UNIT-II
XML

* INTRODUCTION TO XML

what is XML:

- XML stands for Extensible Markup Language.
- XML is a markup language much like HTML.
- XML was designed to carry data, not to display data.
- XML tags are not predefined. You must define your own tags.
- XML is designed to be self-descriptive.
- XML is W3C Recommendation.

Representing Web Data : XML

XML stands for exentensible Markup Language, developed by W3C in 1996. XML 1.0 was officially adopted as a W3C recommendation in 1998. XML was designed to carry data, not to display data. XML is designed to be self-descriptive. XML is a subset of SGML than can be defined in your own tags. A meta Language and tags describe the content. XML supports CSS, XSL, DOM.

Advantages:

* XML is a simple scripting language whereas humans can easily read.

* XML document is language natural that means one language programming code can generate an XML document and these

documents can be passed by other languages.

Goals of XML :

* The user must be able to define and use his own tags.
* Allows the user to build his own tag library, based on his web requirement.
* Allow user to define the formatting rules for the user defined tags.
* XML must support storage or transport of data.

* DEFINING XML TAGS, THEIR ATTRIBUTES AND VALUES

Tags and Elements:

An XML file is structured by several XML-elements, also called XML-nodes or XML-tags. XML-elements names are enclosed by triangular brackets < > as shown below.

```
<element>
```

Syntax Rules for Tags and Elements

Element Syntax: Each XML-element needs to be closed either with start or with end elements as shown below:

```
<element> ... </element>
```

or in simple-cases, just this way:

```
<element/>
```

Nesting of elements:

An XML-element can contain multiple XML-elements as its children, but the children elements must not overlap. i.e., an end tag of an element must have the same name as that of the most recent unmatched start tag.

Following example shows incorrect nested tags.

```
<?xml version = "1.0" ?>
<contact-info>
   <company> IARE
</contact-info>
   </company>
```

Following example shows correct nested tags:

```
<?xml version = "1.0" ?>
<contact-info>
<company> IARE </company>
</contact-info>
```

Let us learn about one of the most important part of XML, the XML tags. XML tags form the foundation of XML. They define the scope of an element in the XML. They can also be used to insert comments, declare settings required for parsing the environment and to insert special instructions.

* We can broadly categorize XML tags as follows:

Start Tag:

The begining of every non-empty XML element is marked by a start-tag. An example of start-tag is:

```
<address>
```

End Tag:

Every element that has a start tag should end with an end-tag. An example of end-tag is:

```
</address>
```

* Note that the end tags include a solidus ("/") before the name of an element.

Empty Tag:

The text that appears between start-tag and end-tag is called content. An element which has no content is termed as empty. An empty element can be represented in two ways as below:

(1) A start tag immediately followed by an end-tag as shown below:

```
<hr></hr>
```

(2) A complete empty-element tag is as shown below:

```
<hr/>
```

* Empty-element tags may be used for any element which has no content.

XML Tags Rules:

Following are the rules that need to be followed to use XML tags:

Rule 1:

XML tags are case-sensitive. Following line of code is an example of wrong syntax </Address>,

because of the case difference in two tags, which is treated as erroneous syntax in XML.

```
<address> This is wrong syntax </Address>
```

Following code shows a correct way, where we use the same case to name the start and the end tag.

```
<address> This is correct syntax </address>
```

Rule 2:

XML tags must be closed in an appropriate order, i.e., an XML tag opened inside another element must be closed before the outer element is closed. For example:

```
<outer_element>
   <internal_element>
        This tag is closed before the outer_element
   </internal_element>
</outer_element>
```

XML elements

XML elements can be defined as building blocks of an XML. Elements can behave as containers to hold text, elements, attributes, media objects or all of these.

* Each XML document contains one or more elements, the scope of which are either delimited by start and end tags, or for empty elements, by an empty-element tag.

Syntax:

Following is the syntax to write an XML element:

```
<element-name attribute1 attribute2>

    .... content

</element-name>
```

where,

- element-name is the name of the element. The name its case in the start and end tags must match

- attribute1, attribute2 are attributes of the element seperated by white spaces. An attribute defines a property of the element. It associates a name with a value, which is a string of characters. An attribute is written as:

        name = "value"

The name is followed by an = sign and a string value inside double (" ") or single (' ') quotes.

Empty Element:

An empty element (element with no content) has following syntax:

```
<name attribute1 attribute2 .../>
```

Example of an XML document using various XML element:

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

<address/>
</contact-info>
```

XML Elements Rules:

Following rules are required to be followed for XML elements.

- An element name can contain any alphanumeric characters. The only punctuation marks allowed in names are the hyphen(-), underscore (_) and period (.).

- Names are case-sensitive. For example, Address, address and ADDRESS are different names.
- Start and end tags of an element must be identical.
- An element, which is a container, can contain text or elements as seen in the above example.

Root element :

An XML document can have only one root element. For example, following is not a correct XML document, because both the x and y elements occur at the top level without a root element.

```
<x> ... </x>
<y> ... </y>
```

The following example shows a correctly formed XML document:

```
<root>
<x> ... </x>
<y> ... </y>
</root>
```

Document Type Definition (DTD):

DTD is an XML technique used to define the structure of a XML document.

* DTD is a text based document with the extension of .dtd.

* A DTD defines the structure and the legal elements and attributes of an XML document

DTD - XML Building Blocks:

The main building blocks of both XML and HTML documents are elements.

* Seen from a DTD point of view, all XML documents are made up by the following building blocks:

a) Elements
b) Attributes
c) Entities
d) PCDATA
e) CDATA

a) Elements:

Declaring Elements

In a DTD, XML elements are declared with the following syntax:

```
<!ELEMENT element-name category>
             or
<!ELEMENT element-name (element-content)>
```

Elements with Parsed character Data

Elements with only parsed character data are declared with #PCDATA inside parentheses:

```
<!ELEMENT element-name (#PCDATA)>
```

Example :

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

Elements with Children (sequences)

Elements with one or more children are declared with the name of the children elements inside parentheses:

```
<!ELEMENT element-name (child1)>
        or
<!ELEMENT element-name (child1, child2..)>
```

Example :

```
<!ELEMENT student (branch, rno, name, marks)>

<!ELEMENT student (branch, rno, name, marks)>
<!ELEMENT branch (#PCDATA)>
<!ELEMENT rno (#PCDATA)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT marks (#PCDATA)>
```

b) Attributes :

Declaring Attributes

```
<!ATTLIST element-name attribute-name
                    attribute-type attribute-value>
```

DTD example :

```
<!ATTLIST payment type CDATA "check">
```

XML example :

```
<payment type="check"/>
```

A Default Attribute Value

```
DTD:        <!ELEMENT square EMPTY>
            <!ATTLIST square width CDATA "0">

Valid XML:  <square width="100"/>
```

#REQUIRED

```
DTD:        <!ATTLIST person number CDATA
                                        #REQUIRED>

Valid XML:  <person number="5677"/>
Invalid XML:  <person/>
```

#IMPLIED:

```
DTD:        <!ATTLIST contact fax CDATA #IMPLIED>

Valid XML:  <contact fax="555-667788"/>
Valid XML:  <contact/>
```

#FIXED:

```
DTD:        <!ATTLIST college name CDATA #FIXED
                                        "NNRG">

Valid XML:  <college name="NNRG"/>
Invalid XML:  <college name="JNTUH"/>
```

Enumerated Attribute Values

```
DTD:   <!ATTLIST payment type (check|cash)
                                        "cash">
```

XML example : <payment type = "check" />

or <payment type = "cash" />

Example :

```
//abc.dtd
<!ELEMENT student (branch, rno, name, marks)>
<!ELEMENT branch (#PCDATA)>
<!ELEMENT rno (#PCDATA)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT marks (#PCDATA)>
<!ATTLIST branch dept CDATA #REQUIRED>

//abc.xml
<?xml version = "1.0"?>
<!DOCTYPE student SYSTEM "abc.dtd">
<student>
      <branch dept = "CSE">
      <rno> 501 </rno>
      <name> naveen </name>
      <marks> 65 </marks>
      </branch>
</student>
```

Linking DTD to XML :

DTD declarations either internal XML document or make external DTD file, after linked to a XML document.

* Internal DTD You can write rules inside XML document using <!DOCTYPE ..> declaration. Scope of this DTD within this document. Advantages is document Validated by itself without external reference.

* External DTD You can write rules in a seperate file (with .dtd extension). later this file linked to a XML document. This way you can linked several XML documents refer same DTD rules.

Internal DTD :

Internal DTD you can declare inside your XML file. In XML file top <!DOCTYPE...> declaration to declare the DTD.

```
<?xml version = "1.0" standalone = "yes" ?>
<!DOCTYPE root_element [
   ...
   ...
]
```

Following internal DTD example define root element <student> and other element are second level element along with discipline attribute.

* DTD rules must be placed specifies top of the XML element (root element) in the document.

```
<?xml version = "1.0" ?>
<!DOCTYPE root_element [
        <!ELEMENT student (branch, rno, name,
                                         marks)>
        <!ELEMENT branch (#PCDATA)>
        <!ELEMENT rno (#PCDATA)>
        <!ELEMENT name (#PCDATA)>
        <!ELEMENT marks (#PCDATA)>
        <!ATTLIST branch dept CDATA
                                 #REQUIRED>]

<student>
<branch dept = "CSE">
<rno> 501 </rno>
<name> naveen </name>
<marks> 65 </marks>
</branch>
</student>
```

External DTD:

External DTD are shared between multiple XML documents. Any changes are update in DTD document effect or updated come to a all XML documents

* External DTD are of two types.

a) Private DTD
b) Public DTD

a) Private DTD:

Private DTD identify by the SYSTEM keyword. Access for single or group of users.

* You can specify the rules in the external DTD file with .dtd extension. Later in XML file <!DOCTYPE ..> declaration is present to link the DTD file

Syntax:

```
<!DOCTYPE root-element SYSTEM "dtd-file-
                               location">
```

Example:

```
<!DOCTYPE root-element SYSTEM "abc-dtd">
```

b) Public DTD:

Public DTD identify by the PUBLIC keyword. Access any users and our XML editor are known to DTD

* Some common DTD : webDTD , XHTML, MathML etc.

Syntax:

```
<!DOCTYPE root-element PUBLIC "dtd-name"
                    "dtd-file-location">
```

Example:

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

XML Schema :

XML schema is commonly known as XML Schema Definition (XSD). It is used to describe and validate the structure and the content of XML data. XML schema defines the elements, attributes and data types. Schema element supports Namespaces. It is similar to database schema that describes the data in a database.

Syntax :

You need to declare a schema in your XML document as follows –

```
<xs:schema>
  — —
  — —
</xs:schema>
```

Example :

The following example shows how to use schema

```
<?xml version = "1.0" encoding = "UTF-8" ?>
<xs:schema xmlns:xs = "http://www.w3.org/2001/
                                XMLSchema">
 <xs:element name = "contact">
  <xs:complexType>
   <xs:sequence>
    <xs:element name = "name" type = "xs:string"/>
    <xs:element name = "company" type = "xs:string"/>
    <xs:element name = "phone" type = "xs:int"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
</xs:schema>
```

The basic idea behind XML schemas is that they describe the legitimate format that an XML document can take.

Elements :

As we saw in XML - Elements chapter, elements are the building blocks of XML document. An element can be defined within an XSD as follows –

```
<xs:element name="x" type="y"/>
```

Definition Types

You can define XML schema elements in the following ways –

Simple Type :

Simple type element is used only in the context of the text. Some of the predefined simple types are :

xs:integer, xs: boolean, xs: string, xs: date.

For example –

```
<xs: element name = "phone_number" type ="xs:int"/>
```

Complex Type :

A complex type is a container for other element definitions. This allows you to specify which child elements an element can contain and to provide some structure within your XML documents.

For example :

```
<xs:element name = "Address">
  <xs:complexType>
    <xs:sequence>
      <xs:element name = "name" type = "xs:string"/>
      <xs:element name = "company" type = "xs:string"/>
      <xs:element name = "phone" type = "xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

In the above example, Address element consists of child elements. This is a container for other <xs:element> definitions, that allows to build a simple hierarchy of elements in the XML document.

Global Type :

With the global type, you can define a single type in your document, which can be used by all other references. For example, suppose you want to generalize the person and company for different address of the company. In such case, you can define a general type as follows –

```
<xs:element name = "AddressType">
  <xs:complexType>
    <xs:sequence>
      <xs:element name = "name" type = "xs:string"/>
      <xs:element name = "company" type = "xs:string"/>
    </xs:sequence>
```

```
 </xs:complexType>
</xs:element>
```

Now let us use this type in our example as follows –

```
<xs:element name="Address1">
 <xs:complexType>
  <xs:sequence>
   <xs:element name="address" type="AddressType"/>
   <xs:element name="phone1" type="xs:int"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>

<xs:element name="Address2">
 <xs:complexType>
  <xs:sequence>
   <xs:element name="address" type="AddressType"/>
   <xs:element name="phone2" type="xs:int"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
```

Instead of having to define the name and the company twice (once for Address1 and once for Address2), we now have a single definition. This makes maintenance simpler, i.e., if you decide to add "Postcode" elements to the address, you need to add them at just one place.

Attributes:

Attributes in XSD provide extra information within an element. Attributes have name and type property as shown below -

```
<xs:attribute name="x" type="y"/>
```

DOCUMENT OBJECT MODEL

" The W3c (DOM) Document Object Model is a platform and language - netural interface that allows programs and scripts to dynamically access and update the content, structure, and style of a document".

* The HTML DOM defines a standard way for accessing and manipulating HTML documents. It presents the HTML doument as a tree-structure.

* In XML, DOM is a standard for how to get, change, add or delete XML elements.

* In XML Dom we use few properties and methods.

XML Properties

1) nodeName
2) nodeValue
3) parentNode
4) childNode
5) attributes

Methods

1) getElementsByTagName(" ")
2) appendChild(node);
3) removeChild(node);

Get the value of an XML Element

```
txt = xmlDoc.getElementsByTagName ("title") [0].
                        childNodes [0]. nodeValue ;
```

Example : xml program

```
<student>
   <Branch  Br = "CSE">
       <Rno> 501 </Rno>
       <title> xyz </title>
   </Branch>
   <Branch  Br = "ECE">
       <Rno> 401 </Rno>
       <title> abc </title>
   </Branch>
</student>
```

XML DOM nodes :

According to the XML DOM, everything in an XML document is a node.

* The entire document is a document node.
* Every XML element is an element node.
* The text in the XML elements are text nodes.
* Every attribute is an attribute node.
* Comments are comment nodes.

Insert a new node in XML :

For inserting a new element we use appendChild which insert a childnode to a specified tag.

```
x = document . append Child ("marks")
```

Example :

```
getElementByTagName ('dept')[1].childNode[0].
                                  nodeValue = 240;
```

program :

```
<college>
  <dept Branch = 'CSE'>
      <student> 120 </student>
      <faculty> 30 </faculty>
  </dept>
  <dept Branch = 'ECE'>
      <student> 115 </student>
      <faculty> 30 </faculty>
  </dept>
</college>
```

In the above example instead of 120 we get 240.

Create a new element:

```
newElement = xmlDoc.CreateElement("lab");
xmlDoc.getElementByTagName("dept")[0].
                    appendChild(newElement);
```

In Javascript we write document

In XML we write Doc

To insert value into that:

```
xmlDoc.getElementByTagName("dept")[0].
              childNode[2].nodeValue = 5;
```

Creating an attribute:

```
xmlDoc.getElementByTagName("dept")[2].
              setAttribute('Branch', 'Mech');
```

For Deleting / Removing:

```
xmlDoc.getElementByTagName("dept")[0].
         chidd Node[1].nodeValue = " ";
```

To remove child:

```
xmlDoc.getElementByTagName("dept")[0].
                 removeChild( )
```

Scanned with CamScanner

* XHTML

XHTML is HTML written as xml.

What is XHTML ?

* XHTML stands for EXtensible HyperText Markup Language.

* XHTML Is almost identical to HTML
* XHTML is supported by all major browsers

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.

<!DOCTYPE ... > is mandatory :

An XHTML document must have a XHTML DOCTYPE declaration.

* A complete list of all the XHTML Doctypes is found in our HTML Tags Reference.

* The <html>, <head>, <title> and <body> elements must also be present, and the xmlns

attribute in <html> must specify the xml namespace for the document.

* XHTML Elements Must Always Be closed

* Empty Elements Must Also be closed.

* XHTML Elements Must Be in Lower Case.

* XHTML Attribute Names Must Be in Lower Case.

* Attribute Values Must Be Quoted

XML NameSpace :

* XML namespace provides a method to avoid element name conflicts

* To avoid such type of conflicts we use a name prefix.

Example:

Instead of <table> we write <f:table> and for all children of table it should be started/ended with the same prefix f

```
<f:table>
    <f:tr>
    <f:td> ...  </f:td>
    <f:td> ...  </f:td>
    </f:tr>
</f:table>
```

Note :

xmlns attribute in html specifies the xml namespace for a document. This attribute will be added to first element of your xml.

Eg: <table xmlns : "filename">

Structure of XHTML :

1) DOCTYPE
2) header
3) body

// Create a student table which has 2 columns roll no & name with 3 rows.

Student . xhtml

```
<DOCTYPE html PUBLIC "-// w3c// DTD xHTML
                    1.0 Transitional // EN"
   "http: // www . w3. org / TR/ xhtml / DTD/
                              xhtml1 . dtd">

<html xmlns = "http:// www. w3. org / 1999/ xhtml">
<head> </head>
<body>
   <table>
      <tr>
         <th> Rno </th>
         <th> Name </th>
      </tr>
```

```
        <tr>
          <td> 501 </td>
          <td> Harika </td>
        </tr>
        <tr>
          <td> 502 </td>
          <td> Sai </td>
        </tr>
        <tr>
          <td> 503 </td>
          <td> Nikhil </td>
        </tr>
      </table>
   </body>
</html>
```

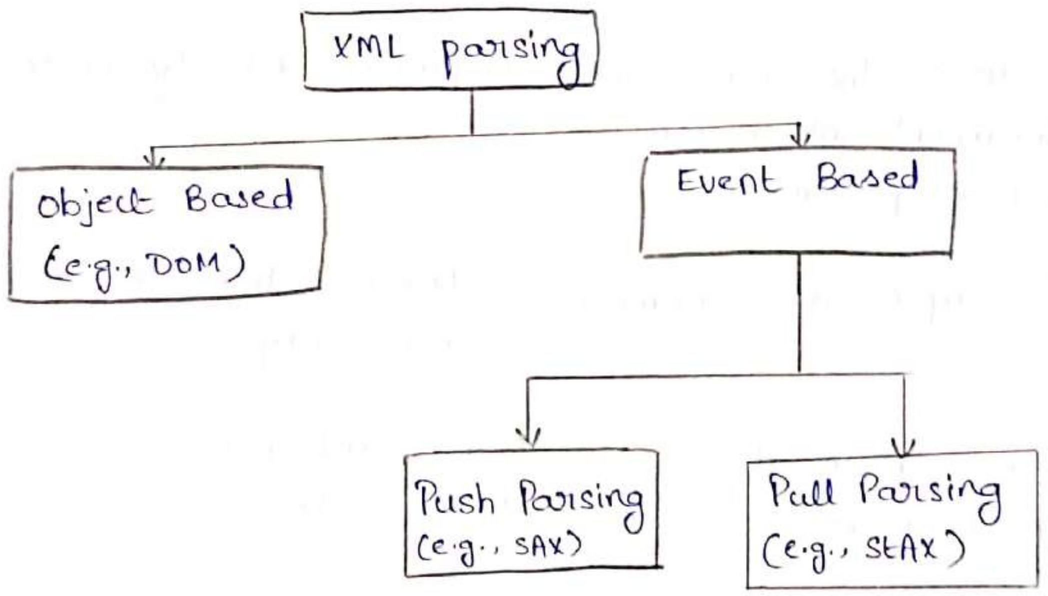
Chapter : 2

* PARSING XML DATA

we can access and parse the XML document in two ways.

- Parsing using DOM (tree based)
- Parsing using SAX (Event based)

* Parsing the XML doc. using DOM methods and properties are called as tree based approach whereas using SAX (simple Api for xml) methods and properties are called as event based approach



DOM and SAX Parsers

DOM	SAX
1) Tree data structure	1) Event based model.
2) Random access	2) Serial access
3) High memory usage	3) Low memory usage
4) Used to process multiple lines (document is loaded in memory)	4) Used to process the document only once.
5) Used to edit the document	5) Used to process parts of the document.
6) Stores the entire xml document into memory before processing	6) parses node by node.
7) occupies more memory	7) Doesn't store the XML in memory
8) we can insert or delete nodes	8) we can't insert or delete nodes.
9) Traverse in any direction	9) Top to bottom traversing
10) Document Object model (DOM) API	10) SAX is Simple API for XML
11) Import javax.xml.parsers.*; import javax.xml.parsers.DocumentBuilder; import javax.xml.parsers.DocumentBuilderFactory;	11) packages required to import import javax.xml.parsers.*; import org.xml.sax.*;
12) Dom is slow rather than SAX	12) SAX generally run a little faster than DOM

* Document Object Model is for defining the standard for accessing and manipulating XML documents. XML DOM is used for

- Loading the xml document
- Accessing the xml document
- Deleting the elements of xml document
- Changing the elements of xml document.

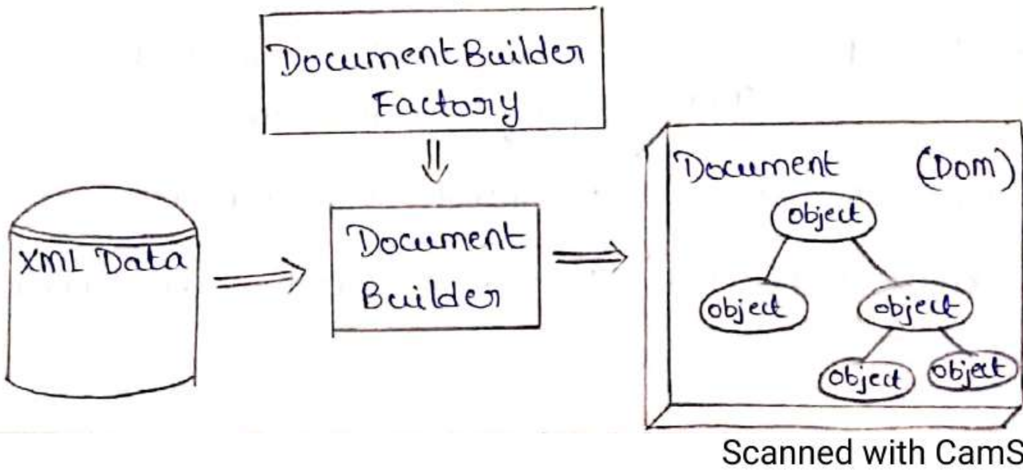
* According to the DOM, everything in an XML document is a node. It considers

- The entire document is a document node
- Every XML element is an element node.
- The text in the XML elements is text nodes.
- Every attribute is an attribute node.
- Comments are comment nodes.

DOM based XML Parsing:

DOM parser parses the entire XML document and loads it into memory; then models it in a "TREE" structure for easy traversal or manipulation.

In short, it turns a XML file into DOM or Tree structure, and you have to traverse a node by node to get what you want.



* In this approach, to access XML document, the document object model implementation is defined in the following packages:

- javax.xml.parsers
- org.w3c.dom

* The following DOM java classes are necessary to process the XML document:

- DocumentBuilderFactory class creates the instance of DocumentBuilder.
- DocumentBuilder produces a document (a DOM) that conforms to DOM specification.

* The following methods and properties are necessary to process the XML document:

Property	Meaning.
nodeName	Finding the name of the node.
nodeValue	Obtaining value of the node.
parentNode	To get parent node.
childNodes	Obtain child nodes.
attributes	For getting the attributes values

Method	Meaning
getElementByTagName (name)	To access the element by specifying its name
appendChild (node)	To insert a child node
removeChild (node)	To remove existing child node.

Java Program to Create XML File

```
import org.w3c.dom.*;
import javax.xml.parsers.*;
import javax.xml.transform.*;
import javax.xml.transform.dom.*;
import java.util.Scanner;
import javax.xml.transform.stream.*;
import java.io.*;
public class CreateXML
{
  public static void main(String[] args) throws
                                        Exception
  {
    DocumentBuilderFactory factory = Document
                      BuilderFactory.newInstance();
    DocumentBuilder builder = factory.
                             newDocumentBuilder();
    Document doc = builder.newDocument();
    Element rootele = doc.createElement
                          ("student-details");
    Element studentele = doc.createElement
                                  ("student");
    Element idele = doc.createElement ("studentid");
    Element nameele = doc.createElement ("name");
    Element markesele = doc.createElement
                                    ("marks");
    Text t1 = doc.createTextNode ("501");
    Text t2 = doc.createTextNode ("naveen");
    Text t3 = doc.createTextNode ("90");
```

~~Text~~

```
idele.appendChild (t1);
nameele.appendChild (t2);
marksele.appendChild (t3);

studentele.appendChild (idele);
studentele.append Child (nameele);
studentele.appendChild (marksele);

rootele.appendChild (studentele);
doc.append Child (rootele);

Transformer t = TransformerFactory.
        newInstance().newTransformer();

t.transform(new DOMSource(doc), new
        StreamResult (new FileOutputStream
                ("student.xml")));

    }
}
```

Above code will generate an xml file with a name student.xml

<u>Student.xml</u>

```
<?xml version="1.0" encoding="UTF-8"
                    standalone="no"?>
<student-details>
   <student>
      <studentid> 501 </studentid>
      <name> naveen </name>
      <marks> 90 </marks>
   </student>
</student-details>
```

Using SAX Parser :

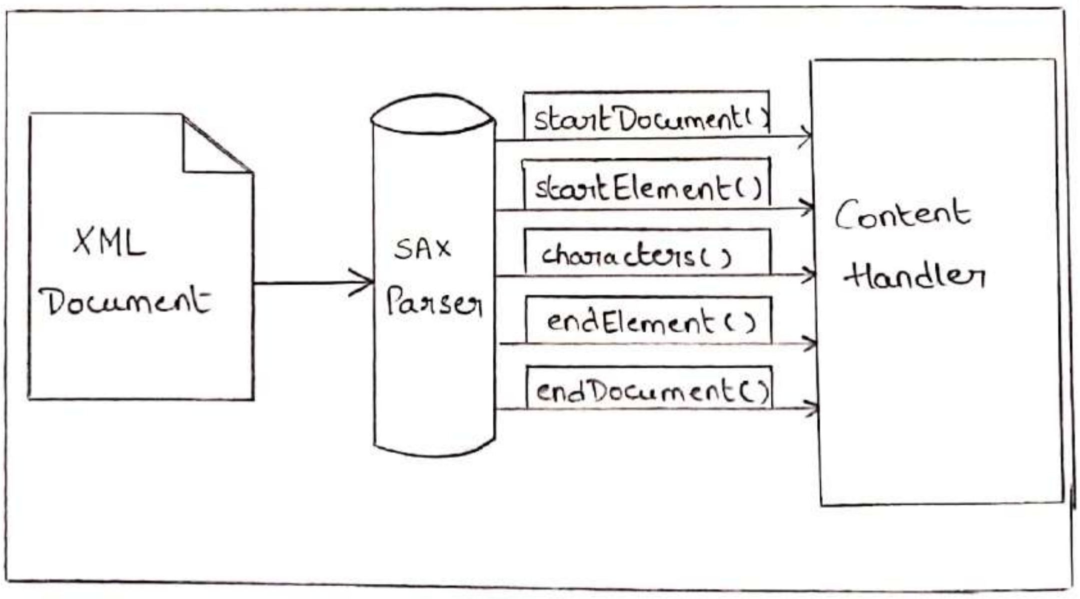
SAX Parser is different from the DOM Parser, where SAX parser doesn't load the complete XML into the memory, instead it parses the XML line by line triggering different events as and when it encounters different elements like: opening tag, closing tag, character data and comments and so on. This is the reason why SAX Parser is called an event based parser.

* Along with the XML source file, we also register a handler which extends the Default Handler class. The DefaultHandler class provides different callbacks out of which we would be interested in:

startElement() - triggers this event when the start of the tag is encountered.

endElement() - triggers this event when the end of the tag is encountered.

characters() - triggers this event when it encounters some text data.



Let's create a demo program to read xml file with SAX parser to understand fully.

student.xml

```
<?xml version = "1.0" encoding = "UTF-8" ?>
<student-details>
   <student>
      <studentid> 501 </Studentid>
      <name> Ramu </name>
      <address> ECIL </address>
      <gender> Male </gender>
   </student>
   <student>
      <Studentid> 502 </Studentid>
      <name> Mahi </name>
      <address> BHEL </address>
      <gender> Male </gender>
   </student>
</student-details>
```

Java program to read data from xml (student.xml) file

```
import java.io.*;
import javax.xml.parsers.SAXParser;
import javax.xml.parsers.SAXParserFactory;
import org.xml.sax.Attributes;
import org.xml.sax.SAXException;
import org.xml.sax.helpers.DefaultHandler;
public class SAXParserDemo extends Default
                                   Handler
{
   public void startDocument()
   {
      System.out.println("begin parsing
                             document");
   }
   public void startElement(String url,
          String localname, String qName,
                        Attributes att)
   {
      System.out.print("<"+qName+">");
   }
   public void characters(char[] ch, int start,
                          int length)
   {
      for(int i=start; i<(start+length); i++)
      {
         System.out.print(ch[i]);
      }
   }
```

```
public void endElement (String uri, String
            localname, String qName )
{
    System.out.print ("</"+ qName + ">");
}
public static void main (String[] arg)
                throws Exception
{
    SAXParser p = SAXParserFactory.
        newInstance(). newSAXParser();
public void endDocument()
{
    System.out.println ("End parsing
                        document");
}
public static void main (String [] arg)
                throws Exception
{
    SAXParser p = SAXParserFactory.
        newInstance(). newSAXParser();
    p.parse(new FileInputStream("student.
            xml"), new SAXParserDemo());
}
}
```