- Java-based XML parsers at  
[http://www.xml.com/pub/rg/Java\\_Parsers](http://www.xml.com/pub/rg/Java_Parsers)
- Recommend Apache Xerces-J parser at  
<http://xml.apache.org/xerces-j/>

## 2. Download the Java API for XML Processing (JAXP)

- JAXP is a small layer on top of SAX which supports specifying parsers through system properties versus hard coded
- See <http://java.sun.com/xml/>
- Note: Apache Xerces-J already incorporates JAXP

# SAX Installation and Setup (continued)

## 3. Set your CLASSPATH to include the SAX (and JAXP) classes

```
set CLASSPATH=xerces_install_dir\xerces.jar;  
          %CLASSPATH%
```

or

```
setenv CLASSPATH xerces_install_dir/xerces.jar:  
                      $CLASSPATH
```

- For servlets, place xerces.jar in the server's lib directory
  - Note: Tomcat 4.0 is prebundled with xerces.jar
  - Xerces-J already incorporates JAXP
    - For other parsers you may need to add jaxp.jar to your classpath and servlet lib directory

# Aside: Using Xerces with Tomcat 3.2.x

- **Problem**
  - Tomcat 3.2.x may load the provided SAX 1.0 parser first (`parser.jar`) before `xerces.jar`, effectively eliminating namespace support required in SAX 2.0
- **Solutions**
  1. Set up a static CLASSPATH and place `xerces.jar` *first* in the list
  2. As the files are loaded alphabetically, rename `parser.jar` to `z_parser.jar` and `xml.jar` to `z_xml.jar`

# SAX Parsing

- **SAX parsing has two high-level tasks:**
  1. Creating a content handler to process the XML elements when they are encountered
  2. Invoking a parser with the designated content handler and document

# Steps for SAX Parsing

- 1. Tell the system which parser you want to use**
- 2. Create a parser instance**
- 3. Create a content handler to respond to parsing events**
- 4. Invoke the parser with the designated content handler and document**

# Step 1: Specifying a Parser

- **Approaches to specify a parser**
  - Set a system property for `javax.xml.parsers.SAXParserFactory`
  - Specify the parser in `jre_dir/lib/jaxp.properties`
  - Through the J2EE Services API and the class specified in `META-INF/services/javax.xml.parsers.SAXParserFactory`
  - Use system-dependant default parser (check documentation)

# Specifying a Parser, Example

- **The following example:**
  - Permits the user to specify the parser through the command line -D option

```
java -Djavax.xml.parser.SAXParserFactory=
      com.sun.xml.parser.SAXParserFactoryImpl ...
```
  - Uses the Apache Xerces parser otherwise

```
public static void main(String[] args) {
    String jaxpPropertyName =
        "javax.xml.parsers.SAXParserFactory";
    if (System.getProperty(jaxpPropertyName) == null) {
        String apacheXercesPropertyValue =
            "org.apache.xerces.jaxp.SAXParserFactoryImpl";
        System.setProperty(jaxpPropertyName,
                           apacheXercesPropertyValue);
    }
    ...
}
```

# Step 2: Creating a Parser Instance

- First create an instance of a parser factory, then use that to create a SAXParser object

```
SAXParserFactory factory =  
    SAXParserFactory.newInstance();  
SAXParser parser = factory.newSAXParser();
```

- To set up namespace awareness and validation, use

```
factory.setNamespaceAware(true)  
factory.setValidating(true)
```

# Step 3: Create a Content Handler

- **Content handler responds to parsing events**

- Typically a subclass of DefaultHandler

```
public class MyHandler extends DefaultHandler {  
    // Callback methods  
    ...  
}
```

- **Primary event methods (callbacks)**

- startDocument, endDocument
    - Respond to the start and end of the document
  - startElement, endElement
    - Respond to the start and end tags of an element
  - characters, ignoreableWhitespace
    - Respond to the tag body

# ContentHandler

## startElement Method

- **Declaration**

```
public void startElement(String namespaceURI,  
                         String localName,  
                         String qualifiedName,  
                         Attributes attributes)  
throws SAXException
```

- **Arguments**

- namespaceUri
  - URI uniquely identifying the namespace
- localname
  - Element name without prefix
- qualifiedName
  - Complete element name, including prefix
- attributes
  - Attributes object representing the attributes of the element

# Anatomy of an Element

```
<cwp:book xmlns:cwp="http://www.corewebprogramming.com/xml/">  
    qualifiedName  
    <cwp:chapter number="23" part="Server-side Programming">  
        attribute[1]  
        <cwp:title>XML Processing with Java</cwp:title>  
        localname  
    </cwp:chapter>  
</cwp:book>
```

# ContentHandler characters Method

- **Declaration**

```
public void characters(char[] chars,  
                      int startIndex,  
                      int length)  
throws SAXException
```

- **Arguments**

- chars
  - Relevant characters form XML document
  - To optimize parsers, the chars array may represent more of the XML document than just the element
  - **PCDATA** may cause multiple invocations of characters
- startIndex
  - Starting position of element
- length
  - The number of characters to extract

# Step 4: Invoke the Parser

- Call the **parse** method, supplying:
  1. The content handler
  2. The XML document
    - File, input stream, or `org.xml.sax.InputSource`

```
parser.parse(filename, handler)
```

# SAX Example 1: Printing the Outline of an XML Document

- **Approach**
  - Define a content handler to respond to three parts of an XML document: start tags, end tag, and tag bodies
  - Content handler implementation overrides the following three methods:
    - `startElement`
      - Prints a message when start tag is found with attributes listed in parentheses
      - Adjusts (increases by 2 spaces) the indentation
    - `endElement`
      - Subtracts 2 from the indentation and prints a message indicating that an end tag was found
    - `characters`
      - Prints the first word of the tag body

# SAX Example 1: PrintHandler

```
import org.xml.sax.*;
import org.xml.sax.helpers.*;
import java.util.StringTokenizer;

public class PrintHandler extends DefaultHandler {
    private int indentation = 0;

    /** When you see a start tag, print it out and then
     * increase indentation by two spaces. If the
     * element has attributes, place them in parens
     * after the element name.
    */
    public void startElement(String namespaceUri,
                            String localName,
                            String qualifiedName,
                            Attributes attributes)
        throws SAXException {
        indent(indentation);
        System.out.print("Start tag: " + qualifiedName);
```

# SAX Example 1: PrintHandler (continued)

```
...
int numAttributes = attributes.getLength();
// For <someTag> just print out "someTag". But for
// <someTag att1="Val1" att2="Val2">, print out
// "someTag (att1=Val1, att2=Val2)".
if (numAttributes > 0) {
    System.out.print(" (");
    for(int i=0; i<numAttributes; i++) {
        if (i>0) {
            System.out.print(", ");
        }
        System.out.print(attributes.getQName(i) + "=" +
                           attributes.getValue(i));
    }
    System.out.print(")");
}
System.out.println();
indentation = indentation + 2;
}
...
```

# SAX Example 1: PrintHandler (continued)

```
/** When you see the end tag, print it out and decrease
 * indentation level by 2.
 */

public void endElement(String namespaceUri,
                      String localName,
                      String qualifiedName)
    throws SAXException {
    indentation = indentation - 2;
    indent(indentation);
    System.out.println("End tag: " + qualifiedName);
}

private void indent(int indentation) {
    for(int i=0; i<indentation; i++) {
        System.out.print(" ");
    }
}
...
```

# SAX Example 1: PrintHandler (continued)

```
/** Print out the first word of each tag body. */

public void characters(char[] chars,
                      int startIndex,
                      int length) {
    String data = new String(chars, startIndex, length);
    // Whitespace makes up default StringTokenizer delimiters
    StringTokenizer tok = new StringTokenizer(data);
    if (tok.hasMoreTokens()) {
        indent(indentation);
        System.out.print(tok.nextToken());
        if (tok.hasMoreTokens()) {
            System.out.println("...");
        } else {
            System.out.println();
        }
    }
}
```

# SAX Example 1: SAXPrinter

```
import javax.xml.parsers.*;
import org.xml.sax.*;
import org.xml.sax.helpers.*;

public class SAXPrinter {
    public static void main(String[] args) {
        String jaxpPropertyName =
            "javax.xml.parsers.SAXParserFactory";
        // Pass the parser factory in on the command line with
        // -D to override the use of the Apache parser.
        if (System.getProperty(jaxpPropertyName) == null) {
            String apacheXercesPropertyValue =
                "org.apache.xerces.jaxp.SAXParserFactoryImpl";
            System.setProperty(jaxpPropertyName,
                               apacheXercesPropertyValue);
        }
    }
}
```

# SAX Example 1: SAXPrinter (continued)

```
...
String filename;
if (args.length > 0) {
    filename = args[0];
} else {
    String[] extensions = { "xml", "tld" };
    WindowUtilities.setNativeLookAndFeel();
    filename =
        ExtensionFileFilter.getFileName(".", "XML Files",
                                         extensions);
    if (filename == null) {
        filename = "test.xml";
    }
}
printOutline(filename);
System.exit(0);
}
```

...

# SAX Example 1: SAXPrinter (continued)

```
...
public static void printOutline(String filename) {
    DefaultHandler handler = new PrintHandler();
    SAXParserFactory factory =
        SAXParserFactory.newInstance();
    try {
        SAXParser parser = factory.newSAXParser();
        parser.parse(filename, handler);
    } catch(Exception e) {
        String errorMessage =
            "Error parsing " + filename + ": " + e;
        System.err.println(errorMessage);
        e.printStackTrace();
    }
}
```

# SAX Example 1: orders.xml

```
<?xml version="1.0"?>
<orders>
  <order>
    <count>1</count>
    <price>9.95</price>
    <yacht>
      <manufacturer>Luxury Yachts, Inc.</manufacturer>
      <model>M-1</model>
      <standardFeatures oars="plastic"
                      lifeVests="none">
        false
      </standardFeatures>
    </yacht>
  </order>
  ...
</orders>
```

# SAX Example 1: Result

```
Start tag: orders
Start tag: order
  Start tag: count
    1
  End tag: count
  Start tag: price
    9.95
  End tag: price
  Start tag: yacht
    Start tag: manufacturer
      Luxury...
    End tag: manufacturer
    Start tag: model
      M-1
    End tag: model
    Start tag: standardFeatures (oars=plastic, lifeVests=none)
      false
    End tag: standardFeatures
  End tag: yacht
End tag: order
...
End tag: orders
```

# SAX Example 2: Counting Book Orders

- **Objective**
  - To process XML files that look like:

```
<orders>
  ...
  <count>23</count>
  <book>
    <isbn>013897930</isbn>
    ...
  </book>
  ...
</orders>
```

and count up how many copies of Core Web Programming (ISBN 013897930) are contained in the order

# SAX Example 2: Counting Book Orders (continued)

- **Problem**
  - SAX doesn't store data automatically
  - The `isbn` element comes after the `count` element
  - Need to record every `count` temporarily, but only add the temporary value (to the running total) when the ISBN number matches

# SAX Example 2: Approach

- Define a content handler to override the following four methods:
  - startElement
    - Checks whether the name of the element is either count or isbn
    - Set flag to tell characters method be on the lookout
  - endElement
    - Again, checks whether the name of the element is either count or isbn
    - If so, turns off the flag that the characters method watches

# SAX Example 2: Approach (continued)

- characters
  - Subtracts 2 from the indentation and prints a message indicating that an end tag was found
- endDocument
  - Prints out the running count in a Message Dialog

# SAX Example 2: CountHandler

```
import org.xml.sax.*;
import org.xml.sax.helpers.*;
...

public class CountHandler extends DefaultHandler {
    private boolean collectCount = false;
    private boolean collectISBN = false;
    private int currentCount = 0;
    private int totalCount = 0;

    public void startElement(String namespaceUri,
                            String localName,
                            String qualifiedName,
                            Attributes attributes)
        throws SAXException {
        if (qualifiedName.equals("count")) {
            collectCount = true;
            currentCount = 0;
        } else if (qualifiedName.equals("isbn")) {
            collectISBN = true;
        }
    }
}
```

# SAX Example 2: CountHandler (continued)

```
...
public void endElement(String namespaceUri,
                      String localName,
                      String qualifiedName)
    throws SAXException {
    if (qualifiedName.equals("count")) {
        collectCount = false;
    } else if (qualifiedName.equals("isbn")) {
        collectISBN = false;
    }
}

public void endDocument() throws SAXException {
    String message =
        "You ordered " + totalCount + " copies of \n" +
        "Core Web Programming Second Edition.\n";
    if (totalCount < 250) {
        message = message + "Please order more next time!";
    } else {
        message = message + "Thanks for your order.";
    }
    JOptionPane.showMessageDialog(null, message);
}
```

# SAX Example 2: CountHandler (continued)

```
...
public void characters(char[] chars, int startIndex,
                      int length) {
    if (collectCount || collectISBN) {
        String dataString =
            new String(chars, startIndex, length).trim();
        if (collectCount) {
            try {
                currentCount = Integer.parseInt(dataString);
            } catch (NumberFormatException nfe) {
                System.err.println("Ignoring malformed count: " +
                                   dataString);
            }
        } else if (collectISBN) {
            if (dataString.equals("0130897930")) {
                totalCount = totalCount + currentCount;
            }
        }
    }
}
```

# SAX Example 2: CountBooks

```
import javax.xml.parsers.*;
import org.xml.sax.*;
import org.xml.sax.helpers.*;

public class CountBooks {
    public static void main(String[] args) {
        String jaxpPropertyName = "javax.xml.parsers.SAXParserFactory";
        // Use -D to override the use of the Apache parser.
        if (System.getProperty(jaxpPropertyName) == null) {
            String apacheXercesPropertyValue =
                "org.apache.xerces.jaxp.SAXParserFactoryImpl";
            System.setProperty(jaxpPropertyName,
                               apacheXercesPropertyValue);
        }
        String filename;
        if (args.length > 0) {
            filename = args[0];
        } else {
            ...
        }
        countBooks(filename);
        System.exit(0);
    }
}
```

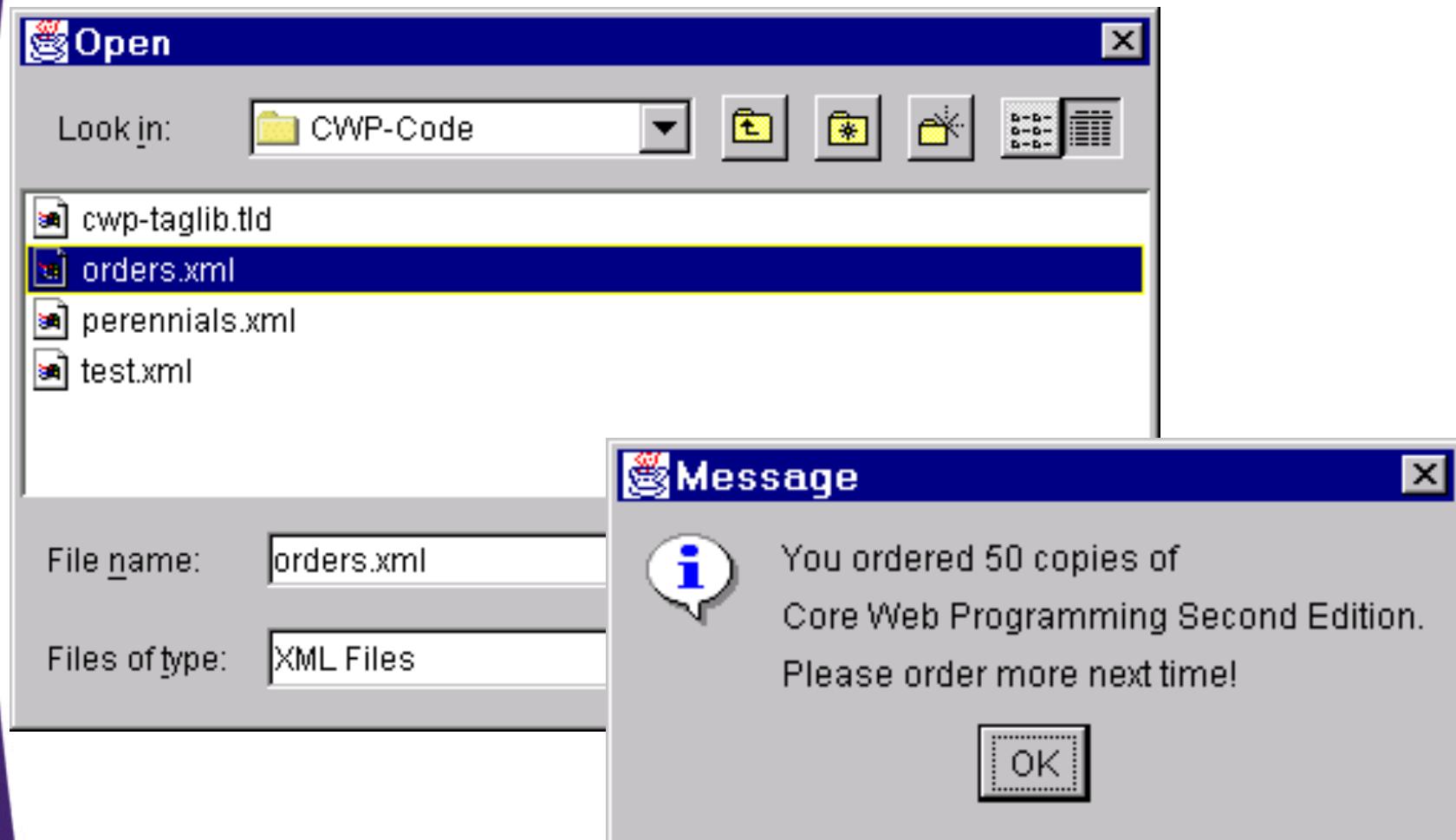
# SAX Example 2: CountBooks (continued)

```
private static void countBooks(String filename) {  
    DefaultHandler handler = new CountHandler();  
    SAXParserFactory factory =  
        SAXParserFactory.newInstance();  
    try {  
        SAXParser parser = factory.newSAXParser();  
        parser.parse(filename, handler);  
    } catch(Exception e) {  
        String errorMessage =  
            "Error parsing " + filename + ": " + e;  
        System.err.println(errorMessage);  
        e.printStackTrace();  
    }  
}
```

# SAX Example 2: orders.xml

```
<?xml version="1.0"?>
<orders>
  <order>
    <count>37</count>
    <price>49.99</price>
    <book>
      <isbn>0130897930</isbn>
      <title>Core Web Programming Second Edition</title>
      <authors>
        <author>Marty Hall</author>
        <author>Larry Brown</author>
      </authors>
    </book>
  </order>
  ...
</orders>
```

# SAX Example 2: Result



# Error Handlers

- Responds to parsing errors
  - Typically a subclass of `DefaultErrorHandler`
- Useful callback methods
  - `error`
    - Nonfatal error
    - Usual a result of document validity problems
  - `fatalError`
    - A fatal error resulting from a malformed document
  - Receive a `SAXParseException` from which to obtain the location of the problem (`getColumnNumber`, `getLineNumber`)

# Error Handler Example

```
import org.xml.sax.*;
import org.apache.xml.utils.*;

class MyErrorHandler extends DefaultErrorHandler  {

    public void error(SAXParseException exception)
        throws SAXException {
        System.out.println(
            "***Parsing Error**\n" +
            "  Line:      " + exception.getLineNumber() + "\n" +
            "  URI:       " + exception.getSystemId() + "\n" +
            "  Message:  " + exception.getMessage() + "\n");
        throw new SAXException("Error encountered");
    }
}
```

# Namespace Awareness and Validation

- **Approaches**

1. Through the SAXParserFactory

```
factory.setNamespaceAware(true)  
factory.setValidating(true)  
SAXParser parser = factory.newSAXParser();
```

2. By setting XMLReader features

```
XMLReader reader = parser.getXMLReader();  
reader.setFeature(  
    "http://xml.org/sax/features/validation", true);  
reader.setFeature(  
    "http://xml.org/sax/features/namespaces", false);
```

- Note: a SAXParser is a vendor-neutral wrapper around a SAX 2 XMLReader

# Validation Example

```
public class SAXValidator {  
    public static void main(String[] args) {  
        String jaxpPropertyName =  
            "javax.xml.parsers.SAXParserFactory";  
        // Use -D to override the use of the Apache parser.  
        if (System.getProperty(jaxpPropertyName) == null) {  
            String apacheXercesPropertyValue =  
                "org.apache.xerces.jaxp.SAXParserFactoryImpl";  
            System.setProperty(jaxpPropertyName,  
                apacheXercesPropertyValue);  
        }  
        String filename;  
        if (args.length > 0) {  
            filename = args[0];  
        } else {  
            ...  
        }  
        validate(filename);  
        System.exit(0);  
    }  
}
```

# Validation Example (continued)

```
...
public static void validate(String filename) {
    DefaultHandler contentHandler = new DefaultHandler();
    ErrorHandler errHandler = new MyErrorHandler();
    SAXParserFactory factory =
        SAXParserFactory.newInstance();
factory.setValidating(true);
try {
    SAXParser parser = factory.newSAXParser();
    XMLReader reader = parser.getXMLReader();
    reader.setContentHandler(contentHandler);
    reader.setErrorHandler(errHandler);
    reader.parse(new InputSource(filename));
} catch(Exception e) {
    String errorMessage =
        "Error parsing " + filename;
    System.out.println(errorMessage);
}
}
```

# Instructors.xml

```
<?xml version="1.0" standalone="yes"?>
<!DOCTYPE jhu [
<!ELEMENT jhu (instructor)*>
<!ELEMENT instructor (firstname, lastname)+>
<!ELEMENT firstname (#PCDATA)>
<!ELEMENT lastname (#PCDATA)>
]>
<jhu>
  <instructor>
    <firstname>Larry</firstname>
    <lastname>Brown</lastname>
  </instructor>
  <instructor>
    <lastname>Hall</lastname>
    <firstname>Marty</firstname>
  </instructor>
</jhu>
```

# Validation Results

```
>java SAXValidator
```

Parsing Error:

Line: 16

URI: file:///C:/CWP2-Book/chapter23/Instructors.xml

Message: The content of element type "instructor"  
must match "(firstname, lastname)+".

Error parsing C:\CWP2-Book\chapter23\Instructors.xml

# Summary

- **SAX processing of XML documents is fast and memory efficient**
- **JAXP is a simple API to provide vendor neutral SAX parsing**
  - Parser is specified through system properties
- **Processing is achieved through event callbacks**
  - Parser communicates with a DocumentHandler
  - May require tracking the location in document and storing data in temporary variables
- **Parsing properties (validation, namespace awareness) are set through the SAXParser or underlying XMLReader**

# XSLT Processing with Java

- XSLT Overview
- Understanding XPath notation
- Processing elements in XSLT templates
- XSLT installation and setup
- An XSL Transformer
- Example:
  - Document Editor
  - XSLT custom tag

# Extensible Stylesheet Language Transformations

- **XSLT applies user-defined transformations to an XML document**
  - Transformed output can be:
    - HTML, XML, WML, etc.
- **XSLT Versions**
  - XSLT 1.0 (Nov 1999)
  - XSLT 2.0 (requirements, Feb 2001)
    - Namespace addition
  - Official Website for XSLT
    - <http://www.w3.org/Style/XSL/>

# Extensible Stylesheet Language (XSL)

- XSL is a language for expressing stylesheets
  - XSLT
    - Transformation of XML document
    - <http://www.w3.org/TR/xslt>
  - XPath
    - An expression language used by XSLT to locate elements and/or attributes within an XML document
    - <http://www.w3.org/TR/xpath>
  - XSL-FO (Formatting Objects)
    - Specifies the formatting properties for rendering the document
    - <http://www.w3.org/TR/XSL/XSL-FO/>

# XSLT Advantages and Disadvantages

- **Advantages**
  - Easy to merge XML data into a presentation
  - More resilient to changes in the details of the XML documents than low-level DOM and SAX
  - Database queries can be retuned in XML
    - Insensitive to column order
- **Disadvantages**
  - Memory intensive and suffers a performance penalty
  - Difficult to implement complicated business rules
  - Have to learn a new language
  - Can't change the value of variables (requires recursion)

# XSLT Parsers

- **Apache Xalan**
  - <http://xml.apache.org/xalan/>
- **Oracle**
  - <http://technet.oracle.com/tech/xml/>
- **Saxon**
  - <http://saxon.sourceforge.net/>
  - Written by Michael Kay
- **Microsoft's XML Parser 4.0 (MSXML)**
  - <http://www.microsoft.com/xml/>

# XSLT Installation and Setup

## 1. Download a XSLT compliant parser

- XSLT parsers at  
<http://www.xmlsoftware.com/xslt/>
- Recommend Apache Xalan-J 2.3 parser at  
<http://xml.apache.org/xalan-j/>

## 2. Download a SAX 2-compliant parser

- Java-based XML parsers at  
[http://www.xml.com/pub/rg/Java\\_Parsers](http://www.xml.com/pub/rg/Java_Parsers)
- Recommend Apache Xerces-J parser at  
<http://xml.apache.org/xerces-j/>
- Note that `xerces.jar` is bundled with the Xalan-J download and is also prebundled with Tomcat 4.0

# XSLT Installation and Setup (continued)

## 3. Download the Java API for XML Processing (JAXP)

- JAXP provides TrAX, a small layer on top of SAX and DOM which supports specifying transformers through system properties versus hard coded values
- See <http://java.sun.com/xml/>
- Note that TrAX is incorporated in Xalan-J

## 4. Bookmark the Java XSLT API

- Xalan-Java API is located at  
<http://xml.apache.org/xalan-j/apidocs/index.html>

# XSLT Installation and Setup (continued)

## 5. Set your CLASSPATH to include the XSLT and XML parser classes

```
set CLASSPATH=xalan_install_dir\xalan.jar;  
xalan_install_dir\xerces.jar;%CLASSPATH%
```

or

```
setenv CLASSPATH xalan_install_dir\xalan.jar:  
xalan_install_dir\xerces.jar:$CLASSPATH
```

- For servlets, place xalan.jar and xerces.jar in the server's lib directory
- Note that xerces.jar is the parser used by Tomcat 4.0 and is already located at (\$CATALINA\_HOME/common/lib/)

# Aside: Using Xerces with Tomcat 3.2.x

- **Problem**
  - Tomcat 3.2.x may load the provided DOM Level 1 parser first (`parser.jar`) from the `TOMCAT_HOME/lib` directory before `xerces.jar`, effectively eliminating SAX 2 and DOM Level 2 support (namespaces)
- **Solutions**
  1. Set up a static CLASSPATH and place `xalan.jar` and `xerces.jar` *first* in the list
  2. As the files are loaded alphabetically, rename `parser.jar` to `z_parser.jar` and `xml.jar` to `z_xml.jar`

# XSL Transformations

- **Use**
  - **XPath** to identify (select) parts of an XML document
  - **XSLT templates** to apply transformations
- **Requires**
  - Well formed XML document
  - XSL document (style sheet) that contains formatting and transformation templates
  - XSLT parser to perform the transformation

# Simple XSLT Example

- The following example illustrates transforming an XML document into an **HTML TABLE**
  - Input
    - Style sheet (XSL): `table.xsl`
    - XML document: `acronym.xml`
  - Output
    - HTML document: `acronym.html`

# XSLT Stylesheet: table.xsl

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="html" />
  <xsl:template match="/">
    <TABLE CELLPADDING="3" BORDER="1" ALIGN="CENTER">
      <!-- Build table header, by selecting the
          name of each element in the first ROW. -->
      <TR><TH></TH>
      <xsl:for-each select="ROWSET/ROW[1]/*">
        <TH><xsl:value-of select="name()" /></TH>
      </xsl:for-each>
      </TR>
      <!-- Apply template to build table rows -->
      <xsl:apply-templates select="ROWSET" />
    </TABLE>
  </xsl:template>
  ...

```

# XSLT Stylesheet: table.xsl (continues)

...

```
<xsl:template match="ROW">
  <TR><TD><xsl:number /></TD>
    <!-- Select all elements in the ROW.
        Populate each TD with the corresponding
        text value of the element.
        Note: &amp;#160; produces &nbsp; by Xalan --&gt;
    &lt;xsl:for-each select="*"
      &lt;TD&gt;&lt;xsl:value-of select=". " /&gt;&amp;#160;&lt;/TD&gt;
    &lt;/xsl:for-each&gt;
  &lt;/TR&gt;
&lt;/xsl:template&gt;
&lt;/xsl:stylesheet&gt;</pre>
```

# XML Document: acronyms.xml

```
<?xml version="1.0"?>
<ROWSET>
  <ROW>
    <ACRONYM>DOM</ACRONYM>
    <DESCRIPTION>Document Object Model</DESCRIPTION>
  </ROW>
  <ROW>
    <ACRONYM>JAXP</ACRONYM>
    <DESCRIPTION>Java API for XML Parsing</DESCRIPTION>
  </ROW>
  <ROW>
    <ACRONYM>SAX</ACRONYM>
    <DESCRIPTION>Simple API for XML</DESCRIPTION>
  </ROW>
  <ROW>
    <ACRONYM>TrAX</ACRONYM>
    <DESCRIPTION>Transformation API for XML</DESCRIPTION>
  </ROW>
  <ROW>
    <ACRONYM>XSLT</ACRONYM>
    <DESCRIPTION>XSL Transformation</DESCRIPTION>
  </ROW>
</ROWSET>
```

# Transforming the XML Document

- Use Xalan command-line interface

```
> java org.apache.xalan.xslt.Process  
      -in acronyms.xml  
      -xsl table.xsl  
      -out acronyms.html
```

# Transformation Result

```
<TABLE ALIGN="CENTER" BORDER="1" CELLPADDING="3">
<TR>
<TH></TH><TH>ACRONYM</TH><TH>DESCRIPTION</TH>
</TR>
<TR>
<TD>1</TD><TD>DOM </TD><TD>Document Object Model </TD>
</TR>
<TR>
<TD>2</TD><TD>JAXP </TD><TD>Java API for XML Parsing </TD>
</TR>
<TR>
<TD>3</TD><TD>SAX </TD><TD>Simple API for XML </TD>
</TR>
<TR>
<TD>4</TD><TD>TrAX </TD><TD>Transformation API for
      XML </TD>
</TR>
<TR>
<TD>5</TD><TD>XSLT </TD><TD>XSL Transformation </TD>
</TR>
</TABLE>
```

# Transformation Result (continued)

A screenshot of Microsoft Internet Explorer version 6.0 displaying a table of acronyms and their descriptions. The table has two columns: 'ACRONYM' and 'DESCRIPTION'. The rows are numbered 1 through 5. The browser interface includes a menu bar (File, Edit, View, Favorites, Tools, Help), a toolbar with various icons, and a status bar at the bottom.

|   | ACRONYM | DESCRIPTION                |
|---|---------|----------------------------|
| 1 | DOM     | Document Object Model      |
| 2 | JAXP    | Java API for XML Parsing   |
| 3 | SAX     | Simple API for XML         |
| 4 | TrAX    | Transformation API for XML |
| 5 | XSLT    | XSL Transformation         |

# Understanding XPath

- **XPath is an *expression language* to:**
  - Identify parts (location paths) of the input document
    - Commonly used in **match** and **select** attributes in XSLT elements

```
<xsl:template match="/name/first" >  
    ...  
</xsl:template>
```

- Test boolean conditions
- Manipulate strings
- Perform numerical calculations

# Location Paths

- **Location paths are interpreted with respect to a context**
  - Or simply, the node in the tree from which the expression is evaluated
- **The evaluated expression represents a set of nodes matching the condition**
  - Possibly the empty set if no matches occur
- **A location path consists of one or more location steps separated by / or //**
- **Paths can be relative or absolute**

# Simple Location Paths

- **Matching the root node**

- A leading `/` indicates the root node

```
<xsl:template match="/" >  
    <!-- Matches the root node. -->  
</xsl:template>
```

- **Matching all children**

- Use the `*` wildcard to select all element nodes in the current context

```
<xsl:template match="*" >  
    <!-- Matches all children nodes. -->  
</xsl:template>
```

# Simple Location Paths (continued)

- **Matching an element**

- Use `/` to separate elements when referring to a child
- Use `//` to indicate that zero or more elements may occur between the slashes

```
<xsl:template match="/catalog/*/manufacturer" >
    <!-- Match all manufacturer elements -->
    <!-- that are a grandchild of catalog. -->
</xsl:template>
```

```
<xsl:template match="order//item" >
    <!-- Match all item elements that are -->
    <!-- descendants of order. -->
</xsl:template>
```

# Matching with Predicates

- **Matching a specific element**

- Use [...] as a predicate filter to select a *particular* context node
- The predicate is evaluated as a boolean expression; if the condition is true, then the node is selected

```
<xsl:template match="author/name[middle]">
    <!-- Match all name elements that have an          -->
    <!-- author parent and a middle child.           -->
</xsl:template>
```

```
<xsl:template match="/ROW/ROWSET[1]">
    <!-- Match the first ROWSET element that is      -->
    <!-- a child of ROW (from the root).           -->
</xsl:template>
```

# Matching with Predicates (continued)

- Matching a specific attribute
  - Use the @ sign followed by the attribute name to select a particular node

```
<xsl:template match="order[@discount]" >
    <!-- Match all order elements that have a      -->
        <!-- discount attribute.                  -->
</xsl:template>

<xsl:template match="catalog/item[@id='3145']" >
    <!-- Match all item elements that are a child -->
        <!-- of catalog and have an id attribute with -->
            <!-- a value of 3145.                      -->
</xsl:template>
```

# XSLT Stylesheet Elements

- **Matching and selection templates**
  - xsl:template
  - xsl:apply-templates
  - xsl:value-of
- **Branching elements**
  - xsl:for-each
  - xsl:if
  - xsl:choose

# XSLT template Element

- **xsl:template match="xpath"**
  - Defines a template rule for producing output
  - Applied only to nodes which match the pattern
  - Invoked by using <xsl:apply-templates>

```
<xsl:template match="/">
  <html>
    <head><title>Ktee Siamese</title></head>
    <body>
      <xsl:apply-templates/>
    </body>
  </html>
</xsl:template>
```

```
<xsl:template match="name">
  <h2><xsl:value-of select=". . ."/></h2>
</xsl:template>
```

# XSLT apply-templates Element

- **xsl:apply-templates**
  - Applies matching templates to the children of the context node

```
<xsl:template match="/">
  <html>
    <head><title>Ktee Siamese</title></head>
    <body>
      <xsl:apply-templates />
    </body>
  </html>
</xsl:template>

<xsl:template match="name">
  <h2><xsl:value-of select=". "/></h2>
</xsl:template>
```

# XSLT value-of Element

- **xsl:value-of select="expression"**
  - Evaluates the expression as a string and sends the result to the output
  - Applied only to the first match
  - ". " selects the **text value** of the current node

```
<xsl:template match="name">
    <!-- Select text of name node. -->
    <h2><xsl:value-of select=". " /></h2>
</xsl:template>
```

# XSLT value-of Element (continued)

- Example

```
<xsl:template match="daylily">
  <TR>
    <!-- Selects the award child of the
        daylily element. By default, outputs
        the text of the award element. -->
    <TD><xsl:value-of select="award" /></TD>

    <!-- Selects the code attribute of the
        daylily's bloom child and outputs
        the text of the attribute. -->
    <TD><xsl:value-of select="bloom/@code" /></TD>
  </TR>
</xsl:template>
```

# XSLT for-each Element

- **xsl:for-each select="expression"**
  - Processes each node selected by the XPath expression

```
<book>
    <author>Larry Brown</author>
    <author>Marty Hall</author>
</book>
```

---

```
<xsl:template match="book">
    <!-- Selects each author name.    -->
    <xsl:for-each select="author">
        <b><xsl:value-of select=". " /></b>
    </xsl:for-each>
</xsl:template>
```

# XSLT if Element

- **xsl:if test="expression"**
  - Evaluates the expression to a boolean and if true, applies the template body
  - XSLT has no if-else construct (use choose)

```
<xsl:template match="ROW">
    <!-- Selects first node in the node set. -->
    <xsl:if test="position() = first()">
        <b><xsl:value-of select=". " />
    </xsl:if>
</xsl:template>

<xsl:template match="ROW">
    <!-- Select if the current node has children. -->
    <xsl:if test="node()">
        <xsl:apply-templates />
    </xsl:if>
</xsl:template>
```

# XSLT choose Element

- **xsl:choose**
  - Select any number of alternatives
  - Instruction to use in place of if-else or switch construct found in other programming languages

```
<xsl:choose>
    <xsl:when test="not(text())">
        Missing value!
    </xsl:when>
    <xsl:otherwise>
        <xsl:value-of select=". " />
    </xsl:otherwise>
</xsl:choose>
```

# XSLT output Element

- **xsl:output**
  - Controls the format of the stylesheet output
  - Useful attributes:  
`method= " [html|xml|text] "`  
`indent=" [yes|no] "`  
`version="version"`  
`doctype-public="specification"`  
`encoding="encoding"`  
`standalone=" [yes|no] "`
- Example

```
<xsl:output method="html"  
doctype-public="-//W3C//DTD HTML 4.0 Transitional//EN"/>
```

# Steps for Translating a Document

1. Tell the system which parser to use
2. Establish a factory in which to create transformations
3. Create a transformer for a particular style sheet
4. Invoke the transformer to process the document

# Step 1: Specifying a Transformer

## 1. Approaches to specify a transformer

- Set a system property for `javax.xml.transform.TransformerFactory`
- Specify the parser in `jre_dir/lib/jaxp.properties`
- Through the J2EE Services API and the class specified in `META-INF/services/javax.xml.transform.TransformerFactory`
- Use system-dependant default parser (check documentation)

# Specifying a Transformer, Example

- **The following example:**

- Permits the user to specify the transformer through the command line -D option

```
java -Djavax.xml.transform.TransformerFactory=
      weblogic.apache.xalan.processor.TransformerFactoryImpl ...
```

- Uses the Apache Xalan transformer otherwise

```
public static void main(String[] args) {
    String jaxpPropertyName =
        "javax.xml.transform.TransformerFactory ";
    if (System.getProperty(jaxpPropertyName) == null) {
        String apacheXercesPropertyValue =
            "org.apache.xalan.xsltc.trax.TransformerFactoryImpl";
        System.setProperty(jaxpPropertyName,
                           apacheXercesPropertyValue);
    }
    ...
}
```

# Step 2: Creating a Transformer Factory

- Establish a factory in which to create transformations

```
TransformerFactory factory =  
    new TransformerFactory.newInstance();
```

- May create multiple transformers from the same factory

# Step 3: Creating a Transformer

- Create a transformer for a particular style sheet

```
Source xsl = new StreamSource(xslStream);  
Templates template = factory.newTemplates(xsl);  
Transformer transformer =  
    template.newTransformer();
```

# Step 4: Invoke the Transformer

- **Invoke the transformer to process the document**

```
Source xml = new StreamSource(xmlStream) ;  
Result result = new StreamResult(outputStream) ;  
transformer.transform(xml, result) ;
```

- Create a `StreamSource` from a File, Reader, InputStream or URI reference (String)
- Create a `StreamResult` from a File, Writer, OutputStream or URI reference (String)

# A Simple XSL Transformer

- Creates an XSL transformer for processing an XML and XSL document
  - Provides multiple overloaded process methods for handling different input and output streams

```
public class XslTransformer {  
    private TransformerFactory factory;  
  
    // Use system defaults for transformer.  
    public XslTransformer() {  
        factory = TransformerFactory.newInstance();  
    }  
    ...  
}
```

# A Simple XSL Transformer

```
/** For transforming an XML documents as a String StringReader
 * residing in memory, not on disk. The output document could
 * easily be handled as a String (StringWriter) or as a
 * JSPWriter in a JavaServer page.
 */
public void process(Reader xmlFile, Reader xslFile,
                    Writer output)
    throws TransformerException {
process(new StreamSource(xmlFile),
        new StreamSource(xslFile),
        new StreamResult(output));
}

/** For transforming an XML and XSL document as Files,
 * placing the result in a Writer.
 */
public void process(File xmlFile, File xslFile,
                    Writer output)
    throws TransformerException {
process(new StreamSource(xmlFile),
        new StreamSource(xslFile),
        new StreamResult(output));
}
```

# Simple XSL Transformer (continued)

```
/** Transform an XML File based on an XSL File, placing the
 * resulting transformed document in an OutputStream.
 * Convenient for handling the result as a FileOutputStream or
 * ByteArrayOutputStream.
 */
public void process(Source xml, Source xsl, Result result)
                    throws TransformerException {
    try {
        Templates template = factory.newTemplates(xsl);
        Transformer transformer = template.newTransformer();
        transformer.transform(xml, result);
    } catch(TransformerConfigurationException tce) {
        throw new TransformerException(tce.getMessageAndLocation());
    } catch (TransformerException te) {
        throw new TransformerException(te.getMessageAndLocation());
    }
}
```

# Example 1: XSLT Document Editor

- **Objective**
  - Provide a graphical interface for editing XML and XSL documents, and to view the transformed result
- **Approach**
  - Use a Swing JTabbedPane with three tabs (XML, XSL, XSLT) to present each of the three corresponding documents
  - Each document is represented by a JEditorPane
    - XML and XSL panes are editable
  - Selecting the XSLT tab performs the transformation

# Example 1: XsltEditor

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.event.*;
import java.io.*;
import javax.xml.transform.*;
import cwp.XslTransformer;

public class XsltEditor extends JFrame
    implements ChangeListener {
    private static final int XML = 0;
    private static final int XSL = 1;
    private static final int XSLT = 2;
    private Action openAction, saveAction, exitAction;
    private JTabbedPane tabbedPane;
    private DocumentPane[] documents;
    private XslTransformer transformer;
    ...
}
```

# Example 1: XsltEditor (continued)

```
...
/** Checks to see which tabbed pane was selected by the
 * user. If the XML and XSL panes hold a document, then
 * selecting the XSLT tab will perform the transformation.
 */
public void stateChanged(ChangeEvent event) {
    int index = tabbedPane.getSelectedIndex();
    switch (index) {
        case XSLT: if (documents[XML].isLoaded() &&
                        documents[XSL].isLoaded()) {
                        doTransform();
                    }
        case XML:
        case XSL: updateMenuAndTitle(index);
                    break;
        default:
    }
}
```

# Example 1: XsltEditor (continued)

```
...
private void doTransform() {
    StringWriter strWriter = new StringWriter();
    try {
        Reader xmlInput =
            new StringReader(documents[XML].getText());
        Reader xslInput =
            new StringReader(documents[XSL].getText());
        transformer.process(xmlInput, xslInput, strWriter);
    } catch(TransformerException te) {
        JOptionPane.showMessageDialog(this,
            "Error: " + te.getMessage());
    }
    documents[XSLT].setText(strWriter.toString());
}
...
}
```

# Example 1: DocumentPane

```
public class DocumentPane extends JEditorPane {  
    public static final String TEXT = "text/plain";  
    public static final String HTML = "text/html";  
    private boolean loaded = false;  
    private String filename = "";  
  
    /** Set the current page displayed in the editor pane,  
     * replacing the existing document.  
     */  
    public void setPage(URL url) {  
        loaded = false;  
        try {  
            super.setPage(url);  
            File file = new File(getPage().toString());  
            setFilename(file.getName());  
            loaded = true;  
        } catch (IOException ioe) {  
            System.err.println("Unable to set page: " + url);  
        }  
    }  
}
```

# Example 1: DocumentPane (continued)

```
public void setText(String text) {  
    super.setText(text);  
    setFilename("");  
    loaded = true;  
}  
  
public void loadFile(String filename) {  
    try {  
        File file = new File(filename);  
        setPage(file.toURL());  
    } catch (IOException mue) {  
        System.err.println("Unable to load file: " + filename);  
    }  
}  
  
public boolean isLoaded() {  
    return (loaded);  
}  
...  
}
```

# Example 1: XsltEditor, Result

The image shows three windows illustrating an XSLT transformation process:

- Left Window (XSLT Example - [perennials.xml]):** Displays the XML input file `perennials.xml`. The XML structure includes a root element `<perennials>`, a `<daylily>` element with attributes `status="in-stock"` and `cultivar="Luxury Lace"`, and two `<award>` elements. The first award has a note `"small-flowered"` and a year `1965`. The second award also has a year `1965`.
- Middle Window (XSLT Example - [perennials.xsl]):** Displays the XSLT stylesheet `perennials.xsl`. It defines an XSLT template for the root node that outputs an HTML document. The template includes a `CAPTAIN` element with the value `"Stout Medal Award"` and a `TABLE` element with `CELLPADDING="3"`.
- Right Window (XSLT Example):** Shows the resulting HTML output. The page title is `"Stout Medal Award"`. The table has columns: `Year`, `Cultivar`, `Bloom Season`, and `Cost`. The data rows are:

| Year | Cultivar      | Bloom Season | Cost  |
|------|---------------|--------------|-------|
| 1965 | Luxury Lace   | M            | 11.75 |
| 1976 | Green Flutter | M            | 7.50  |
| 1984 | My Belle      | E            | 12.00 |
| 1985 | Stella De Oro | E-L          | 5.00  |
| 1989 | Brocaded Gown | E            | 14.50 |

Below the table, the text `E-early M-midseason L-late` is displayed.

# Example 2: XSLT Custom Tag

- **Objective**
  - Develop a JSP custom tag to transform an XML document and create an HTML table
- **Problem**
  - THEAD, TBODY, and TFOOT elements supported by Internet Explorer, but not by Netscape 4.x

# Example 2: XSLT Custom Tag (continued)

- Approach
  - Use different stylesheet for Internet Explorer and Netscape
  - Determine the browser type based on the User-Agent HTTP header
  - Provide both stylesheets in custom tag

```
<cwp:xsltransform xml='perennials.xml'  
                   xslie='perennials-ie.xsl'  
                   xslns='perennials-ns.xsl' />
```

# Example 2: Custom Tag Specification, Xsltransform.tld

```
...
<tag>
    <name>xsltransform</name>
    <tagclass>cwp.tags.XslTransformTag</tagclass>
    <attribute>
        <name>xml</name>
        <required>yes</required>
    </attribute>
    <attribute>
        <name>xslie</name>
        <required>false</required>
    </attribute>
    <attribute>
        <name>xslns</name>
        <required>true</required>
    </attribute>
</tag>
```

# Example 2: XslTransformTag

```
public class XslTransformTag extends TagSupport {  
    private static final int IE = 1;  
    private static final int NS = 2;  
  
    public int doStartTag() throws JspException {  
        ServletContext context = pageContext.getServletContext();  
        HttpServletRequest request =  
            (HttpServletRequest)pageContext.getRequest();  
  
        File xslFile = null;  
        if ((browserType(request) == IE) &&  
            (getXsliel() != null)) {  
            xslFile = new File(path + getXsliel());  
        } else {  
            xslFile = new File(path + getXslns());  
        }  
        File xmlFile = new File(path + getXml());  
        ...  
    }  
}
```

# Example 2: XslTransformTag (continued)

```
// doStartTag
try {
    JspWriter out = pageContext.getOut();
    XslTransformer transformer = new XslTransformer();
    transformer.process(xmlFile, xslFile, out);
}
catch(TransformerException tx) {
    context.log("XslTransformTag: " + tx.getMessage());
}
return(SKIP_BODY);
}
...
}
```

# Example 2: XslTransformTag (continued)

```
// Determine the browser type based on the User-Agent
// HTTP request header.

private int browserType(HttpServletRequest request) {
    int type = NS;
    String userAgent = request.getHeader("User-Agent");
    if ((userAgent != null) &&
        (userAgent.indexOf("IE") >= 0)) {
        type = IE;
    }
    return(type);
}
```

# Example 2: Daylilies.jsp

```
<HTML>
<HEAD>
    <TITLE>Daylilies</TITLE>
</HEAD>
<BODY>
<%@ taglib uri="cwp-tags/xsltransform.tld" prefix="cwp" %>

<H1 ALIGN="CENTER">Katie's Favorite Daylilies</H1>
<P>
<cwp:xsltransform xml='perennials.xml'
                  xslice='perennials-ie.xsl'
                  xslns='perennials-ns.xsl' />

</BODY>
</HTML>
```

# Example 2: perennials-ie.xsl

```
<xsl:template match="/">
  <TABLE CELLPADDING="3" RULES="GROUPS" ALIGN="CENTER">
    <CAPTION>Stout Medal Award</CAPTION>
    <COLGROUP>
      <COL ALIGN="CENTER"/>
      <COL ALIGN="LEFT"/>
      <COL ALIGN="CENTER"/>
      <COL ALIGN="RIGHT"/>
    </COLGROUP>
    <THEAD>
      <TR><TH>Year</TH><TH>Cultivar</TH><TH>Bloom Season</TH>
        <TH>Cost</TH></TR>
    </THEAD>
    <TBODY>
      <xsl:apply-templates
        select="/perennials/daylily[award/name='Stout Medal']"/>
    </TBODY>
    <TFOOT>
      <TR><TD COLSPAN="4">E-early M-midseason L-late</TD></TR>
    </TFOOT>
  </TABLE>
</xsl:template>
```

# Example 2: perennials-ns.xsl

```
<xsl:template match="/">
  <TABLE CELLPADDING="3" BORDER="1" ALIGN="CENTER">
    <CAPTION>Stout Medal Award</CAPTION>
    <TR>
      <TH>Year</TH>
      <TH>Cultivar</TH>
      <TH>Bloom Season</TH>
      <TH>Cost</TH>
    </TR>
    <xsl:apply-templates
      select="/perennials/daylily[award/name='Stout Medal']"/>
    <TR>
      <TD COLSPAN="4" ALIGN="CENTER">
        E-early M-midseason L-late</TD>
    </TR>
  </TABLE>
</xsl:template>
```

# XSLT Custom Tag, Result

Daylilies - Microsoft Internet Explorer

File Edit View Favorites Tools Help

Back Search Favorites History Links

## Katie's Favorite Daylilies

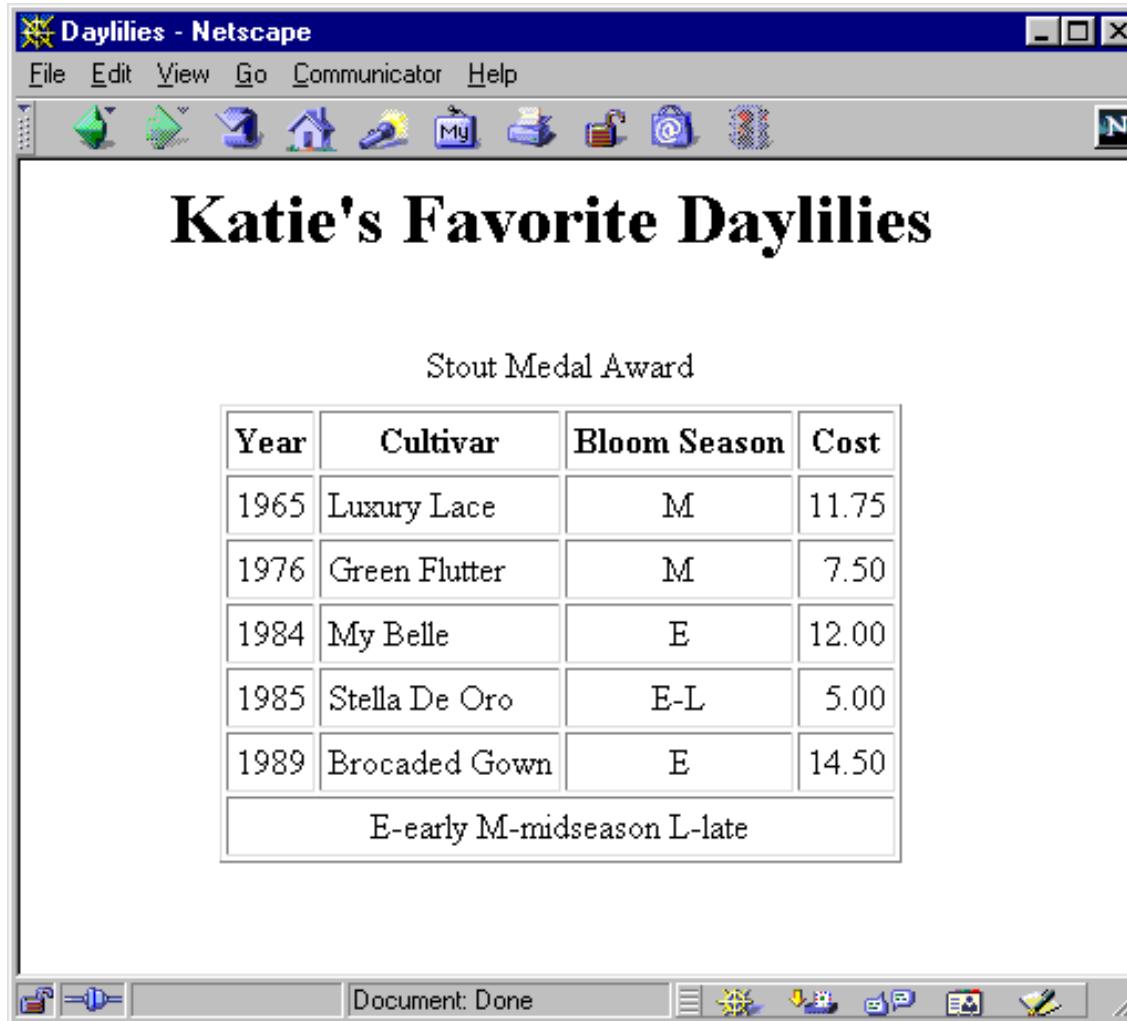
Stout Medal Award

| Year | Cultivar      | Bloom Season | Cost  |
|------|---------------|--------------|-------|
| 1965 | Luxury Lace   | M            | 11.75 |
| 1976 | Green Flutter | M            | 7.50  |
| 1984 | My Belle      | E            | 12.00 |
| 1985 | Stella De Oro | E-L          | 5.00  |
| 1989 | Brocaded Gown | E            | 14.50 |

E-early M-midseason L-late

Done Local intranet

# XSLT Custom Tag, Result



# Summary

- **XSLT specifies how to transform XML into HTML, XML, or other document formats**
- **XPath pattern selects a set of nodes for processing**
- **Control conditional processing through XSLT templates (elements)**
- **Apache Xalan-J in a popular XSLT compliant transformer**
  - InputSource document is typically a File or String (StringReader)
  - Result document typically sent to a File or JspWriter in a servlet