* Introduction to XML :-

XML stands for "Extensible Markup Language". This is used store the data in a document.

By using XML we can displaying of data along with its description and also includes specification of syntaxes.

It is the extension of HTML. generally html restricted to only displaying data.

XML is a meta language that describes contents of document. so in this tags can be called as self-describing data tags.

General syntax of an XML document

```
<root>
 <child>
 <subchild> ... </subchild>
 </child>
 </root>
```

The XML code can be written on a simple notepad and should be saved as "filename.xml".

XML have following features

→ XML documents should be easily and clearly understand by humans.

→ The design of XML documents should be faster to generate

→ The XML document should be simple and easy to create.

→ XML should support many number of applications.

→ XML should be directly usable over the internet

→ XML should be easier to write programs.

→ XML allows users to create their own tags.

Example :-

```
<?xml version="1.0"?>
<document>
<student>
<name> Madhu </name>
<rollno> 10-521 </rollno>
<age> 24 </age>
</student>
<student>
<name> Madan </name>
<rollno> 10-522 </name>
<age> 25 </age>
</student>
</document>
```

⇒ Differences between XML and HTML.

XML	HTML
⇒ XML - eXtensible Markup Language.	HTML - HyperText Markup Language.
⇒ user-defined tags	pre-defined tags.
⇒ user has control on tags	No control.
⇒ case-sensitive	Not-case sensitive
⇒ Root element is user defined and only one root element is allowed	Root element is <html>
⇒ we can generate new Markup language using XML	No such possibility
⇒ self describing data can be possible	Not possible.

⇒ XML declaration always starts with XML keyword

<?xml version="1.0"?>

* Document Type Definition (DTD) :-

Document type definition is a certain pieces of code which can defines structure of XML document.

Each DTD's carries certain list of elements, which specifies the rules for structuring a given XML document.

Each DTD specifies the relationship between root elements, child elements and sub child elements.

DTD's are optional in xml, but recommended for clarity purpose.

DTD's can be declared internally (with in XML document) and as an external files (with in sparate file with ".dtd" extension).

Syntax:-

```
<! DOCTYPE name-doc SYSTEM "path">
```

Basic building blocks :-

→ Tags :-

The XML allows user to create own tags. usually the tag <anyname> is an opening tag </anyname> refers to its equivalent closing tag.

Example : <name> Madhu </name>

→ Elements :-

The XML DTD document is composed of number of elements. These elements are used to represent the data in XML documents.

Declaration:-

```
<!ELEMENT name-of-element (context)>
```

* To declare an empty element

Syntax:- <! ELEMENT name-of-element (EMPTY)>

* To declare elements, which carries data

Syntax:

<!ELEMENT name of element (#PCDATA)>

Here "PCDATA" or 'parsed character data' is the data which is to be parsed by the XML parser.

(or)

<!ELEMENT name-of-element (#CDATA)>

Here "CDATA" or 'Character Data' is the data which will not be parsed by the XML processor.

* To declare elements, which can have childrens.

Syntax:

<!ELEMENT name-of-element (child.name)>.

→ Attributes:-

These are used to provide the additional information along with elements.

In DTD, the attributes are declared by using "ATTLIST".

Syntax:-

<!ATTLIST name-of-element name-of-attribute
attribute-type [default-value]>

In the above syntax, the field 'attribute-type' can specifies the following pre defined values

* ID ⇒ It is a value which remains unique

* CDATA ⇒ The value supplied here is nothing but character data.

* ENTITY ⇒ The value supplied in this case is nothing but an entity.

In the same way, the field 'default-value' can specifies the following predefined values.

* Default-value ⇒ It is a default value of given attribute.
* #REQUIRED ⇒ It means the value for the attribute is required
* #IMPLIED ⇒ It means the attribute is not required
* #FIXED ⇒ Here a fixed value is supplied.

EX:-

<!ATTLIST madhu address CDATA #REQUIRED>

→ Entities:-

Some markup elements can contain complex data. These elements are called as entities

These are included in sml file, these are used to create small pieces of data which you want use repeatedly throughout your schema.

Syntax:- <!ENTITY name-of-entity "value">

EX: <!ENTITY Book "web Technology">

used as

<author> the Book author is Madhu </author>

* Important Keywords used in DTD declaration:-

The following are important keywords in DTD

→ #REQUIRED keyword

→ #IMPLIED keyword

→ #FIXED keyword

→ #PCDATA keyword.

→ #REQUIRED keyword:-

In some situations, when an attribute value is essential, a keyword called as #REQUIRED is used following by the value.

Example:-

<! ATTLIST branch name (CSE | IT | ECE | MECH) #REQ

In the above example, the name attribute should require a value.

→ #IMPLIED Keyword:-

In most cases the attribute is not required and may or may not appear within the element. Hence the attribute will not hold a fixed or default value.

Example:-

<! ATTLIST Emp Address CDATA #IMPLIED>

In the above example, the address attribute is a char data attribute, which is optional (i.e it has no default value no fixed value and is not required).

→ #FIXED Keyword:-

In some situations an attribute value is always fixed. when the attributes value cannot be changed, #FIXED keyword is used followed by the value.

Example:-

<! ATTLIST Emp Empid CDATA #FIXED '46'>

In the above example, the empid attribute has a fixed value i.e 46. #FIXED keyword is added before the fixed value

→ #PCDATA Keyword:-

PCDATA refers to the parsed character DATA. #PCDATA within the element contains the data that is parsed by the parser.

Example:-

<! ELEMENT name (#PCDATA)>

Examples for DTD:-

Example:-

Create a DTD for a remainder, it has following child elements - heading, to, from, message.

remainder.dtd

```
<!ELEMENT remainder(heading, to, from, message)>
<!ELEMENT heading(#PCDATA)>
<!ELEMENT to(#PCDATA)>
<!ELEMENT from(#PCDATA)>
<!ELEMENT Message(#PCDATA)>
```

remainder.sml

```
<?xml version="1.0"?>
<!DOCTYPE note system "remainder.dtd">
<remainder>
<heading> Final notice to Mr.Madhu </heading>
<to> Mr. Madhu </to>
<from> Ravi gupta (Hr. manger) </from>
<message> your last date for joining office is 4th
          Feb </message>
</remainder>
```

Similarly, the above declaration can be made internally as shown below

```
<?xml version="1.0"?>
<!DOCTYPE remainder [
<!ELEMENT remainder (heading, to, from, message)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT message (#PCDATA)>
]>
```

```
<remainder>
<heading> Final notice to Mr. Madhu </heading>
<to> Mr. madhu </to>
<from> Ravi gupta </from>
<message> your last date for joining office is 14th
          FEB </message>
</remainder>
```

Example :-

create a DTD for a catalog of four-stroke motorbike where each motorbike has the following child elements. make, model, year, color, engine, chasis number, and accessories. The engine element has the child elements these are engine number, number of cylinders, type of fuel. The accessories element has the attributes like disc brake, auto-start & radio, each of which is required and has the possible values 'yes' and 'NO'. Entities must be declared for the names of the popular motorbike makes.

Solution :-

automob.dtd

```
<!ELEMENT catalog (motorBike)* >
<!ELEMENT motorBike (make, model, year, color, engine
          chasis_num, accessories)>
<!ELEMENT make (#PCDATA)>
<!ELEMENT model (#PCDATA)>
<!ELEMENT year (#PCDATA)>
<!ELEMENT color (#PCDATA)>
<!ELEMENT engine (engine_num, cylinders_num, fuel_type)
<!ELEMENT chasis_num (#PCDATA)>
<!ATTLIST accessories disk_brake (YES|NO) #REQUIRED
          auto_start (YES|NO) #REQUIRED
          radio (YES|NO) #REQUIRED >
```

The DTD can be used by the following XML document

automob.xml

```
<? xml version = "1.0" ? >
<! DOCTYPE catalog SYSTEM "automob.dtd" >
<catalog >
  <motorBike>
   <make> Bajaj Auto </make>
   <model> pulsor 180 DTSi </model>
   <year> 2013 </year>
   <color> Black </color>
   <engine>
    <engine.num> 234 </engine_num>
    <Cylinder.num> 20 </ Cylinder.num >
    <fuel.type > premium </fuel type >
   </engine>
   <chasis_num> 789 </chasis_num>
   <accessories disc-brake = "yes" auto-start = "yes"
                                  radio = "No" />
  </motorBike>
  </catalog>
```

* XML Schema :-

XML schema mainly used for structuring XML documents. like DTDs, they also from major building blocks of xml documents.

using DTD along XML raises the complexity level as DTDs defines their own sets of grammer as well as syntaxes.

XML schema is capable of defining several elements and attributes which ought to appear in a given documents.

It can be defined child elements, their number as well as their order.

It defines data types, default and fixed values for the elements and attributes.

For this purpose, it has the form of XML schema language which is also known as XML schema definition (xsd).

XML schemas are created by using XML syntax where as DTD's use a separate syntax.

XML schema support the namespace functionality but DTD doesn't support this functionality.

XML schemas specify the type of textual data that can be used with in attributes and elements.

Disadvantages of XML schema :-

→ XML schema is difficult to learn and design.

→ If we use the XML schema for complex and large operations, then the processing of XML document may slow down.

→ The XML document can't be displayed if the corresponding schema file is absent.

→ Namespace in XML :-

The purpose of using namespace in XML is to avoid clashing (or) confusion of names.

Ex:-

```
<course> CCNA </course>
<course> Java </course>
```

In the above example, it can be seens the names of the courses. In those one is network course and another one is software course. But it is difficult to identify, both are given in the same

name 'course'. To avoid confusion, XML namespaces uses prefix before the name.

Ex:-

<networking : course> CCNA </networking : course>

<software : course> JAVA </software : course>

Creation of XML namespace is done by using the 'xmlns' keyword along with URI.

Example:-

```
<? xml version = "1.0" ? >
<prefix = "school" xmlns:school = " http:// URI// namespace">
<prefix = " pg " xmlns:pg = " http:// URI// namespace " > .
<education>
<school: subject > science </school: subject>
<pg : subject> MFCS </pg: subject>
</education>
```

Data types in XML schema :-

It supports following data types.

those are

→ Binary data type :- It includes the binary data i.e '0's or '1's.

→ Boolean data type :- It includes one of the following two values - true
- false.

→ Number data types :-

There are three main number data types

Those are

→ float datatype :- This data type contains 32-bit floating point values.

→ Double data type :- This data type contains 64-bit floating point values.

→ Decimal data type :- It includes the decimal numbers either +ve or -ve values.

→ Date data type :- It specifies the current date in the format of YYYY-MM-DD.

→ Time data type :- It shows the time in the form of hh:mm:ss.

→ String data type :- It includes the series of characters such as strings.

Example :-

The following example is an XML schema file called "note.xsd", that defines the elements of XML document note.xsd.

```
<xs: schema>
<xs: element name="note">
<xs: complexType>
<xs: sequence>
  <xs: element name="to" type="xs: string"/>
  <xs: element name="from" type="xs: string"/>
  <xs: element name="heading" type="xs: string"/>
  <xs: element name="body" type="xs: string"/>
</xs: sequence>
</xs: complexType>
</xs: element>
</xs: schema>
```

note.xml :-

```
<? xml version="1.0" ?>
<note schemalocation="note.xsd">
<to> Madhu </to>
<from> Ravi </from>
<heading> Reminder </heading>
<body> Don't forget me this weekend </body>
</note>
```

* Presenting XML :- (XSLT)

Generally the XML document is used to describe the data. We can display the data in XML document with in browser by using XSL.

XSL - extensible stylesheet language.

→ XSL describes how the XML document should be displayed.

→ To display the data in XML document, XSL a template is created.

→ XSL basically transforms one data structure to another i.e XML to HTML.

The XSL have following elements

* <xsl: stylesheet>, defines that this document is a XSLT style sheet document.
* <xsl: template>, defines a template. the match="/" attribute associates the template with the root of the XML document.
* <xsl: value-of> can be used to extract the value of an xml element and add it to the o/p stream.
* <xsl: for-each> can be used to extract the value of an xml element repeatedly.

Example:-

Students.xml

```
<?xml version="1.0" encoding="utf-8"?>
<?xml-stylesheet type="text/xsl" href="students.xsl"
<students>
<student>
<sno> 501 </sno>
<name> ABC </name>
<branch> CSE </branch>
</student>
<student>
<sno> 401 </sno>
<name> XYZ </name>
<branch> ECE </branch>
</student>
</students>
```

Students.xsl

```
<?xml version="1.0" ?>
<xsl: stylesheet version="1.0"
    xmlns:xsl = "http://www.w3.org/1999/XSL/Transf
<xsl:template match="/">
<html>
<body>
<h2> student details </h2>
<br/>
<table border="1">
<tr>
<th> SNO </th>
<th> NAME </th>
<th> BRANCH </th>
</tr>
<xsl:for-each select="students/student">
<tr>
<td>
<xsl: value-of select="sno"/>
</td>
```

```
<td>
<xsl:value-of select="name" />
</td>
<td>
<xsl:value-of select="branch" />
</td>
</tr>
</xsl:for-each>
</table>
</body>
</html>
</xsl:template>
</xsl:stylesheet>.
```

output:-

Student details

SNO	NAME	BRANCH
501	ABC	CSE
401	XYZ	ECE

* DOM (Document Object Model):-

DOM is a platform and language that allows programs and scripts to dynamically access and update the content, structure and style of a xml document.

DOM defines the objects and properties of XML elements and the methods to access them.

DOM exposes the whole document to applications.

The parser reads XML into memory and converts into an XML DOM object that can be accesses with Java script.

→ Microsoft's XML parser is built into Internet Explorer.

```
var XMLDOC = new ActiveXObject ("Microsoft.XMLD
XMLDOC.load ("file1.xml");
```

In the above code

→ The first line creates an empty Microsoft XML downm object

→ The second line tells the parser to load an XML down called "file1.xml".

XML DOM properties :-

The following are some typical DOM properties

→ x.nodeName – the name of x.

→ x.nodeValue – the value of x.

→ x.parentNode – The parent node of x.

→ x.childNodes – the child nodes of x.

→ x.attributes – The attributes nodes of x.

Where x is the node object.

XML DOM Methods :-

→ x.getElementsByTagName(name) – get all elements w a specific tag name.

→ x.appendChild(node) – insert a child node to x.

→ x.removeChild(node) – remove a child node from x.

Example :-

The following is details of users i.e "users.xml"

```
<?xml version="1.0"?>
<users>
<user>
<userid> 1 </userid>
<password> a </password>
</user>
```

```
<user>
<userid> 2 </userid>
<password> b </password>
</user>
<user>
<userid> 3 </userid>
<password> c </password>
</user>
</users>
```

"Login.html"

```
<html>
<head>
<title> Login </title>
<script language="Javascript" >
function validate()
{
var xmldoc = new ActiveXObject("Microsoft.XMLDOM");
xmldoc.load("users.xml");
var id = fl.uname.value;
var pass = fl.pwd.value;
var x = xmldoc.getElementsByTagName("userid");
var y = xmldoc.getElementsByTagName("password");
for(i=0; i<x.length; i++)
{
if(x[i].childNodes[0].nodevalue == id)
{
if(y[i].childNodes[0].nodevalue == pass)
{
alert("successfully logged...");
return;
}
else
{
alert("Invalid password...");
fl.pwd.focus();
```

```
    return;
  }
 }
}
alert ("Invalid userName ...");
f1.uname.focus();
}
</script>
</head>
<body>
<form name="f1">
<br/>
<br/>
<table align="center">
<caption> Login </caption>
<tr>
<td> User name : </td>
<td><input type="text" name="Uname" ></inputs
</td>
</tr>
<tr>
<td> password : </td>
<td> <input type="password" name="pwd"></input
</td>
</tr>
<tr>
<td>
<input type="button" name="Login" value="log
onclick="validate()"> </input>
</td>
<td>
<input type="button" name="resel" value="Reset"/>
</td>
</tr>
</table>
</form></body>
</html>
```

* XML processors :-

The purposes of XML processors are as follows

→ First, The processor must check the basic syntax of document for well-structureness.

→ second, the processor must replace all references to entities in an XML document.

→ Third, DTD's and XML schemas can specify that certain values in an XML document have default values, which must be copied into the XML document during processing

Generally we have two processors for XML documents for processing

Those are → SAX (Simple API for XML)
→ DOM (Document object Model)

SAX (Simple API for XML)	DOM (Document Object Model)
* The SAX approach to processing is called event processing	* The DOM Approach to processing for build The DOM tree.
* This scans The XML document from begining to end.	* This access the random parts of The XML document.
* The entire document accessed only once by the application when the even generated.	* If any part of The document must be accessed more than once by the application.
* It is faster than The DOM	* It is the slow than the SAX.
* The SAX structure doesn't store entire document	* The DOM Structure is stored entirely in memory, for large documents.
* It performs the syntactic analysis of the document	* DOM requires SAX parser as a front end for analysis.