



Introduction to XML and XML With Java

If you are looking for sample programs to parse a XML file using DOM/SAX parser or looking for a program to generate a XML file please proceed directly to [programs](#).

This small tutorial introduces you to the basic concepts of XML and using Xerces parser for Java to generate and parse XML.

The intended audience are XML beginners with knowledge of Java.

Last updated:

03 Oct 2005.Modified programs runs in JDK 1.3, JDK 1.4 and JDK 1.5. Xerces parser is used with JDK 1.4 and 1.3.

Contents

- [1 Introduction](#)
- [2 What is XML](#)
- [3 Advantages of XML](#)
 - [3.1 Readability](#)
 - [3.2 Hierarchical](#)
 - [3.3 Language Independent](#)
 - [3.4 OS independent](#)
- [4 Uses of XML](#)
 - [4.1 MetaContent](#)
 - [4.2 Messaging](#)
 - [4.3 Database](#)
- [5 Parsers](#)
 - [5.1 DOM](#)
 - [5.2 SAX](#)
 - [5.3 When to use DOM Parser](#)
 - [5.4 When to use SAX Parser](#)
 - [5.5 Validating And Non Validating](#)
 - [5.6 Well Formedness](#)
- [6 Programs](#)

- [6.1 Parsing and Printing](#)
 - [6.1.1 Using DOM](#)
 - [6.1.2 Using SAX](#)
- [6.2 Generating XML](#)
- [7 Instructions to run these programs](#)
 - [7.1 Using JDK 1.5](#)
 - [7.2 Using JDK 1.4](#)
 - [7.3 Using JDK 1.3](#)
- [Definitions](#)

Introduction:

This is a gentle introduction to XML and using XML with Java. We see what makes XML tick and the advantages of XML. Also we have programs to show how to use Xerces parser for parsing and printing an XML document, to generate XML from arbitrary data. These programs have been intentionally kept small to get the point across.

[Top](#)

2 What is XML ?

Extensible Markup Language

XML is text based markup language.

XML is more than html because html only deal with presentation and carries no information about the data itself.

In HTML

For example

```
<b>Seagull<b>
```

```
<i>3674<i>
```

```
<i> 34 </i>
```

could mean anything

In XML we represent it as.

```
<Personnel>
```

```
<Employee>
```

```
<Name>Seagull</Name>
```

```
< ID> 3674 </ID>
```

```
<Age>34</Age>
```

```
</Employee>  
</Personnel>
```

And this clearly expresses Seagull is an employee name his ID is 3674 and his age is 34.

HTML has limited set of Tags
But XML can be extended (i.e we can create our own tags)

[Top](#)

3 Advantages of XML

Here are a few advantages.

3.1 Readability

XML document is plain text and human readable, To edit/view XML documents any simple text editor will suffice .

3.2 Hierarchical

XML document has a tree structure which is powerful enough to express complex data and simple enough to understand

3.3 Language Independent

XML documents are language neutral.For e.g. a Java program can generate a XML which can be parsed by a program written in C++ or Perl.

3.4 OS Independent

XML files are Operating System independent.

[Top](#)

4 Uses of XML

4.1 MetaContent

To describe the contents of a document.

4.2 Messaging

Where applications or organizations exchanges data between them.

4.3 Database

The data extracted from the database can be preserved with original information and can be used more than one application in different ways. One application might just display the data and the other application might perform some complex calculation on this data

[Top](#)

5 Parsers

5.1 DOM (Document Object Model) parser - Tree Structure based API:

The Dom parser implements the dom api and it creates a DOM tree in memory for a XML document

5.2 SAX (Simple API For XML) parser - Event Based API

The SAX parser implements the SAX API and it is event driven interface. As it parses it invokes the callback methods

5.3 When to use DOM parser

- Manipulate the document
- Traverse the document back and forth
- Small XML files

Drawbacks of DOM parser
Consumes lot of memory

5.4 When to use SAX parser

- No structural modification
- Huge XML files

5.5 Validating And Non Validating

DOM and SAX can either be a validating or a non validating parser.

A validating parser checks the XML file against the rules imposed by DTD or XML Schema.

A non validating parser doesn't validate the XML file against a DTD or XML Schema.

Both Validating and non validating parser checks for the well formedness of the xml document

[Top](#)

5.6 Well Formedness

A well formed XML must satisfy these constraints

- a) All tags must have a corresponding ending tag.

Well Formed	Not Well formed
-------------	-----------------

<code><Personnel> <Employee> <Name>Seagull</Name> <ID> 3674 </ID> <Age>34</Age> </Employee> </Personnel></code>	<code><Personnel> <Employee> <Name>Seagull</Name> <ID> 3674 </ID> <Age>34 </Employee> </Personnel></code>
---	---

(Note: Missing ending tag for Age)

b) No overlapping tags

Well Formed	Not Well formed
<code><Personnel> <Employee> <Name>Seagull</Name> <ID> 3674 </ID> <Age>34</Age> </Employee> </Personnel></code>	<code><Personnel> <Employee> <Name>Seagull</Name> <ID> 3674 </ID> <Age>34</Age> </Personnel> </Employee></code>

Note: </Employee> and </Personnel > tags are not in the right order

[Top](#)

6 Programs

All these Programs uses Xerces for Java Parser.

6.1 Parsing XML

This is like "hello world" to programming. One of the first encounters with XML for a beginner would be to parse a XML file create Java Objects and manipulate them.

The idea here is to parse the [employees.xml](#) file with content as below

```
<?xml version="1.0" encoding="UTF-8"?>  
<Personnel>  
  <Employee type="permanent" >  
    <Name>Seagull</Name>  
    <Id>3674</Id>  
    <Age>34</Age>
```

```

</Employee>
<Employee type="contract">
  <Name>Robin</Name>
  <Id>3675</Id>
  <Age>25</Age>
</Employee>
<Employee type="permanent">
  <Name>Crow</Name>
  <Id>3676</Id>
  <Age>28</Age>
</Employee>
</Personnel>

```

From the parsed content create a list of Employee objects and print it to the console. The output would be something like

```

Employee Details - Name:Seagull, Type:permanent, Id:3674, Age:34.
Employee Details - Name:Robin, Type:contract, Id:3675, Age:25.
Employee Details - Name:Crow, Type:permanent, Id:3676, Age:28.

```

We will start with a DOM parser to parse the xml file, create Employee value objects and add them to a list. To ensure we parsed the file correctly let's iterate through the list and print the employees data to the console. Later we will see how to implement the same using SAX parser.

In a real world situation you might get a xml file from a third party vendor which you need to parse and update your database.

6.1.1 Using DOM

This program [DomParserExample.java](#) uses DOM API.

The steps are

- Get a document builder using document builder factory and parse the xml file to create a DOM object
- Get a list of employee elements from the DOM
- For each employee element get the id,name,age and type. Create an employee value object and add it to the list.
- At the end iterate through the list and print the employees to verify we parsed it right.

a) Getting a document builder

```

private void parseXmlFile(){
    //get the factory
    DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();

    try {

        //Using factory get an instance of document builder
        DocumentBuilder db = dbf.newDocumentBuilder();

        //parse using builder to get DOM representation of the XML file
        dom = db.parse("employees.xml");

    }catch(ParserConfigurationException pce) {
        pce.printStackTrace();
    }catch(SAXException se) {
        se.printStackTrace();
    }catch(IOException ioe) {
        ioe.printStackTrace();
    }

}
}

```

b) Get a list of employee elements

Get the rootElement from the DOM object. From the root element get all employee elements. Iterate through each employee element to load the data.

```

private void parseDocument(){
    //get the root element
    Element docEle = dom.getDocumentElement();

    //get a nodelist of elements
    NodeList nl = docEle.getElementsByTagName("Employee");
    if(nl != null && nl.getLength() > 0) {
        for(int i = 0 ; i < nl.getLength(); i++) {

            //get the employee element
            Element e1 = (Element)nl.item(i);

            //get the Employee object
            Employee e = getEmployee(e1);

            //add it to list
            myEmps.add(e);
        }
    }
}

```

```

    }
}

```

c) Reading in data from each employee.

```

/**
 * I take an employee element and read the values in, create
 * an Employee object and return it
 */
private Employee getEmployee(Element empEl) {
    //for each <employee> element get text or int values of
    //name ,id, age and name
    String name = getTextValue(empEl, "Name");
    int id = getIntValue(empEl, "Id");
    int age = getIntValue(empEl, "Age");

    String type = empEl.getAttribute("type");

    //Create a new Employee with the value read from the xml nodes
    Employee e = new Employee(name,id,age,type);

    return e;
}

/**
 * I take a xml element and the tag name, look for the tag and get
 * the text content
 * i.e for <employee><name>John</name></employee> xml snippet if
 * the Element points to employee node and tagName is 'name' I will return John
 */
private String getTextValue(Element ele, String tagName) {
    String textVal = null;
    NodeList nl = ele.getElementsByTagName(tagName);
    if(nl != null && nl.getLength() > 0) {
        Element el = (Element)nl.item(0);
        textVal = el.getFirstChild().getNodeValue();
    }

    return textVal;
}

```



```

/**
 * Calls getTextView and returns a int value
 */
private int getIntValue(Element ele, String tagName) {
    //in production application you would catch the exception
    return Integer.parseInt(getTextView(ele, tagName));
}

```

d) Iterating and printing.

```

private void printData(){
    System.out.println("No of Employees '" + myEmpIs.size() + "'");

    Iterator it = myEmpIs.iterator();
    while(it.hasNext()) {
        System.out.println(it.next().toString());
    }
}

```

[Top](#)

6.1.2 Using SAX

This program [SAXParserExample.java](#) parses a XML document and prints it on the console.

Sax parsing is event based modelling. When a Sax parser parses a XML document and every time it encounters a tag it calls the corresponding tag handler methods

when it encounters a Start Tag it calls this method
 public void startElement(String uri,...

when it encounters a End Tag it calls this method
 public void endElement(String uri,...

Like the dom example this program also parses the xml file, creates a list of employees and prints it to the console. The steps involved are

- Create a Sax parser and parse the xml
- In the event handler create the employee object
- Print out the data

Basically the class extends DefaultHandler to listen for call back events. And we register this handler with the Sax parser to notify us of call back events. We are only interested in *start event*, *end event* and *character event*. In *start event* if the element is employee we create a new instant of employee object and if the element is Name/Id/Age we initialize the character buffer to get the text value. In *end event* if the node is employee then we know we are at the end of the employee node and we add the Employee object to the list. If it is any other node like Name/Id/Age we call the corresponding methods like setName/SetId/setAge on the Employee object. In *character event* we store the data in a temp string variable.

a) Create a Sax Parser and parse the xml

```
private void parseDocument() {
    //get a factory
    SAXParserFactory spf = SAXParserFactory.newInstance();
    try {
        //get a new instance of parser
        SAXParser sp = spf.newSAXParser();

        //parse the file and also register this class for call backs
        sp.parse("employees.xml", this);

    } catch (SAXException se) {
        se.printStackTrace();
    } catch (ParserConfigurationException pce) {
        pce.printStackTrace();
    } catch (IOException ie) {
        ie.printStackTrace();
    }
}
```

b) In the event handlers create the Employee object and call the corresponding setter methods.

```
//Event Handlers
public void startElement(String uri, String localName, String qName,
    Attributes attributes) throws SAXException {
    //reset
    tempVal = "";
    if(qName.equalsIgnoreCase("Employee")) {
        //create a new instance of employee
        tempEmp = new Employee();
    }
}
```

```

        tempEmp.setType(attributes.getValue("type"));
    }

    public void characters(char[] ch, int start, int length) throws SAXException {
        tempVal = new String(ch,start,length);
    }

    public void endElement(String uri, String localName,
        String qName) throws SAXException {
        if(qName.equalsIgnoreCase("Employee")) {
            //add it to the list
            myEmps.add(tempEmp);
        }else if (qName.equalsIgnoreCase("Name")) {
            tempEmp.setName(tempVal);
        }else if (qName.equalsIgnoreCase("Id")) {
            tempEmp.setId(Integer.parseInt(tempVal));
        }else if (qName.equalsIgnoreCase("Age")) {
            tempEmp.setAge(Integer.parseInt(tempVal));
        }
    }
}

```

c) Iterating and printing.

```

private void printData(){
    System.out.println("No of Employees " + myEmps.size() + " .");
    Iterator it = myEmps.iterator();
    while(it.hasNext()) {
        System.out.println(it.next().toString());
    }
}

```

[Top](#)

6.2 Generating XML

The previous programs illustrated how to parse an existing XML file using both SAX and DOM Parsers. But generating a XML file from scratch is a different story, for instance you might like to generate a xml file for

the data extracted from a database. To keep the example simple this program [XMLCreatorExample.java](#) generates XML from a list preloaded with hard coded data. The output will be book.xml file with the following content.

```
<?xml version="1.0" encoding="UTF-8"?>
<Books>
  <Book Subject="Java 1.5">
    <Author>Kathy Sierra .. etc</Author>
    <Title>Head First Java</Title>
  </Book>
  <Book Subject="Java Architect">
    <Author>Kathy Sierra .. etc</Author>
    <Title>Head First Design Patterns</Title>
  </Book>
</Books>
```

The steps involved are

- Load Data
- Get an instance of Document object using document builder factory
- Create the root element Books
- For each item in the list create a Book element and attach it to Books element
- Serialize DOM to FileOutputStream to generate the xml file "book.xml".

a) Load Data.

```
/**
 * Add a list of books to the list
 * In a production system you might populate the list from a DB
 */
private void loadData(){
    myData.add(new Book("Head First Java",
        "Kathy Sierra .. etc", "Java 1.5"));
    myData.add(new Book("Head First Design Patterns",
        "Kathy Sierra .. etc", "Java Architect"));
}
```

c) Getting an instance of DOM.

```

/**
 * Using JAXP in implementation independent manner create a document object
 * using which we create a xml tree in memory
 */
private void createDocument() {
    //get an instance of factory
    DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
    try {
        //get an instance of builder
        DocumentBuilder db = dbf.newDocumentBuilder();

        //create an instance of DOM
        dom = db.newDocument();
    } catch (ParserConfigurationException pce) {
        //dump it
        System.out.println("Error while trying to instantiate DocumentBuilder " + pce);
        System.exit(1);
    }
}

```

c) Create the root element Books.

```

/**
 * The real workhorse which creates the XML structure
 */
private void createDOMTree(){
    //create the root element
    Element rootEle = dom.createElement("Books");
    dom.appendChild(rootEle);

    //No enhanced for
    Iterator it = myData.iterator();
    while(it.hasNext()) {
        Book b = (Book)it.next();
    }
}

```

```

//For each Book object create element and attach it to root
Element bookEle = createBookElement(b);
rootEle.appendChild(bookEle);
}

}

```

d) Creating a book element.

```

/**
 * Helper method which creates a XML element
 * @param b The book for which we need to create an xml representation
 * @return XML element snippet representing a book
 */
private Element createBookElement(Book b){
    Element bookEle = dom.createElement("Book");
    bookEle.setAttribute("Subject", b.getSubject());

    //create author element and author text node and attach it to bookElement
    Element authEle = dom.createElement("Author");
    Text authText = dom.createTextNode(b.getAuthor());
    authEle.appendChild(authText);
    bookEle.appendChild(authEle);

    //create title element and title text node and attach it to bookElement
    Element titleEle = dom.createElement("Title");
    Text titleText = dom.createTextNode(b.getTitle());
    titleEle.appendChild(titleText);
    bookEle.appendChild(titleEle);

    return bookEle;
}

```

e) Serialize DOM to FileOutputStream to generate the xml file "book.xml".

```

/**
 * This method uses Xerces specific classes
 * prints the XML document to file.
 */
private void printToFile(){
    try
    {
        //print
        OutputFormat format = new OutputFormat(dom);
        format.setIndenting(true);

        //to generate output to console use this serializer
        //XMLSerializer serializer = new XMLSerializer(System.out, format);

        //to generate a file output use fileoutputstream instead of system.out
        XMLSerializer serializer = new XMLSerializer(
            new FileOutputStream(new File("book.xml")), format);

        serializer.serialize(dom);

    } catch(IOException ie) {
        ie.printStackTrace();
    }
}

```

Note:

The Xerces internal classes `OutputFormat` and `XMLSerializer` are in different packages.

In JDK 1.5 with built in Xerces parser they are under

```

com.sun.org.apache.xml.internal.serialize.OutputFormat
com.sun.org.apache.xml.internal.serialize.XMLSerializer

```

In Xerces 2.7.1 which we are using to run these examples they are under

```

org.apache.xml.serialize.XMLSerializer
org.apache.xml.serialize.OutputFormat

```

We are using Xerces 2.7.1 with JDK 1.4 and JDK 1.3 as the default parser with JDK 1.4 is Crimson and there is no built in parser with JDK 1.3.

Also please remember it is not advisable to use parser implementation specific classes like `OutputFormat` and `XMLSerializer` as they are only available in Xerces and if you switch to another parser in the future you may have

to rewrite.

See [below for steps](#) to make your code JAXP compliant.

[Top](#)

7 Instructions to run these programs

The instructions to compile and run these programs varies based on the JDK that you are using. This is due to the way the XML parser is bundled with various Java distributions. These instructions are for Windows OS. For Unix or Linux OS you just need to change the folder paths accordingly.

7.1 Using JDK 1.5

Xerces parser is bundled with the JDK 1.5 distribution. So you need not download the parser separately.

Running DOMParserExample

1. Download [DomParserExample.java](#), [Employee.java](#), [employees.xml](#) to c:\xercesTest
2. Go to command prompt and type
cd c:\xercesTest
3. To compile, type
javac -classpath . DomParserExample.java
4. To run, type
java -classpath . DomParserExample

Running SAXParserExample

1. Download [SAXParserExample.java](#), [Employee.java](#), [employees.xml](#) to c:\xercesTest
2. Go to command prompt and type
cd c:\xercesTest
3. To compile, type
javac -classpath . SAXParserExample.java
4. To run, type
java -classpath . SAXParserExample

Running XMLCreatorExample

1. Download [XMLCreatorExample.java](#), [Book.java](#) to c:\xercesTest
2. Go to command prompt and type
cd c:\xercesTest

3. To compile, type
`javac -classpath . XMLCreatorExample.java`
4. To run, type
`java -classpath . XMLCreatorExample`

7.2 Using JDK 1.4

Crimson parser is bundled by default with JDK 1.4. To override this parser we need to run our example with -Djava.endorsed.dirs option. Download Xerces 2.7.1 from [here](#). Unzip and copy the jar files to c:\xerces\lib. If it is installed in any other folder modify the instructions below accordingly.

Running DOMParserExample

1. Download [DomParserExample.java](#), [Employee.java](#), [employees.xml](#) to c:\xercesTest
2. Go to command prompt and type
`cd c:\xercesTest`
3. To compile, type
`javac -classpath .;c:\xerces\lib\xercesImpl.jar DomParserExample.java`
4. To run, type
`java -classpath .; -Djava.endorsed.dirs=c:\xerces\lib DomParserExample`

Running SAXParserExample

1. Download [SAXParserExample.java](#), [Employee.java](#), [employees.xml](#) to c:\xercesTest
2. Go to command prompt and type
`cd c:\xercesTest`
3. To compile, type
`javac -classpath .;c:\xerces\lib\xercesImpl.jar SAXParserExample.java`
4. To run, type
`java -classpath .; -Djava.endorsed.dirs=c:\xerces\lib SAXParserExample`

Running XMLCreatorExample

1. Download [XMLCreatorExample.java](#), [Book.java](#) to c:\xercesTest
2. Go to command prompt and type
`cd c:\xercesTest`
3. To compile, type
`javac -classpath .;c:\xerces\lib\xercesImpl.jar XMLCreatorExample.java`
4. To run, type
`java -classpath .; -Djava.endorsed.dirs=c:\xerces\lib XMLCreatorExample`

7.3 Using JDK 1.3

No parser is bundled by default with JDK 1.3. All we need to do is set the xerces jar files in classpath for our compilation and execution. Download xerces 2.7.1 from [here](#). Unzip and copy the jar files to c:\xerces\lib. If it is installed in any other folder modify the instructions below accordingly.

Running DOMParserExample

1. Download [DomParserExample.java](#), [Employeee.java](#), [employees.xml](#) to c:\xercesTest
2. Go to command prompt and type


```
cd c:\xercesTest
```
3. To compile, type


```
javac -classpath .;c:\xerces\lib\xercesImpl.jar;c:\xerces\lib\xml-apis.jar DomParserExample.java
```
4. To run, type


```
java -classpath .;c:\xerces\lib\xercesImpl.jar;c:\xerces\lib\xml-apis.jar DomParserExample
```

Running SAXParserExample

1. Download [SAXParserExample.java](#), [Employeee.java](#), [employees.xml](#) to c:\xercesTest
2. Go to command prompt and type


```
cd c:\xercesTest
```
3. To compile, type


```
javac -classpath .;c:\xerces\lib\xercesImpl.jar;c:\xerces\lib\xml-apis.jar SAXParserExample.java
```
4. To run, type


```
java -classpath .;c:\xerces\lib\xercesImpl.jar;c:\xerces\lib\xml-apis.jar SAXParserExample
```

Running XMLCreatorExample

1. Download [XMLCreatorExample.java](#), [Book.java](#) to c:\xercesTest
2. Go to command prompt and type


```
cd c:\xercesTest
```
3. To compile, type


```
javac -classpath .;c:\xerces\lib\xercesImpl.jar;c:\xerces\lib\xml-apis.jar XMLCreatorExample.java
```
4. To run, type


```
java -classpath .;c:\xerces\lib\xercesImpl.jar;c:\xerces\lib\xml-apis.jar XMLCreatorExample
```

[Top](#)

Misc

Alternative to XMLSerializer

It is not advisable to use parser implementation specific classes like `OutputFormat` and `XMLSerializer` as they are only available in Xerces and if you switch to another parser in the future you may have to rewrite. To be JAXP compliant use the following to classes to generate the output.

```
import javax.xml.transform.TransformerFactory;
import javax.xml.transform.dom.DOMSource;
import javax.xml.transform.stream.StreamResult;
```

The code to generate xml to a file would be something like

```
Transformer tr = TransformerFactory.newInstance().newTransformer();
tr.setOutputProperty(OutputKeys.INDENT, "yes");
tr.setOutputProperty(OutputKeys.METHOD, "xml");
tr.setOutputProperty("{http://xml.apache.org/xslt}indent-amount", "3");

//to send the output to a file
tr.transform( new DOMSource(dom), new StreamResult(new FileOutputStream("test.xml")));

//to send the output to console
//tr.transform( new DOMSource(dom), new StreamResult(System.out));
```

What is endorsed standard?

An endorsed standard is a Java™ API defined through a standards process other than the Java Community Process™ (JCP™). Because endorsed standards are defined outside the JCP, it is anticipated that such standards may be revised between releases of the Java 2 Platform. In order to take advantage of new revisions to endorsed standards, developers and software vendors may use the Endorsed Standards Override Mechanism to provide newer versions of an endorsed standard than those included in the Java 2 Platform as released by Sun Microsystems.

By default java looks for endorsed jar files under

```
<java-home>\lib\endorsed      [Microsoft Windows]
<java-home>/lib/endorsed     [Solaris or Linux]
```

This can be overridden using java system property `java.endorsed.dirs` as follows

```
java -Djava.endorsed.dirs=<folder_name>
```

For more information see [Endorsed Standards Override Mechanism](#)

Definitions

(From java.sun.com)

Attribute

A qualifier on an XML tag that provides additional information. For example, in the tag `<slide title="My Slide">`, title is an attribute, and My Slide is its value.

Tag

A piece of text that describes a unit of data, or element, in XML. A tag is surrounded by angle brackets `<Age>25</Age>`
`<Age>` is the start tag

Element

A unit of XML data, delimited by tags. An XML element can enclose other elements.

```
<Personnel>
  <Employee>..</Employee>
  <Employee>..</Employee>
</Personnel>
```

In this Personnel has two Employee elements.

CDATA

A predefined XML tag for "Character DATA" that says "don't interpret these characters", as opposed to "Parsed Character Data" (PCDATA), in which the normal rules of XML syntax apply (for example, angle brackets demarcate XML tags, tags define XML elements, etc.).

CDATA sections are typically used to show examples of XML syntax. Like this:

```
<![CDATA[ <slide>..A sample slide.</slide> ]]>
which displays as:
<slide>..A sample slide.. </slide>
```

Declaration

The very first thing in an XML document, which declares it as XML. The minimal declaration is `<?xml version="1.0"?>`. The declaration is part of the document prolog.

Data

The contents of an element, generally used when the element does not contain any subelements.

```
<Age>25</Age>
```

Here 25 is the data

DTD

Document Type Definition. An optional part of the document prolog, as specified by the XML standard. The DTD specifies constraints on the valid tags and tag sequences that can be in the document.

[Top](#)

The following segment contains the older version of the code. Please use the latest [programs found above](#).

6 Programs(Old Version)

All these Programs uses Xerces for Java Parser.

6.1 Parsing and Printing (Old Version)

This is like "hello world" to programming. The first step we should do is to learn how to parse and print a simple XML document using both DOM and SAX.

This will help you to get the basic concepts in parsing and how does DOM API differ from SAX.

6.1.1 Using DOM (Old Version)

This program [PrintUsingDOM](#) parses a XML file and prints it on the console . It uses DOM API.

This is a two stage process, first it parses the XML file and creates a tree structure in the memory. The next stage is to traverse the tree and print the contents.

a) Construction of DOM tree

```
//get an instance of DomParser
DOMParserWrapper myDOMParser = (DOMParserWrapper) new dom.wrapper.DOMParser();
//Parse the XML File which results in the creation of the Dom tree
Document doc = myDOMParser.parse(xmlFile);
```

Once the Dom tree is constructed the Walk method is called which traverses the tree and prints the content.

```
walk(doc);
```

b) Traversing the DOM tree

It does a Depth First Pre Order Traversal , for more on tree traversal techniques see here
As it goes through each node it prints it contents.

This snippet explains how to handle the element Node

```
//For an element node print the node name and also its attributes if any
case Node.ELEMENT_NODE:
{
    System.out.print('<' + node.getNodeName() );
    NamedNodeMap nnm = node.getAttributes();
    if(nnm != null )
    {
        int len = nnm.getLength() ;
        Attr attr;
        for ( int i = 0; i < len; i++ )
        {
            attr = (Attr)nnm.item(i);
            System.out.print(' ' ,
                + attr.getNodeName()
                + "=\""
                + attr.getNodeValue()
                + "\"");
        }
    }
    System.out.print('>');
    break;
}
} //end of element
```

This recurses through each node

```
//recurse
for(Node child = node.getFirstChild(); child != null; child = child.getNextSibling())
{
    walk(child);
}
```

Hence we print recursively all the contents to the console

[Top](#)

6.1.2 Using SAX (*Old Version*)

Sax is much more easier to understand and implement.

This program [PrintUsingSAX](#) parses a XML document and prints it on the console.

When a Sax parser parses a XML document and every time it encounters a tag it calls the corresponding tag handler methods

when it encounter a Start Tag it calls this method

```
public void startElement(String name, AttributeList attrs)
```

when it encounter a End Tag it calls this method

```
public void endElement(String name)
```

This program also parses a XML file and prints it on the console . It uses SAX API.

In this example, the PrintUsingSax Class extends the HandlerBase class and implements the call back methods to handle the printing

The steps involved are

Get a instance of SAX parser,

```
Parser myParser = new SAXParser();
```

Register the class which extends the HandlerBase so that it handle the call backs

```
myParser.setDocumentHandler(this);
```

Parse the XML file

```
myParser.parse(fileName);
```

Print the document as it is parsed

```
//handle start element
public void startElement(String name, AttributeList attrs)
{
    System.out.print('<' + name );
    if (attrs != null)
    {
        int len = attrs.getLength();
        for (int i = 0; i < len; i++)
```

```

{
    System.out.print(" "
    + attrs.getName(i)
    + "=\""
    + attrs.getValue(i)
    + "\"");
}
} //end of if
System.out.print('>');

} // startElement(String,AttributeList)

```

That's how we print using a SAX parser

[Top](#)

6.2 Generate XML *(Old Version)*

The previous program illustrated how to Parse an existing XML file using both SAX and DOM Parsers. But generating a XML file from scratch is a different story , for instance you might like to generate a xml file for the data extracted from the database. To keep the example simple this program [GenerateXML](#) generates a XML file from a Vector preloaded with hard coded data.

The steps involved are

- Get an instance of DOM parser
- Create a new Document
- Load the Data
- Create a DOM tree with this Data
- Print the DOM tree which will be the XML file

In detail

- Get an instance of DOM prsrer

```
DocumentBuilderFactory dbf =DocumentBuilderFactory.newInstance();
DocumentBuilder db = dbf.newDocumentBuilder();
```
- Create a new Document

```
doc = db.newDocument();
```


c) Load the Data

In this case the vector is loaded with hard coded data for simplicity.

```
private void loadData()
{
    vData = new Vector();
    //trivial example
    //data is Name$Age$sex
    vData.addElement("John$35$M");
    vData.addElement("Jessie$30$F");
    vData.addElement("Jonathan$28$M");
    vData.addElement("Laila$20$F");
}
```

d) Create a DOM tree with this Data

```
root = doc.createElement("Employee");
root.appendChild(doc.createTextNode("\n"));

//add name Element
item = doc.createElement("Name");
item.appendChild(doc.createTextNode(name));
root.appendChild(item);
root.appendChild(doc.createTextNode("\n"));

//add Age Element
item = doc.createElement("Age");
item.appendChild(doc.createTextNode(age));
root.appendChild(item);
root.appendChild(doc.createTextNode("\n"));

//add Sex Element
item = doc.createElement("Sex");
item.appendChild(doc.createTextNode(sex));
root.appendChild(item);
root.appendChild(doc.createTextNode("\n"));

//add to the Top Element
main.appendChild(root);
```

```
main.appendChild(doc.createTextNode("\n"));
```

This is nothing but a creating a tree structure,
 Create a Root Node Employee
 Append Child nodes Name, Age and Sex to the Root Node
 Append Root Node to the Top Node Personal

This is similar to this

```
<Personnel>
  <Employee>
    <Name>John</Name>
    <Age>35</Age>
    <Sex> M </Sex>
  </Employee>
</Personnel>
```

e) Print the DOM tree

This is similar to the printing done in the previous program PrintUsingDOM.java

[Top](#)

7 Instructions to run these programs (Old Version)

If you wish you can download [Personnel.xml](#) and [Personnel.dtd](#) Personnel.dtd to the same directory where you have downloaded these programs.

Requirements

JDK of course

Xerces for Java Parser

Assumption

Xerces is installed in c:\xerces-1_3_0 if not change the path accordingly in these instructions

[PrintUsingDOM.java](#)

To compile

```
set CLASSPATH=c:\xerces-1_3_0\xerces.jar;c:\xerces-1_3_0\xercesSamples.jar;.;%classpath%;
```

```
javac PrintUsingDOM.java
```

To run

```
java PrintUsingDOM xmlFileName
```

or if you have downloaded the Personnel.xml and Personnel.DTD

```
java PrintUsingDOM Personnel.xml
```

[PrintUsingSAX](#)

To compile

```
set CLASSPATH=c:\xerces-1_3_0\xerces.jar;c:\xerces-1_3_0\xercesSamples.jar;.%classpath%;  
javac PrintUsingSAX.java
```

To run

```
java PrintUsingSAX xmlFileName
```

or if you have downloaded the sample xml file and DTD

```
java PrintUsingSAX Personnel.xml
```

[GenerateXML.java](#)

To compile

```
set CLASSPATH=c:\xerces-1_3_0\xerces.jar;c:\xerces-1_3_0\xercesSamples.jar;.%classpath%;  
javac GenerateXML.java
```

To run

```
java GenerateXML
```