UNIT-2

XML: Introduction to XML, Defining XML tags, their attributes and values, Document Type Definition, XML Schemas, Document Object Model, XHTML.

Parsing XML Data – DOM and SAX Parsers in Java.

Introduction to XML

XML is a software- and hardware-independent tool for storing and transporting data.

What is XML?

•XML stands for EXtensible Markup Language

•XML is a markup language much like HTML

•XML was designed to store and transport data

•XML was designed to be self-descriptive

Introduction to XML

Advantages of XML?

•XML document is human readable and we can edit any XML document in simple text editors.

•The XML document is language neutral. That means a Java program can generate an XML document and this document can be parsed by Perl.

•Every XML document has a tree structure. Hence complex data can be arranged systematically and can be understood in simple manner.

•XML files are independent of an Operating System.

Introduction to XML

Features of XML?

•XML is Extensible Markup Language intended for transport or storage of data.

•The most important feature is that user is able to define and use his own tags. This allows us to restrict the use of the set of tags defined by proprietary vendors.

•XML contains only data and does not contain any formatting information. Hence document developers can decide how to display the data.

•XML permits the document writers to create the tags of any type of information.

•Searching, sorting, rendering or manipulating XML document is possible using extended style sheet language (XSL).

•XML is not at all vendor specific or browser specific.

Introduction to XML

Basic XML Document

•In XML we can create our own tags.

Example: first.xml

```
<Person>
        <Personal-Info>
                <name>My name is Swathi</name>
                <city>Hyderabad</city>
        </Personal-info>
        <Hobby>
                <first>Reading</first>
                <second>Programming</second>
                <third>Movies</third>
        </Hobby>
</Person>
```

The XML scripts are self descriptive

Introduction to XML

Rules that must be followed while writing XML-

1. XML is case sensitive.
2. In XML each start tag must have matching end tag.
3. The elements in XML must be properly nested.
4. The syntax of writing comments in XML is similar to HTML comments.

```
<!– comment line -->
<hello>
<!--
<body>Welcome</body>
-->
</hello>
```

5. XML does not truncate multiple white-spaces (HTML truncates multiple white-spaces to one single white-space)

Defining XML tags, their attributes and values

An XML document contains XML Elements.

An XML element is everything from (including) the element's start tag to (including) the element's end tag.

```
<price>29.99</price>
```

An element can contain:

- text
- attributes
- other elements
- or a mix of the above

Defining XML tags, their attributes and values

Empty XML Elements

An element with no content is said to be empty.

Eg: <element></element>

We can also use a so called self-closing tag:

<element />

XML Naming Rules

XML elements must follow these naming rules:

- Element names are case-sensitive
- Element names must start with a letter or underscore
- Element names cannot start with the letters xml (or XML, or Xml, etc)
- Element names can contain letters, digits, hyphens, underscores, and periods
- Element names cannot contain spaces
- Any name can be used, no words are reserved (except xml).

Defining XML tags, their attributes and values

XML Attributes

XML elements can have attributes, just like HTML.

Attributes are designed to contain data related to a specific element.

Attribute values must always be quoted. Either single or double quotes can be used.

Eg: <person gender="female"> or <person gender=‘female’>

XML Elements vs. Attributes

```
<person gender="female">
  <firstname>Anna</firstname>
  <lastname>Smith</lastname>
</person>
```

```
<person>
  <gender>female</gender>
  <firstname>Anna</firstname>
  <lastname>Smith</lastname>
</person>
```

Defining XML tags, their attributes and values

XML Namespace

XML Namespaces provide a method to avoid element name conflicts.

In XML, element names are defined by the developer. This often results in a conflict when trying to mix XML documents from different XML applications.

```
<table>
  <tr>
    <td>Apples</td>
    <td>Bananas</td>
  </tr>
</table>
```

```
<table>
  <name>African Coffee Table</name>
  <width>80</width>
  <length>120</length>
</table>
```

If these XML fragments were added together, there would be a name conflict. Both contain a <table> element, but the elements have different content and meaning.

A user or an XML application will not know how to handle these differences.

Defining XML tags, their attributes and values

Name conflicts in XML can easily be avoided using a name prefix**.**

```
<h:table>
  <h:tr>
    <h:td>Apples</h:td>
    <h:td>Bananas</h:td>
  </h:tr>
</h:table>

<f:table>
  <f:name>African Coffee Table</f:name>
  <f:width>80</f:width>
  <f:length>120</f:length>
</f:table>
```

Document Type Definition

- An XML document with correct syntax is called "Well Formed".
- An XML document validated against a DTD is both "Well Formed" and "Valid".
- The document type definition is used to define the basic building block of any xml document. Using DTD we can specify the various elements types, attributes and their relationship with one another.
- DTD is used to specify the set of rules for structuring data in any XML file.
- Various building blocks of XML are-
- 1. Elements – Used for defining tags.
- 2. Attribute-Used to specify the values of element.
- 3. CDATA-Character data, parsed by the parser.
- 4. PCDATA-Parsed Character Data (i.e. text)

Document Type Definition

Types of DTD

1. Internal DTD

```
<?xml version = “1.0” encoding=“UTF-8”?>
<!DOCTYPE student [
<!ELEMENT student (name,address,std,marks)>
<!ELEMENT name(#PCDATA)>
<!ELEMENT address (#PCDATA)>
<!ELEMENT std(#PCDATA)>
<!ELEMENT marks(#PCDATA)>
]>
<student>
		<name> Anand</name>
		<address>Hyderabad</address>
		<std>Second</std>
		<marks>70 percent</marks>
</student>
```

Document Type Definition

Types of DTD

2. External DTD (student.dtd)

```
<!ELEMENT student (name,address,std,marks)>

<!ELEMENT name(#PCDATA)>

<!ELEMENT address (#PCDATA)>

<!ELEMENT std(#PCDATA)>

<!ELEMENT marks(#PCDATA)>
```

DTDDemo.xml

```
<?xml version = “1.0”?>
<!DOCTYPE student SYSTEM “student.dtd”>

<student>
            <name> Anand</name>
            <address>Hyderabad</address>
            <std>Second</std>
            <marks>70 percent</marks>
</student>
```

Document Type Definition

Merits of DTD

1. DTDs are used to define the structural components of XML document.
2. These are relatively simple and compact.
3. DTDs can be defined inline and hence can be embedded directly in the XML document.

Demerits of DTD

1. The DTDs are very basic and cannot be much specific for complex documents.
2. The language that DTD uses is not an XML document. Hence various frameworks used by XML cannot be supported by the dTDs.
3. The DTD cannot define the type of data contained within the XML document. Hence using DTD we cannot specify whether the element is numeric or string or of data type.
4. There are some XML processor which do not understand DTDs.
5. The DTDs are not aware of namespace concept.

XML Schemas

An XML Schema describes the structure of an XML document.
The XML Schema language is also referred to as XML Schema Definition (XSD).
The XML Schema became the World Wide Web Consortium (W3C) recommendation in 2001.

The purpose of an XML Schema is to define the legal building blocks of an XML document:

- the elements and attributes that can appear in a document
- the number of (and order of) child elements
- data types for elements and attributes
- default and fixed values for elements and attributes

XML Schemas Support Data Types
One of the greatest strength of XML Schemas is the support for data types.

- It is easier to describe allowable document content
- It is easier to validate the correctness of data
- It is easier to define data facets (restrictions on data)
- It is easier to define data patterns (data formats)
- It is easier to convert data between different data types

XML Schemas

Example:

```
<?xml version="1.0"?>
<xs:schema xmlns:xs=http://www.w3.org/2001/XMLSchema>
        <xs:element name="Student">
                <xs:complexType>
                        <xs:sequence>
                                <xs:element name="name" type="xs.string"/>
                                <xs:element name="address" type="xs.string"/>
                                <xs:element name="std" type="xs.string"/>
                                <xs:element name="marks" type="xs.string"/>
                        </xs:sequence>
                </xs:complexType>
        </xs:element>
</xs:schema>
```

xs – to identify the schema elements and types.
xs:schema – root element. It takes the attribute xmlns:xs which has the value came from the namespace http://www.w3.org/2001/XMLSchema
Student is **complex type** because it contains other elements.

XML Schemas

SimpleXml.xml

```
<?xml version="1.0" encoding="UTF-8">?
<Student>
		<name>Anand</name>
		<address>Hyderabad</address>
		<std>Tenth</std>
		<marks>90 percent</marks>
</Student>
```

student.dtd

```
<!ELEMENT
Student(name,address,std,marks)>
<!ELEMENT name(#PCDATA)>
<!ELEMENT address(#PCDATA)>
<!ELEMENT std(#PCDATA)>
<!ELEMENT marks(#PCDATA)>
```

DTDDemo.xml

```
<?xml version="1.0"?>
<!DOCTYPE Student SYSTEM "student.dtd">
<Student>
		<name>Anand</name>
		<address>Hyderabad</address>
		<std>Tenth</std>
		<marks>90 percent</marks>
</Student>
```

XML Schemas

StudentSchema.xsd

```
<?xml version="1.0"?>
<xs:schema xmlns:xs=http://www.w3.org/2001/XMLSchema>
		<xs:element name="Student">
				<xs:complexType>
						<xs:sequence>
								<xs:element name="name" type="xs.string"/>
								<xs:element name="address" type="xs.string"/>
								<xs:element name="std" type="xs.string"/>
								<xs:element name="marks" type="xs.string"/>
						</xs:sequence>
				</xs:complexType>
		</xs:element>
</xs:schema>
```

MySchema.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<Student xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="StudentSchema.xsd">

		<name>Anand</name>
		<address>Hyderabad</address>
		<std>Tenth</std>
		<marks>90 percent</marks>
</Student>
```

XML Schemas

XML Schema has a lot of built-in data types. The most common types are:

- xs:string
- xs:decimal
- xs:integer
- xs:boolean
- xs:date
- xs:time

XML Schemas

Advantages of Schema:

1. Schemas are more specific.
2. The schema provide the support for data types.
3. The schema is aware of namespace.
4. It is written in XML itself and has a large number of built in and derived types.
5. The XML schema is W3C recommendation, hence it is supported by cvarious XML validator and XML processors.

Disadvantages of Schema:

1. The XML schema is complex to design and hard to learn
2. The XML document cannot be if the corresponding schema file is absent.
3. Maintaining the schema for large and complex operations sometimes slows down the processing of XML document.

XML Schemas

Advantages of Schema over DTD:

1. Both the schemas and DTDs are useful for defining structural components of XML. But the DTDs are basic and cannot be much specific for complex operations. On the other hand schemas are more specific.
2. The schemas provide support for defining the type of data. The DTDs do not have this ability. Hence content definition is possible using schema.
3. The schemas are namespace aware and DTDs are not.
4. The XML schema is written in XML itself and has a large number of built in and derived types.
5. The schema is the W3C recommendation. Hence it is supported by various XML validator and XML processors but there are some XML processors which do not support DTD.
6. Large number of web applications can be built using XML schema. On the other hand relatively simple and compact operations can be built using DTD.

Document Object Model (DOM)

The document object modeling is for defining the standard for accessing and manipulating XML.

It is W3 recommendation for handling the structured documents.

DOM provides standard set of programming interfaces for working with XML and HTML.

Document Object Model (DOM) is a set of platform independent and language neutral application programming interface (API) which describes how to access and manipulate the information stored in XML or in HTML documents.

XML DOM is for

loading the XML document.

accessing the elements of XML document

deleting the elements of XML document

changing the elements of XML document

Document Object Model (DOM)

A DOM Document is a collection of nodes or pieces of information organized in a hierarchy. This hierarchy allows a developer to navigate through the tree looking for specific information. Because it is based on a hierarchy of information, the DOM is said to be tree based.

The XML DOM, on the other hand, also provides an API that allows a developer to add, edit, move, or remove nodes in the tree at any point in order to create an application.

Document Object Model (DOM)

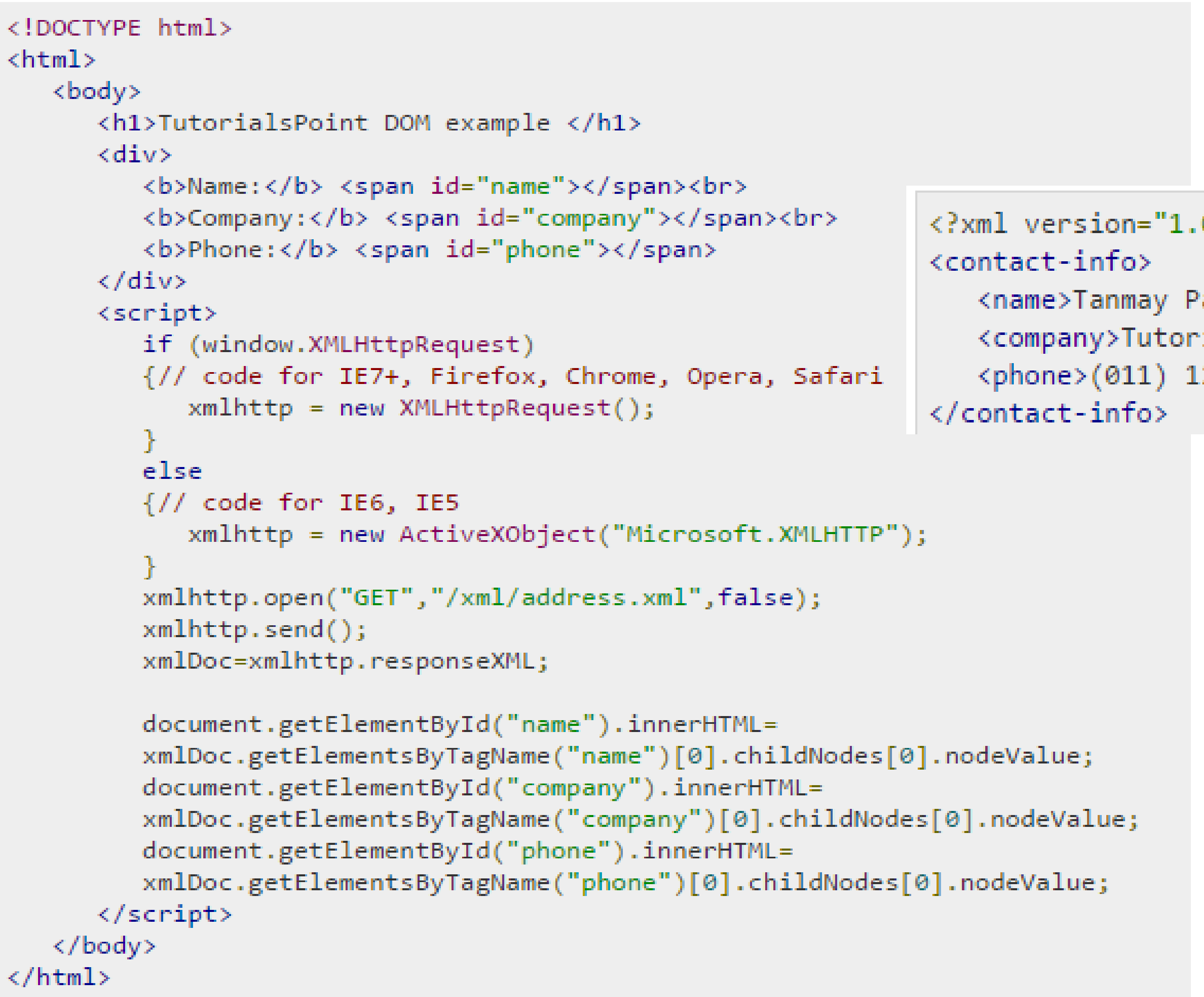
1. **Loading an XML file:**

```
<html>
<!—Simple DOM example for loading xml file-->
<body>
<script type="text/javascript">
try
{
xmlDocument=new ActiveXObject("Microsoft.XMLDOM");
}
catch(e)
{
            try
            {
                        xmlDocument=document.implementation.createDocument("","",null);
            }
            catch(e)
            { alert(e.message)}
}
try
{
            xmlDocument.async=false;
            xmlDocument.load("student.xml");
            document.write("XML Document student.xml is loaded");
}
catch(e){alert(e.message)}
            </script>
</body>
</html>
```

student.xml

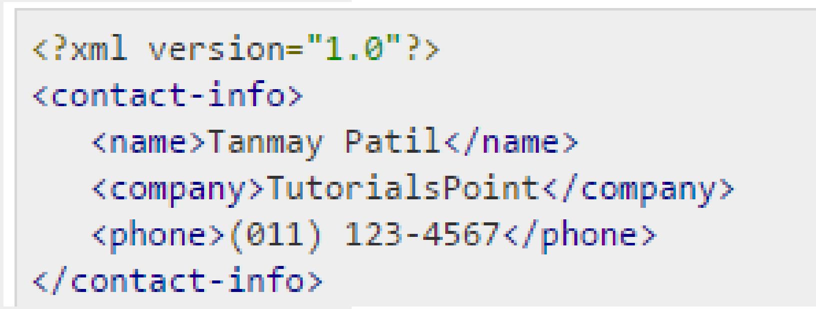
```
<?xml version="1.0">?
<Student>
            <Roll_No>10</Roll_No>
            <Personal_Info>
                        <Name>Anand</Name>
                        <Address>Hyderabad</Address>
                        <Phone>8978903739</Phone>
            </Personal_Info>
            <Class>B.Tech</Class>
            <Subject>WT</Subject>
            <Marks>100</Marks>
</Student>
```

Document Object Model (DOM)



```
<!DOCTYPE html>
<html>
   <body>
      <h1>TutorialsPoint DOM example </h1>
      <div>
         <b>Name:</b> <span id="name"></span><br>
         <b>Company:</b> <span id="company"></span><br>
         <b>Phone:</b> <span id="phone"></span>
      </div>
      <script>
         if (window.XMLHttpRequest)
         {// code for IE7+, Firefox, Chrome, Opera, Safari
            xmlhttp = new XMLHttpRequest();
         }
         else
         {// code for IE6, IE5
            xmlhttp = new ActiveXObject("Microsoft.XMLHTTP");
         }
         xmlhttp.open("GET","/xml/address.xml",false);
         xmlhttp.send();
         xmlDoc=xmlhttp.responseXML;

         document.getElementById("name").innerHTML=
         xmlDoc.getElementsByTagName("name")[0].childNodes[0].nodeValue;
         document.getElementById("company").innerHTML=
         xmlDoc.getElementsByTagName("company")[0].childNodes[0].nodeValue;
         document.getElementById("phone").innerHTML=
         xmlDoc.getElementsByTagName("phone")[0].childNodes[0].nodeValue;
      </script>
   </body>
</html>
```



```
<?xml version="1.0"?>
<contact-info>
   <name>Tanmay Patil</name>
   <company>TutorialsPoint</company>
   <phone>(011) 123-4567</phone>
</contact-info>
```

XHTML

- XHTML is HTML written as XML.
- XHTML stands for EXtensible Hypertext Markup Language
- XHTML is almost identical to HTML
- XHTML is stricter than HTML
- XHTML is HTML defined as an XML application
- XHTML is supported by all major browsers

XHTML

The Most Important Differences from HTML:

Document Structure

XHTML DOCTYPE is mandatory
The xmlns attribute in <html> is mandatory
<html>, <head>, <title>, and <body> are mandatory

XHTML Elements

XHTML elements must be properly nested
XHTML elements must always be closed
XHTML elements must be in lowercase
XHTML documents must have one root element

XHTML Attributes
Attribute names must be in lower case
Attribute values must be quoted
Attribute minimization is forbidden

XHTML

Example:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

        <head>
                <title>Title of document</title>
        </head>

        <body>
                some content
        </body>

</html>
```

XHTML

XHTML Elements Must Be Properly Nested

```
<b><i>This text is bold and italic</i></b>
```

XHTML Elements Must Always Be Closed

```
<p>This is a paragraph</p>
<p>This is another paragraph</p>
```

Empty Elements Must Also Be Closed

```
A break: <br />
A horizontal rule: <hr />
An image: <img src="happy.gif" alt="Happy face" />
```

XHTML

XHTML Elements Must Be In Lower Case

```
<body>
<p>This is a paragraph</p>
</body>
```

XHTML Attribute Names Must Be In Lower Case

```
<table width="100%">
```

Attribute Values Must Be Quoted

```
<table width="100%">
```

XHTML

How to Convert from HTML to XHTML

- Add an XHTML <!DOCTYPE> to the first line of every page
- Add an xmlns attribute to the html element of every page
- Change all element names to lowercase
- Close all empty elements
- Change all attribute names to lowercase
- Quote all attribute values

Parsing XML Data

DOM and SAX Parsers

The primary goal of any XML processor is to parse the given XML document. Java has a rich source of in-built APIs for parsing the given XML document. It is parse in two ways-

•Tree based parsing

•Event based parsing

DOM is used to parse the XML document using the tree structure. We can access the information of an XML document by interacting with the tree nodes.

Simple API for XML (i.e. SAX) allows us to access the information of XML document using sequences of events. Thus there are two methods of parsing the XML document:

•Parsing using DOM (tree based)

•Parsing using SAX(event based)

Parsing XML Data

DOM	SAX
DOM is a tree based parsing method used to parse the given XML document.	SAX is an event based parsing method used to parse the given XML document.
In this method, the entire XML document is stored in the memory before actual processing. Hence it requires more memory.	In this method, the parsing is done by generating the sequence of events or it calls handler functions.
The DOM approach is useful for smaller applications because it is simpler to use but it is certainly not used for larger XML documents because it will then require larger amount of memory.	Although SAX development is much more challenging, it is useful for parsing the large XML document because the approach is event based, xml gets parsed node by node and does not require large amount of memory.
We can insert or delete a node.	We can insert or delete a node.
Traversing is done in any direction in DOM approach.	Top to bottom traversing is done in this approach.

Parsing XML Data

Using DOM API

In Java JDK, two built-in XML parsers are available – DOM and SAX, both have their pros and cons.

The DOM is the easiest to use Java XML Parser. It parses an entire XML document and load it into memory, modeling it with Object for easy nodel traversal. DOM Parser is slow and consume a lot memory if it load a XML document which contains a lot of data.

1. Read a XML file   http://www.mkyong.com/tutorials/java-xml-tutorials/

2. Modify existing XML file

3. Create a new XML file

4. Count XML Elements

Parsing XML Data

SAX XML Parser

SAX parser is work differently with DOM parser, it does not load any XML document into memory and create some object representation of the XML document. Instead, the SAX parser use callback function (org.xml.sax.helpers.DefaultHandler) to informs clients of the XML document structure.