



core  
**WEB**  
programming

## Document Object Model

DOM

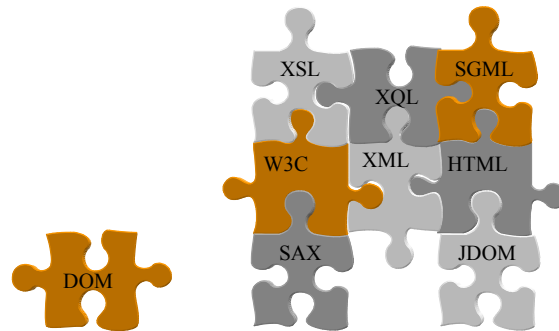
© 2001 Marty Hall, Larry Brown <http://www.corewebprogramming.com>

## Agenda

- 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?

## Overview

- **Programming XML focuses on DOM and SAX technologies**



3

DOM

www.corewebprogramming.com

## Document Object Model (DOM)

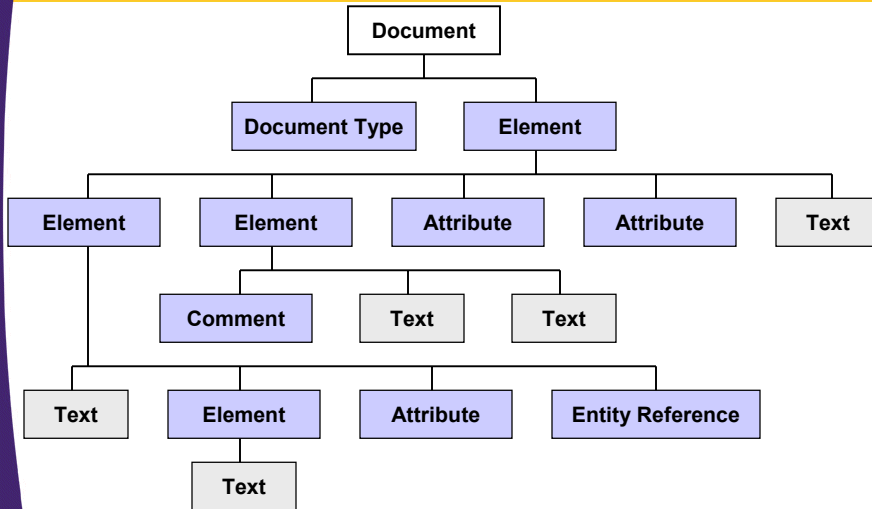
- **DOM supports navigating and modifying XML documents**
  - Hierarchical tree representation of document
    - Tree follows standard API
    - Creating tree is vendor specific
- **DOM is a language-neutral specification**
  - Bindings exists for Java, C++, CORBA, JavaScript
- **DOM Versions**
  - DOM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

4

DOM

www.corewebprogramming.com

# DOM Tree



5

DOM

[www.corewebprogramming.com](http://www.corewebprogramming.com)

# 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

6

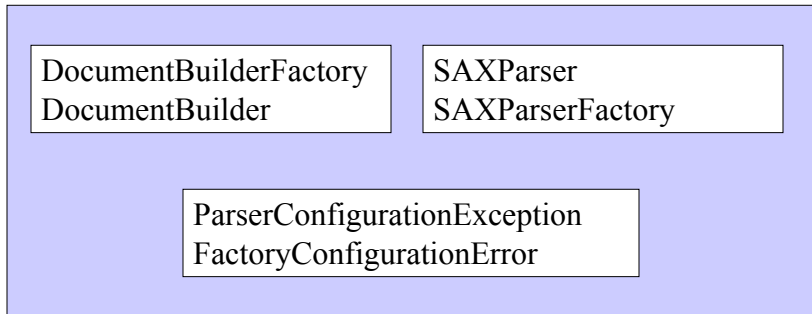
DOM

[www.corewebprogramming.com](http://www.corewebprogramming.com)

# Java API for XML Parsing (JAXP)

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

`javax.xml.parsers`



7

DOM

[www.corewebprogramming.com](http://www.corewebprogramming.com)

# 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/>

8

DOM

[www.corewebprogramming.com](http://www.corewebprogramming.com)

## DOM Installation and Setup (continued)

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

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

or

```
setenv CLASSPATH xerces_install_dir\xerces.jar:  
jaxp_install_dir\jaxp.jar:$CLASSPATH
```

- For servlets, place `xerces.jar` and `jaxp.jar` in the server's `lib` directory
  - Note: On Tomcat 3.2.x you may need set up a static CLASSPATH and place `xerces.jar` and `jaxp.jar` *first* in the list

## Aside: Using Xerces with Tomcat

- **Problem**
  - Tomcat 3.2+ may load the provided DOM Level 1 parser first (`parser.jar`) from the `TOMCAT_HOME/lib` directory before `xerces.jar`, effectively eliminating DOM Level 2 support (namespaces)
- **Solutions**
  1. Set up a static CLASSPATH and place `xerces.jar` and `jaxp.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

19

DOM

[www.corewebprogramming.com](http://www.corewebprogramming.com)

## 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

20

DOM

[www.corewebprogramming.com](http://www.corewebprogramming.com)

## 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));
    }
}
```

23

DOM

[www.corewebprogramming.com](http://www.corewebprogramming.com)

# 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));
    }
}
```

24

DOM

[www.corewebprogramming.com](http://www.corewebprogramming.com)

## 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. We have two types
    // of nodes here: the ones corresponding to the actual
    // XML structure and the entries of the graphical JTree.
    // The convention is that 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);
}
}
```

27

DOM

www.corewebprogramming.com

## 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);
        }
        ...
    }
}
```

28

DOM

www.corewebprogramming.com

## 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);
    }
}
}
```

29

DOM

[www.corewebprogramming.com](http://www.corewebprogramming.com)

## DOM Example: perennials.xml

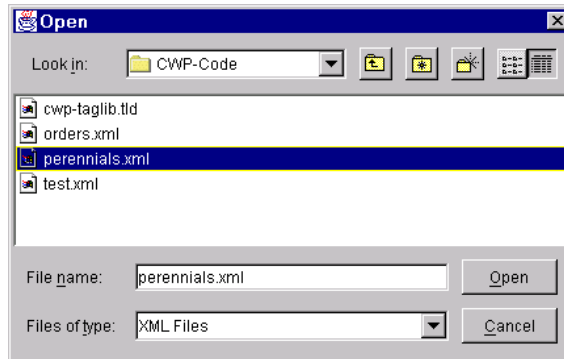
```
<?xml version="1.0"?>
<!DOCTYPE perennials SYSTEM "dtds/perennials.dtd">
<perennials>
  <daylily status="in-stock">
    <cultivar>Luxury Lace</cultivar>
    <award>
      <name>Stout Medal</name>
      <year>1965</year>
    </award>
    <award>
      <name note="small-flowered">Annie T. Giles</name>
      <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>
```

30

DOM

[www.corewebprogramming.com](http://www.corewebprogramming.com)

# DOM Example: Results

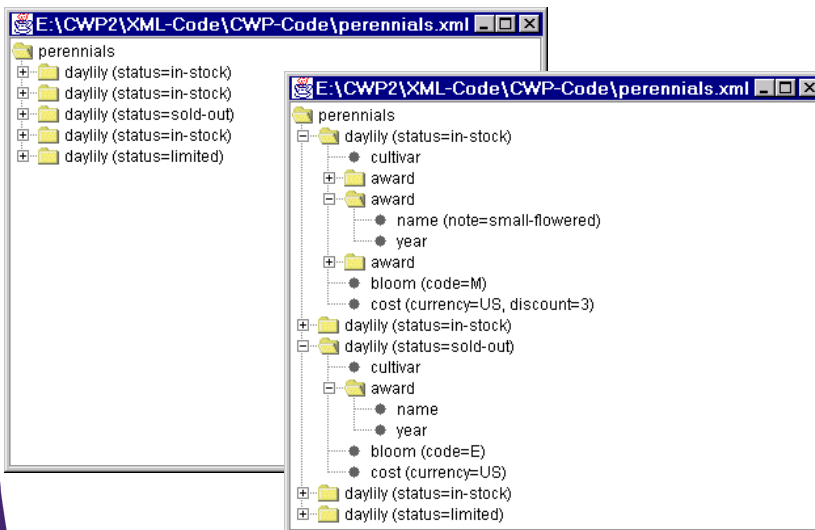


31

DOM

www.corewebprogramming.com

# DOM Example: Results (continued)



32

DOM

www.corewebprogramming.com

## 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**



*core*  
**WEB**  
*programming*

**Questions?**