UNIT - II

XML

→ XML stands for extensible markup language

→ XML was designed to store & transport data.

→ XML was designed to be both human and machine-readable.

Why study XML?:

XML plays an important role in many diffent IT systems.

XML is often used for distributing data over the internet.

It is important (for all types of software developers) to have a good understanding of XML.

→ There are 3 important characteristics of XML that make it useful in a variety of systems and solutions.

→ XML is extensible :- XML allows you to create your own self-descriptive tags, or language, that suits your application.

→ XML carries the data, does not present it :- XML allows you to store the data irrespective of how it will be presented.

→ XML is a public standard :- XML was developed by an organization called the world wide web consortium (W3C) and is available as an open standard.

XML Usage:

A short list of XML usage says it all.

→ XML can work behind the scene to simplify the certain of HTML documents for large web sites.

→ XML can be used to exchange the information b/w organizations and systems.

→ XML can be used for offloading & reloading of databases.

→ XML can be used to store and arrange the data, which can customize your data handling needs.

→ XML can easily be merged with style sheets to create almost any desired output.

→ Virtually, any type of data can be expressed as an XML document.

what is markup :- XML is a markup language that defines set of rules for encoding documents in a format that is both human-readable & machine-readable.

→ following example shows how XML markup looks, when embeded in a piece of text.

```
<message>
    <text> Hello, world! </text>
</message>
```

→ this snippet includes the markup symbols, or the tags such as <message>. </message> and <text> ..... </text>

IS XML a programming language?

→ A programming language consists of grammer rules and it's own vocabulary which is used to create computer programs.

→ These programs instructs computer to perform specific task.

→ XML does not qualify to be a programming language as it does not perform any computation or algorithms. It is usually stored in a simple text file and is processed by special s/w that is capable of interpreting XML.

XML syntax:-

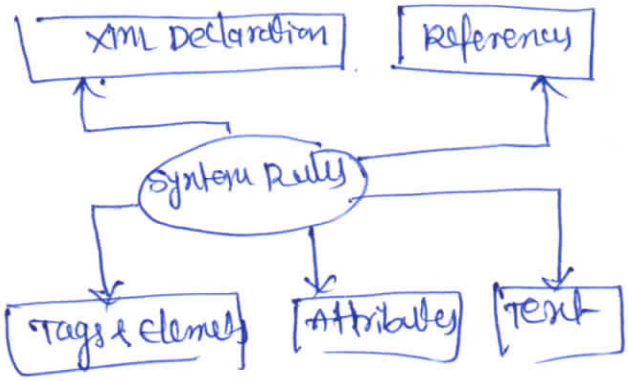
```
<?xml version="1.0"?>
<contact-info>
    <name> Anyname </name>
    <company> Infosys </company>
    <phone> 99999 </phone>
</contact-info>
```

→ you can notice there are two kinds of information in the above example.

→ markup, like <contact-info> and

→ the text, or the character data, Tutorials Infosys and phone no.

→ The following diagram depicts the syntax rules to write different types of markup and text in an XML document.



→ Unicode transformati
format

XML Declaration :-

The XML document can optionally have an XML declaration. It is written as bellow.

<?xml version="1.0" encoding="UTF-8"?>

where version is XML version and encoding specifies the character encoding used in the document.

Syntax Rules for XML Declaration,

→ The XML declaration is case sensitive and must begin with "<?xml>" where XML is lowercase.

→ If document contains XML declaration, then it strictly needs to be the first statement of the XML document.

→ The XML declaration strictly needs be the first statement in the XML document.

Tags and Elements :-

→ An XML file is structured by several XML elements, also called XML nodes or XML tags.

→ XML elements name are enclosed by triangular brackets <> as shown bel

<element>

Syntax Rules for Tags and Elements :-

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

<element> ..... </element> (or) <element />

Nesting of elements :- An XML-elements can contain multiple XML-elements as it's 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> CMR <contact-info>
</company>
```

following examples shows correct nested tags :

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

Root element :- A XML document can have only one root element

```
<root>
  <x> --- </x>
  <y> --- </y>
</root>
```

for example following is not a correct XML document, because both the x and y elements occure at the top level without a root element.

```
<x> ---- </x>
<y> ---- </y>
```

Case sensitivity :- The names of XML-elements are case-sensitive. that means the name of the start and the end elements need to be exactly in the same case.

for example :- <contact-info> is different from <Contact_Info>.

Attributes :-

An attributes specifies a single property for the element, using a name/value pair,

→ An XML-element can have one or more attributes. for example.

```
<a href = " http://www.google.com/ "> XML </a>
```

Here, href is the attribute name and

http://www.google.com is a arribute value.

Syntax Rules for XML attributes :-

→ Attribute names in XML are case sensitive. that is HREF and and href are considerd two different XML attributes.

→ Same attribute cannot have two values in a syntex. the following example shows incorrect syntax. because the attribute b is specified twice:

.....

→ attribute values must always appear in quotation marks.

XML References :- References usually allow you to add or include additional text markup in an XML document.

→ References always begin with the symbol "&",

→ which is reserved character and end with the symbol ";"

→ XML has two types of reference.

① Entity References

② Character References.

→ Entity Reference :- An entity reference contains a name b/w the start and the end dilimeters.

→ Ex :- &amp ; where amp is name, The name refers to a predefined string of text and for markup.

→ Character References :- these contain references such as A contains a hash mark ("#") followed by a number.

→ The number always refers to the unicode code of a character.

→ In this case 65 refers to alphabet 'A'.

XML Text :- The name of XML-elements and XML-attributes are case-sensitive which means the name of start and end elemets need to be written in the same case.

→ TO avoid character encoding problems, all xml files should be saved as unicode UTF-8 or) UTF-16 files.

→ whitespace characters like blanks, tabs and line-breaks between XML-elements and between the XML-attribute will be ignored.

→ Some characters are reserved by the xml syntax itself. Hence, they cannot be used directly.

→ To use them, some replacement-entities are used, which are listed below

not allowed character	replaced-entity	character description
<	<	less than
>	>	greater than
&	&	ampersand
'	'	apostrophe
"	"	quotation mark.

XML Tags:-

→ 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

→ Start tag (<address>

→ End tag </address>

→ Empty tag → The text that appears b/w start tag & end-tag is called content, An element which has no contents is termed as empty.

→ An empty element can be represented in two ways.

(1) A start-tag immediately followed by an end-tag. as follows.

<hr> </hr>

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

<hr/>

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.

Syntax:-

<element-name attribute1 attribute2>

... content

</element-name>

Empty element:- An empty element has following syntax.

<name attribute1 attribute2/>

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 used 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>
```

Empty tag:-

→ The text that appears 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-tg immediately followed by an end-tag as shown below.

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

2) A compile empty-element tag is as show below

```
<hr/>
```

XML Tags Rules:-

Rule 1:-

XML tags are case-sensitive.

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

XML elements :-

→ XML elements can be defined as basic building blocks of XML

→ Elements can behave as containers to hold text, elements, attributes, media objects or all of them.

→ Each XML document contains one or more elements.

Syntax :

```
<element-name attribute1 attribute2 >
    .... content
</element_name>
```

Empty element :-

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

XML Attributes :-

→ Attributes are part of the XML elements.

→ An element can have multiple unique attributes.

→ Attributes gives more information about XML elements.

→ An XML attribute is always a name-value pair.

Syntax

An XML attribute has following syntax:

```
<element-name attribute1 attribute2>
    ...
    Content...
</element-name>
```

where attribute1 and attribute2 has the following form:

name = "value".

→ The value has to be in double (" ") or single (' ') quotes.

Here, attribute1 and attribute2 are unique attribute labels.

Attribute Types :-

Following table lists the type of attributes:

→ String Type :- It takes any literal string as a value. CDATA is a string type CDATA is character data. This means, any string of non-markup characters is a legal part of the attribute

Tokenized type:- This is more constrained type. The validity constraints noted in the grammer are applied after the attributes value is normalized.

→ The Tokenized type attributes are given as.

ID : It is used to specify the elemet as unique

IDREF :- It is used to reference an ID that has been named for another elemet.

IDREFS: It is used to reference all IDs of an elemets.

ENTITY: It indicates that the attribute will represent-an external entity in the document.

NMTOKEN :- It is similar to CDATA with restrictions on what data can be part of the document.

Enumerated Type: This has a list of predefined value in its declaration. out of which it must assign one value. There are two types of enumerated attribute.

Notation Type :- It declares that an element will be referenced to a Notation declared somewhere else in the XML Document.

Enumeration :- Enumeration allows you to define a specific list of value that the attribute value must match

Elements Attributes Rules :-

→ An-attribute name must not appear more than once in the same start-tag or-empty elemt tag.

→ An-attribute must be declared in the Document Type Defination (DTD) using an Attribute-list Declaration.

XML comments :-

→ XML comments are similer to HTML comments

→ The comments are added as notes or lines for understanding the purpose of an XML code.

→ Comments can be used to inside related linics ⇔, information and terms.

They visible in the source code.

→ commetry may appear in anywhere in XML code.

Syntax :- <! ..... your commet ..... >

Document Type Definition :- It defines the structure & the legal elements and attributes of an XML document.

Why use a DTD? with a DTD, independent groups of people can agree on a standard DTD for interchanging Data.

→ An application can use a DTD to verify that xml data is valid.

Building Blocks of DTD :-

- → Elements
- → Attributes
- → Entities
- → PCDATA — parsed character Data
- → CDATA → character data.

→ DTD check vocabulary and validy of the structure of xml document.

→ An XML DTD can be either specified inside the document, or it can be kept in a seperate document and then linked separately.

Syntax :-

<! DOCTYPE element DTD identifier [ ] >

Ex <! DOCTYPE note [

<! ELEMENT note(to, from, heading, body) >

< u to (#PCDATA)>

< u from <#PCDATA>>

< u heading (#PCDATA)>

< u body (#PCDATA)>]>

⇒ !DOCTYPE note defines that the root element of the document is note.

⇒ !ELEMENT note defines that the note element must contain the elements to, from, heading, body".

⇒ !ELEMENT to defines the to elements to be type #PCDATA.

→ Internal DTDS :-

→ A DTD is referred to as an internal DTD if elements declared within the xml file. To refer it as internal DTD.

→ Standalone attribute in xml declaration must be set to yes.

→ This means, the declaration works independent of external source.

Syntax :-

<! DOCTYPE root-elemt [elemt--declaration]>

where root_element is the name of root element

→ element-declarations is where you declare the elemts.

Example :

```
<?xml version = "1.0" encoding = "UTF-8" standalone = "yes"?>
<! DOCTYPE address [
        <! ELEMENT address (name, company, phone)>
        <  "  name (#PDATA)>
        <  "  company (#PDATA)>
        <  "  phone (#PDATA)> ]>
<address>
    <name> CMR </name>
    <company> CMR </company>
    <phone> 040 </phone>
</address>
```

Start declaration - Begin the xml declaration with the following statement

<?xml version = "1.0" encoding = "UTF-8" Standalone = "yes" >

DTD :- Immediately after the xml header, the document type declaration follows, commonly referred to as the DOCTYPE :

<! DOCTYPE address [

→ The DOCTYPE declaration has an exclamation mark(!) at the start of the elemt name.

→ The DOCTYPE informs the parser that a DTD is associated with this xml document

DTD Body :- The DOCTYPE declaration is followed by body of the DTD, where you declare elements, attributes, entities, and notations

END Declaration :- finally, the declaration section of the DTD is closed using a closing bracket and a closing angle bracket (]>)

→ This effectively ends the definition, and thereafter, the XML document follows immediately.

Rule :-

→ The document type declaration must appear at the start of the document

→ Similar to the DOCTYPE declaration, the element declarations must start with an exclamation mark.

→ The name in the document type declaration must match the element type of the root element.

External DTD :-

→ In external DTD elements are declared outside the XML file.

→ They are accessed by specifying the system attributes which may be either the legal.dtd file or a valid URL

→ To refer it as external DTD, standalone attribute in the XML declaration must be set as no.

→ This means, declaration includes information from the the external source.

Syntax

<! DOCTYPE root-element SYSTEM "file_name">

where file-name is the file with .dtd extension.

Example :-

```
<? xml version = "1.0" encoding = 'UTF-8" Standalone = "no" ?>
    <? DOCTYPE address system "address.dtd">
     <address>
          <name> CMR </name>
          <company> CMR </company>
          <phone> 040 </phone>
     </address>
```

The content of the DTD file address.dtd are as follows

```
<! ELEMENT address (name, company, phone)>
<! ELEMENT name (#PDATA)>
<!  "    company ( "  )>
<!  "    phone ( " )>
```

Types :- you can refer to an external DTD by using either system identifier or public identifiers.

System identifiers :-

A system identifier enables you to specify the location of an external file containing DTD declaration.

Syntax :- <! DOCTYPE name SYSTEM "address.dtd" [...]>

→ As you can see, it contain keyword SYSTEM and a URI reference pointing to the location of the document.

public identifiers :- public identifiers provide a mechanism to locate DTD resources and are written as below.

<! DOCTYPE name PUBLIC "-// Begining xml / DTD address Example //En" >

→ As you can see, it begin with keyword public, followed by a specialized identifier.

→ public identifier are used to identify an entry in a catalog.

→ public identifiers can follow any format, however, a commonly used format is called formal public identifiers or FPIDS

XML character Entities :-

3 types of character entities

→ predefined character entities → Ampersand : &
  single quote : '
  Greater than : >
  less than : <
  Double quote : "

→ Numbered character entities → decimal (or) hexadecimal format.
  &# decimal number; &#x hexadecimal number;
  ex :- " 2

→ Named character entity

→ Each entity is identified with a name.
→ Acute represents captial 'A'

XML Schemas

→ DTD's have several disadvantages
→ One is that DTD's are written in a syntax unrelated to xml, so they cannot be analyzed with an xml Document.
→ It can be confusing for people to deal with two different Syntatic forms.
→ To overcome their weaknesses, *
→ xml Schema, which was disigned by the w3c, is one of these alternative.
→ xml Schema is an xml document, so it can be parsed with an xml parsere.
→ To promote the transition from DTD to XML Schemas, xml Schema was designed to allow any DTD to be automatically converted to an equivalent xml schema.

Schema fundamentals:

→ A Schema is similar to a class defination.
→ an xml document that conforms to the structure defined in the Schema is similar to an object of the schema's class.
→ Schemas have two primary purposes.
→ First, a schema specifies the structure of it's instance xml documents, including which elemets and attributes may appear in the instance document, as well as where and hoco often they may appear.
→ Second, a schema specifies the data type of every element and attribute of it's instance xml documet.

Namespaces:

→ The w3c has developed a standard for xml namespaces at (http://www.w3.org/TR/REC-xml-names).
→ An xml namespace is a collection of element & attribute names used in xml documents.
→ The name of a namespace usually has the form of a uniform resource identifier (URI)
→ A namespae for the elements and attributes of the root element declared as the value of the attribute xmlns;
→ The form of a namespace declaration for an elemet follows.

<element_name xmlns[:prefix] = URI>

→ The square brackets indicate that what is within them is optional.

→ The prefix, if included, is the name that must be attached to the names in the declared namespace.

→ prefix is used for two Reasons.

→ 1) , the URI is too long to be typed on every occurrence of every name from the namespace.

→ 2) A URI includes characters that are illegal in XML.

Ex prefixed namespace declaration.

```
<birds xmlns:bd = "http://www.audubon.org/names/species">
```

→ If an element has more than one namespace declaration.

```
<birds xmlns:bd = " URI " >
       xmlns:html = "http://www.w3.org/1999/xhtml">
```

→ Default namespace declaration :-

Consider the following example in which two namespaces are declared

→ 1st is declared to be the default namespace,

→ 2nd defines the prefix, with cap.

```
<states>
    xmlns = "http://www.states-info.org/states".
    xmlns:cap = "http://www.state-info.org/states-capitals".
    <state>
        <name> India </name>
        <population> 4 taks. </population>
        <capital>
            <cap:name> Delhi </cap:name>
            <cap:population> 1200 </cap:pupulation>
        </capital>
    </state>
    <!-- more states -->
</states>
```

Defining a schema :-

→ every schema has it's root elemet, As stated, the schema elemet

→ Schema defines a namespaces woker as DTD defny the tag set.

→ The name of the namespace defnd by a schema must be specified with the "targetNamespace" attribute of the schema elemet

→ The targetNamespase is specifed by assigny a namespace to be terget namespace attribute, as in the followny.

targetNamespace = "http://cs.uccs.edu/planeschema"

→ If we want the elements and attributes that are not defined directly in the schema elemet to be included in target namespace.

→ Schema elemet 'elemetFormDefault' must be set to qualifed, as in the followny

elemetFormDefault = "qualifed"

→ The defaut Namespace, which is the source of the unprefixed nams in the schema.

ex xmlns = "http://cs.uccs.edu/planeschema"

ex
```
<xsd:schema
    xmlns:xsd=http://www.w3.org/2001/xmlschema (the namespace for the schema itself prefix is xsd)
    xmlns targetNamespace = http://cs.uccs.edu/planeschema
    xmlns = http://cs.uccs.edu/plan.  (CNS where elems defnd here will be placed)
                                      (default schema)
    elemetFormDefault = "qualifed" >
```

→ Defining a Schema Instance :-

→ first, an instance document normally defines it's default namespace to be the one defined in its schema.

<planes xmlns = http://cs.uccs.edu/planeschema... >

→ The second attribute specification in the root element of an instance document is for the schemaLocation attribute.

→ This attribute is used to name the standard namespace for instancy, which is xmlschema-instance.

→ xmlschema-instance namespace and predefines the prefix, xsi for it

xmlns:xsi = "http://www.w3.org/2001/xmlschema-instance";

→ 3rd the instance document must specify the filename of the schema where the default name space is defined.

→ This is accomplished with the schemaLocation attribute, which takes two values: 1) namespace of the schema
2) filename of the schema.

xsi:schemaLocation = "http://cs.uccs.edu/planeschema planes.xsd".

→ An overview of Datatypes:

→ There are all two user-defined XML schema datatypes.

① simple ② complex.

→ A simple data type is one whose content is restricted to strings.

→ A simple type cannot have attributes or include nested elements.

→ A complex type can have attributes and include other data types as elements.

→ XML schema defines all data types, 19 of which are primitive and 25 of which are derived.

→ The primitive data types include string, Boolean, float, time & URI

→ The predifined derived types includes byte, long, decimal, UnsignedInt, PositiveInteger, and NMTOKEN.

→ User defined data types are called as a base type.

→ user-defined types are derived types.

→ integer primitive data types has 8 possible facets :-

① totalDigits
② maxInclusive
③ maxExclusive
④ minInclusive
⑤ minExclusive
⑥ Pattern,
⑦ enumeration
⑧ whitespace.

→ Data Declarations in an xml schema can be either local or global.

→ A local declaration is one that appears inside an element that is a child of the schema element.

→ locally declared element is visible only in that element.

→ A global declaration is one that appears as a child of the schema element.

→ Global elements are visible in the whole schema in which they are declared.

Simple type ex

```
<xsd : simpleType name = "firstName">
    <xsd : restriction base = "xsd : string">
     < xsd : maxLength value = "10" />
    </xsd : restriction>
</xsd : simpleType>
```

Complex types :-

→ Complex types are defind with the "complexType" taye

→ The elemes that are the content of an elemn-only elemnt must be contained in an ordered group, unorder-group, a choice, or a named-group

OR sequence elemn is used to contain an ordered group of elemnts.

```
<xsd : complexType name = "sports_car">
  <xsd : sequence>
   <xsd : element- name = "make" type = "xsd : string" />
   < xsd : u  o = "model" u = u  />
   <xsd : u  u = "year" u = "xsd : decimal" />
  </xsd : sequence>
</xsd : complexType>
```

What is an xml schema?

→ An XML schema describes the structure of an xml document.

→ An xml schema is also referred to as XML Schema Definition (XSD).

Syntax:

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
```

Exam:

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
<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:"  name="body" type="/>
```

→ The purpose of an xml schema is to define the legal building blocks of an xml document:

→ xml schema defines the elements, attributes and datatypes.

→ schema element supports Namespaces.

→ it is similiar to a database schema that describes the data in a database.

→ xml standards are defined by xml schemas.

→ xml schema is an xml-based alternative to DTD.

→ xml schema supports Data types.

xml schema supports 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 patterns (data formats)

→ it is easier to convert data between different data types.

XML Schemas use XML Syntax

⇒ Another great strength about XML Schemas is that they are written in XML

→ you can use your xml editor to edit your schema files.

→ you can use your xml parser to parse your schema files.

→ you can manipulate your schema with the xml Dom.

⇒ xml schemas are extensible, because they are written in xml

⇒ with an extensible schema difination you can:

- → Reuse your schema in other schemas
- → create your own data types derived from the standard types
- → Reference multiple schemas in the same document.

→ XML schemas secure Data communication,

→ when sending data from a sender to a receiver, it is essential that both parts have the same "expectations" about the content.

→ with xml Schemas, the sender can describe the data in a way that the receiver will understand.

XML Schemas :- 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 Name Namespaces.

→ It is similar to a database schema that describes the data in a database.

Syntax :- `<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">`

example : the following example shows how to use schema :

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns="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>
```

Elements :- ~~As we saw XSD~~ An elements can be defined within an XSD as follows :

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

Definition Types :-

you can define XML schema elements in following way

① simple type
② complex type
③ Global type.

Simple type :- Simple type element is used only in the context of the text.

→ Some of predefined simple types are :

```
xs:integer
xs:boolean
xs:string
xs:date
```

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

Complex Type:- A complex type is a container for other element definations.

→ 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="company" type="xs.int" />
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

→ In the above example, Address element consists of child elements.

→ This is a container for other <xs:element> definations, that allows to build a simple hierarchy of elements in the xml document.

Global Types:- with 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 & company for different addresses of the company.

→ In such case, you can define a general type as below;

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

Xml Dom (Document Object Model)

The Dom defines a standard for accessing and manipulating documents.

→ The XML Dom is a standard for how to get, change, add and delete xml elements.

→ The Dom is an application programming interface (API) for HTML & XML documents

→ It defines the logical structure of documents and the way a document is accessed and manipulated.

→ Dom defines the objects and properties & methods (interface) to access all xml elements.

→ The DOM is separated into 3 different parts / levels.

- → Core Dom → standard model for any structured document.
- → XML Dom → Standard model for XML documents.
- → HTML Dom → Standard model for HTML documents.

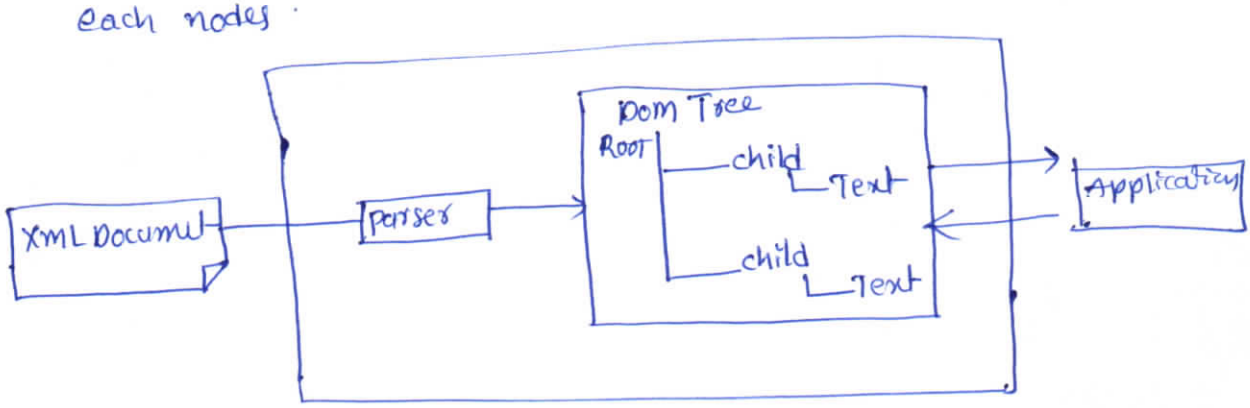
→ XML Dom is a standard object model for XML.

→ XML documents have a hierarchy of informational units called nodes.

→ Dom is a standard programming interface of describing those nodes and the relationships between them.

→ As XML Dom also provides An Api that allows a developer to add, edit, move(or) remove nodes at any point on the tree in order to create an application.

→ Below is the diagram for the Dom structure which depicts that parser evaluates an XML document as a Dom structure by traversing through each nodes.



Advantages:

→ XML Dom is language and platform independent.

→ XML Dom is traversible information in XML Dom is organized in a hierarchy which allows developers to navigate around the hierarchy looking for specific information

→ XML DOM is modifiable - it is dynamic in nature providing developer a scope to add, edit, move or remove nodes at any point on the tree.

Disadvantages:-

→ It consumes more memory (if the xml structure is large)

→ Due to the larger usage of memory its operational speed, compared to SAX is slower.

→ A DOM document is a collection of nodes or pieces of information, organized in a hierarchy.

→ Some types of node may have child nodes of various types and others are leaf nodes that cannot have anything below them in the document structure

→ Below is a list of the node types, and which node types they may have as children:

→ Document -- Element

→ DocumentFragment -- Element

→ Entity Reference -- Element

→ Element

→ Attr -- Text, EntityReference

→ ProcessingInstruction -- no children

→ comment -- no children

→ Text -- no children

→ CDATASection -- no children

→ Entity -- Element

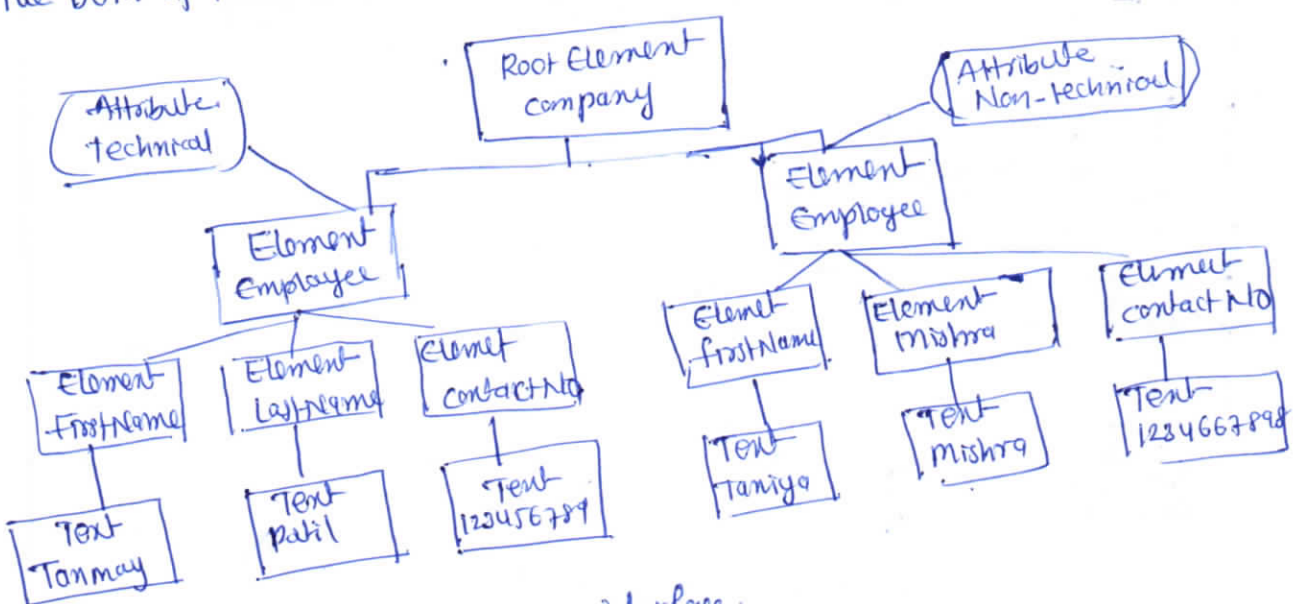
→ Notation -- no children

Example :-

Consider the DOM representation of the following xml document node.xml

```
<?xml version="1.0">
<company>
    <Employee category="technical">
    <firstName> Tanmay </firstname>
    <lastName> patil </lastName>
    <contactNO> 1234567899 </contactNO>
    </Employee>
    <Employee category="non-technical">
    <firstname> Taniya </firstName>
    <lastName> Mishra </lastName>
    <contactNO> 1234667898 </contactNO>
    </employee>
</company>
```

→ The DOM of the above xml document would be as follows :



→ from the above diagram we can interface.

→ Node object can have only one parent node object. This occupies the position above all the nodes Here it is company.

→ The parent node can have multiple nodes called as child nodes.

→ This child nodes can have additional node called as attribute node. In the above example we have two attribute nodes Technical & Non Technical

→ The attribute node is not actually a child of the element node, but is still associated with it

→ These child nodes in turn can have multiple child nodes. The text within the nodes is called text nodes

→ The node objects at the same level are called as siblings.

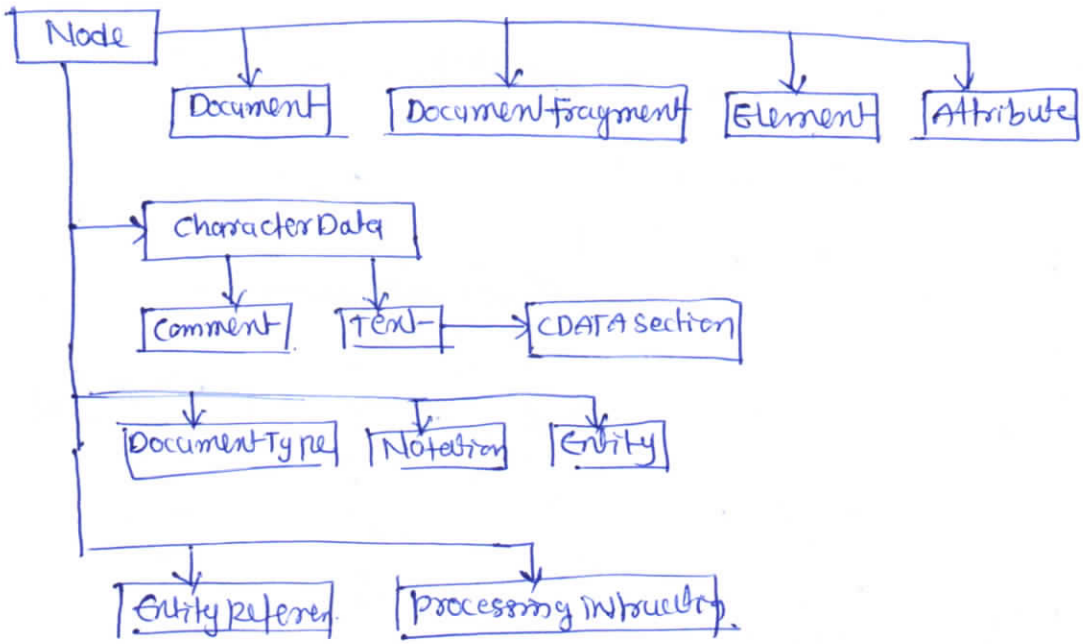
→ The DOM identifies
  → The objects to represent the interface and ~~multiple~~ manipulate this document.
→ The relationship among the objects and interfaces.

DOM nodes :-

→ Every xml DOM contains the information in hierarchical units called nodes. and the DOM describes these nodes and the relationship b/w them.

Node types

→ The following pictorial representations lists all the node types.



→ The most common types of nodes in xml are.
  → Document Node :- complete xml document structure is a document node
  → Element Node :- Every xml element is an element node. This is also the only type of node that can have attributes.
  → Attribute Node :- Each attribute is considered as an attribute node. They contain information about an element node, but are not actually considered to be children of the element.
  → TEXT Node :- The document texts are considered as text node. It can consist of more information or just white space.

XML DOM - methods :-

→ DOM as an API contains interfaces that represent diffent types of information that can be found in a xml Document, such as elements and Text.

→ These interfaces include the methods & properties necessary to work with these objects

→ properties define the characteristic of the node whereas methods giuss the way to manipulate the nodes.

following are the list of classes and interfaces:

DOMImplementation :- It provides a no of methods for performing operations that are independent of any particular instance of the document object model

DocumentFragment :- It is the 'lightweight' or minimal document object, and it is a superclass of the document

Document :- It represents the xml document's top-level node, which provids access to all the nodes in the document, including the root element.

Node :- It represents XML node

Node list :- It represels a read-only list of Node objects.

NamedNodeMap :- It represents collection of nodes that can be accessed by Name

Data :- It extends Node with a set of attributes and methods for accessing Character data in the DOM.

Attribute :- It represents an attribute in an element object.

Element :- It represents element node.

Text :- It represents text node.

comment : It represents comment node.

processingInstruction :- represents a "processing instruction" It is used in XML as a way to keep processor-specific information in the text of the document

CDATA section : Represents CDATA section. Drivers from text.

Entity : It represents an entity. Drivers from Node

EntityReference : This represents an entity reference in the tree. Drivers from Node

→ XHTML :

→ XHTML stands for Extensible Hyper Text Markup language.

→ It is next step in the evolution of the internet.

→ The XHTML 1.0 is the first document type in the XHTML family

→ XHTML is almost identical to HTML 4.01 with only few differency.

→ XHTML was developed by world wide web consortium (W3C) to help web developers make the transition from HTML to XML.

→ By migrating to XHTML today, web developers can enter the XML world with all of its benefits, while still remaining confident in the background and future compatibility of the document content.

Why use XHTML?

Developers who migrate their content to XHTML 1.0 get the following benifits

→ XHTML documents are XML conforming as they are readly, viewed, edited, and validated with standard XML tools.

→ XHTML documents can be written to operate better than they did before In existing browsers as well as in new browsers.

→ XHTML documents can utilize applications such as scripts and applets that rely upon either the HTML Document object model or the XML DOM.

→ XHTML gives you a more consistent, well-structured format so that your webpages can be easily parsed and procured by parent and future web browsers.

→ You can easily maintain, edit, convert and format your document in the long run.

→ XHTML combines strength of HTML & XML.

XHTML - syntax :-

→ XHTML syntax is very similar to HTML syntax and almost all the valid HTML elements are valid in XHTML as well.

→ But when you write an XHTML document, you need to pay a bit extra attention to make your HTML document compliant to XHTML.

→ Here are the important point to remember while writing new XHTML document or converting existing HTML document into XHTML document.

- → Write a DOCTYPE declaration at the start of the XHTML document.
- → write all XHTML tags and attributes in lower case only.
- → close all XHTML tags properly
- → Nest all the tags properly
- → Quote all the attribute value.
- → Replace the name attribute with the id attribute.

DOCTYPE Declaration :- ① Strict, ② transitional, ③ frameset

→ All XHTML document must have a DOCTYPE declaration at the start.

→ There are three types of DOCTYPE declarations;

Example :

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

Case Sensitivity :-

→ XHTML is case sensitive markup language

→ All the XHTML tags and attributes need to be written in lower case only.

Ex. `<a href="/xhtml/xhtml_tutorial.html">XHTML Tutorial</a>`

Closing The tags :-

→ Each and every XHTL tag should have an equivalent closing tag, even empty elements should also have closing tags.

`<img src="/images/xhtml.gif" />`

Attribute Quotes :-

All the values of XHTML attributes must be quoted. otherwise your XHTML document is assumed as an invalid document.

Syntax :-

Attribute minimization :-

→ XHTML does not allow attribute minimization, It means you need to explicitly state the attribute and it's value.

<!--- This is invalid in XHTML -->

<option selected>

<! This is valid in XHTML --->

<option selected = "selected" >

→ List of the a minimization attributes in XHTML and the way you need to write them in XHTML.

HTML style	XHTML style
compact	compact = "compact"
checked	checked = "checked"
declare	declare = "declare"
readonly	readonly = "readonly"
disabled	disabled = "disabled"
selected	selected = "selected"
defer	defer = "defer"
ismap	ismap = "ismap"
nohref	nohref = "nohref"

→ The id attribute :-

The id attribute replaces the name attribute., Instead of using name = "name", XHTML prefers to use id = "id" the following example shows how.

<!--- This is invalid in XHTML -->

<!-- correct XHTML way of writing this is as follows -->

The language Attribute:

→ The language attribute of the script tag is deprecated. The

Ex <-- invalid xhtml>

```
<script language = "javaScript" type = "text/javascript">
  document.write ("Hello XHTML!");
</script>
```

```
<!--- correct xhtml -->
  <script ~~language~~ type = "text/javascript" >
    document.write ("Hello XHTML!");
  </script>
```

Element prohibitions:

→ The following elements are not allowed to have any other elements inside them

→ The prohibition applies to all depths of nesting. means, it includes all the decending elements.

element	prohibition
<a>	must not contain other <a> element
<pre>	must not contain the <img>, <object>, <big>, <small>, <sub> or <sup> elements.
<button>	must not contain the <input>, <select>, <textarea>, <label>, <button> <form>, <fieldset>, <iframe> or <isindex> elements.
<label>	must not contain other <label> elements.
<form>	must not contain other <form> elements.

XHTML-Events :- When users visit a website, they do things such as click on text, images and hyperlinks.

→ These are examples of what javascript calls events.

→ We can write our event handler in javascript (or) vbscript.

→ And can specify these event handlers as a value of event tag. attribute.

→ The <body> & <frameset> level events.

→ This two events occurred at document level.

→ onload → script → script runs when a XHTML document loads

→ unload " " " " " unloads

→ Script → is refers to any function (or) piece of code of javascript.

→ Form level events

onchange → script → script executes when the element changes

onsubmit → " " " " " form is submitted

onreset → " " " " " is reset

onselect → " " " " element is selected.

onblur → " " " " loses focus.

onfocus → " " · element gets focus

Keyboard Events :-

Attribute

onkeydown → script → script executes on key press.

onkeypress → " script executes on key press & release.

onkeyup → script Script executes key release.

Mouse events (7)

onclick - script → script executes on a mouse click

ondbclick - " → " " on a mouse double-click.

onmousedown → " " when a mouse button is pressed.

onmousemove → " " " pointer moves.

onmouseout → script executes when mouse pointer moves out of an

onmouseover → script executes when mouse pointer moves over an element.

onmouseup → script executes when mouse button is released.

XHTML-Modules

Structure-module: it defines major structural elements for XHTML.

→ These elements act as the basis for the content model of any XHTML family document type

→ The elements & attributes included in this module are, body, head, html and title.

Text module: It defines all of the basic text container elements, attributes and their content model → abbr, address, blockquote, br, cite, code, dfn, dive, em, h1 to h6, kdd, p, preq, samp, span & var.

Hypertext module:- to define Hyperlinks, to other resources.

this module supports elements.

List module :- It defines list oriented elements,

it supports following ele & attr. dl, dt, dd, ol, & ul, & li

Object module

Presentation module :- It defines element, attrs & minimal content model for simple presentation-related markup. b, big, hr, i, small, sub, sup, and tt

Edit module: defines editing related markup.

del & ins.

Bidirectional text module:

form module: supports button, fieldset, form, input, label, legend select & textarea.

Table module:- caption, col, colgroup, table, tbody, td, tfoot, th, thead, and tr.

Image module: → img.

→ parsing xml Data :-

→ Dom parser loads whole xml document in memory.

→ SAX only loads small part- of xml file in memory.

→ Dom parser is faster than SAX because it accesses whole xml document in memory.

→ SAX is a event-based model, event-based means some kind of event happens to a node, like when one clicks a particular node it will give all the sub nodes rather than loading all the nodes at the same time.

→ Dom is tree model, it will load all the nodes and make the tree model.

→ Parsing xml file using Dom parser is quite fast if xml file is small but if you to read a large xml, file using Dom parser there is more chances that it will take a long time or even may not- be able to load it completely simply because it requires lot of memory to create xml Dom tree

→ Java provides support Dom parsing and you can parse xml files in java using Dom parser.

→ Dom classes are in w3c.dom package.

→ while Dom parser for java is in JAXP package (Java API for xml parsing

SAX XML parser in java:

→ SAX stands for simple API for XML parsing. This is an event based XML parsing and it parse XML file step by step so much suitable for large XML files.

→ SXA XML parser fires an event when it encountered opening tag, element or attribute, and the parsing works accordingly.

→ It's recommended to use SAX XML parser for parsing large XML files in java because it doesn't require to load whole XML file in java and it can read a big XML fill in small parts.

→ Java provides support for SAX ~~file~~ ~~in~~ parser and you can parse any XML fill in java using SAX parser.

→ Disadvantage of using SAX parser in java is that reading XML file in java using SAX parser requires more code in comparison of Dom parser.

AJAX → Asynchronous java
Asynchronous javascript and xml. It is a group of inter-related technologies like javascript, Dom, XML, HTML, CSS etc.

→ It allows you to send and recive data asynchronously without reloading the web page.

5/2/18

3, 4, 9, 10, 11, 32, 37, ~~39~~, 39, 41, 42, 46, 52, 55, 56, ~~60~~.

21/2/18 → C1, D1, D5, D6, D9, E5, E6, F2, F5, G5, G7, G9, H6, J4