XML

→ XML (Extensible Markup language). It is derived from SGML (standard generalized Markup language).
→ It is introduced in 1998 by w3c → consorse
→ XML is used for storing the data, transfering the data from one application to another application.
→ XML is "case sensitive"
→ It is "platform independent".
→ ~~the~~ ~~state~~ tags in XML should be well balanced i.e once we open the tag, it must be closed.
→ Attribute values must be enclosed within double quotes.
→ Versions of XML : 1.0 (preferred one)
1.1

XML encoding

→ Encoding is the conversion of unicode characters into binary representation.
→ XML uses UTF (~~Unicode Transfer format~~) encoding
UTF means (Universal Transformation format)
→ there are 2 types of UTF encoding
1) UTF-8 : It uses 8-bits to represent the characters
2) UTF-16 : It uses 16-bits to represent the characters

Declaration of XML

<?xml version="1.0" encoding="UTF-8" standalone="yes or no"?>

→ XML declaration is optional
→ If it is defined it should be first stmt inside the document
→ XML declaration is case sensitive. It should be written in lower case
→ XML declaration is not a tag so, we don't have closing tag
→ By default the standalone attribute value is "no"

→ In every xml document we should have <root> tag compulsory.

→ Rules for declaring the tags:

→ tag name should starts with alphabets or underscore

→ tag name should starts with alphabets, digits, hypens (--), underscore & dot.

→ white spaces are preserved in xml document

→ tag name can't contain spaces.

Ex: <company name="ABC corp" location="hyd">
<chairman> vishal </chairman>
<GM> Ron </GM>
</company>

xml Entities

→ Entities are used to represent the characters

→ Entities are also known as Entity references.

→ we can display the entity by using entity name & entity no.

→ entity name should start with &

Ex: &quote → " "
&amp → &
&lt → <
&gt → >
&apos → ' (apostrophe)

→ By using xml we store the data. To display we use either html / css.

Ex: <?xml-stylesheet type="text/css" href="first.css" ?>
<bibliography>
<book>
<author> vishal </author>
<price> 1200 </price>
</book>
</bibliography>

DTD (Document Type Definition)

→ DTD is used for validating the XML document
→ Always DTD should be written in XML
declaration
→ the element declaration should start with '!'
→ DTD name & root tag name must be same.

Types of DTD :

1) Internal DTD
2) External DTD

Internal DTD / Embedded DTD : the DTD which is defined inside the XML document.

External DTD : the DTD which is defined external/ outside to the XML document.

→ By using DTD we apply rules for the elements inside the XML document
→ XML is used to store the data & DTD is used to store metadata of XML document
→ DTD checks the XML, structure of the XML, & validate the XML

Internal DTD & syntax :

declaration of DTD :

```
<!DOCTYPE element_name element declaration >
```

Ex :

```
<!xml version="1.0" encoding="UTF-8" standalone="yes">
<!DOCTYPE student [ → declaration of DTD
<!ELEMENT student (stdid, stdname, stdage) >
<!ELEMENT stdid (#PCDATA)>
<!ELEMENT stdname(#PCDATA)>
<!ELEMENT stdage(#PCDATA)>
]> → end of declaration (DTD)
<student>
  <stdid> 100 </stdid>
  <stdname> isa </stdname>
  <stdage> 11 </stdage>
</student>
```

} body of DTD

} → XML document

PCDATA → Parsed Data means already checked

External DTD

→ In External DTD declaration we use two identifiers:
1) System – address of the DTD
2) public → "//-/www.w3cschool.com/loginpage
→ In XML declaration standalone should be no

syntax:
<!DOCTYPE element-name SYSTEM/PUBLIC >
Ex: <!DOCTYPE student SYSTEM "student.dtd">
~~student~~ system Identifiers
→ It enables to specify the external DTD inside XML doc.

public Identifiers.
→ It enables the DTD file available to the public by using the public keyword & we have to use FPI (Form Public Identifier). It is divided into four fields, each field is separated by backslash (/) & it should starts with double slash (//)
org name
Ex: <!DOCTYPE student PUBLIC "-//sun micro system/
http://www.sun.com/dtd/student.dtd">
usd

Elements
→ Element is a container which consists of text, elements, attributes & mix of all
1) plain text
2) unrestricted elements
3) empty elements
4) child elements
5) Multiple child elements

plain text:
→ If an element should contain only plain text, it should be passed by using #PCDATA.
Ex: <!ELEMENT stdname (#PCDATA)>
<stdname> ira</stdname>

2) unrestricted elements:
→ For this element we can pass any type of data.
Ex: <!ELEMENT student ANY>
→ It can be created by using content model i.e., ANY
3) Empty elements: which doesn't have any closed tag
ex: <!ELEMENT br EMPTY>

4) child element: we can specify that element can contain only one child element.
ex: <!ELEMENT student (stdid)>
<student>
<stdid>1000</stdid>
</student>
5) Multiple child elements (sequence):
→ this particular element can contain one or more child elements & separated by comma (,)
ex: <!ELEMENT student (stdid, stdname, stdage)>

Cardinally operators
1) a+ (one or more occurence of a)
2) a* (zero or more occurence of a)
3) ? (zero or one / either a or empty)
4) , (a,b a followed by b)
5) | (a|b a followed by b) / (either a or either b)
6) () (expressions
Ex: <!ELEMENT college (student*)>

Attributes
→ Attributes are the parts of the XML element
→ An element can have no. of unique attributes
→ we can pass single/multiple attributes by using <!ATTLIST> declaration
• syntax:
<!ATTLIST element-name attribute-name attribute value>
ex: <!ATTLIST student gender CDATA "male">
<student gender="male">

Attribute type :

1) #IMPLIED :- there is no default value for this attribute & this attribute is optional

2) #REQUIRED :- there is no default value but value should be assigned

3) #FIXED :- It has default value.

ex:- <!ELEMENT square EMPTY>
<!ATTLIST square width CDATA "0">
<square width="100"/>

ex:- <!ATTLIST student gender CDATA #REQUIRED>
<student gender="male"/> → valid
<student/> → not valid

ex:- <!ATTLIST student gender CDATA #IMPLIED>
<student gender="male"/> → valid
<student/> → valid.

ex:- <!ATTLIST student technology CDATA #FIXED "java">
<student technology="java"/> → valid
<student technology="ai"/> → invalid.

Disadvantages in DTD :

1) It supports only #PCDATA datatype
2) It doesn't follow XML syntax
3) Cardinally operators can't be effectively specified.
4) DTD doesn't support namespaces

XML Schema Definition (XSD) :

→ XSD is used for validating the XML document
→ It is the extension of DTD
→ DTD support only #PCDATA datatype but XSD support all the datatypes.
→ DTD doesn't support namespaces but XSD support namespaces
→ XSD is written in XML

Syntax for creating xml schema:

```
<?xml version="1.0" creading="UTF-8">
<xs:schema xmlns:xs="https://www.w3c.com/2001/
xmlschema" ..... </xs:schema>
```

→ we can define elements in XSD in 3 ways:

1) simple
2) complex type
3) global.

1) **Simple element**: It is an xml element that contains only text & it doesn't contain other elements/attributes

syntax:

```
<xs:element name="name of the element" type="type
of the element">
```

2) **Complex element**: It will have sequence of attributes with different different datatypes.

Built in datatypes in XSD

1) xs:decimal
2) xs:string
3) xs:integer
4) xs:boolean
5) xs:date
6) xs:time

ex: xml doc

```
<student>
  <stdno>100</stdno>
  <stdname>isa</stdname>
  <stdage>10</stdage>
  <dob>24-11-2024</dob>
</student>
```

XSD

```
<?xml version="1.0" creading="UTF-8">
<xs:schema xmlns:xs="">
<xs:complextype>
<xs:sequence>
<xs:element name="stdno" type="
xs:integer">
<xs:element name="stdname" type=
"xs:string">
<xs:element name="dob" type=
"xs:date">
</xs:sequence>
</xs:complextype>
```

Restrictions on the elements

- Ex: <xs:element name="stdname" type="xs:string"
  DEFAULT="RAJ" FIXED="IRA">

1) we can apply restrictions on the xml elements/attributes based on the user i/p values

Ex: <xs:element name="stdage">
  <xs:simpletype>
  <xs:restriction base="xs:integer">
  <xs:minInclusive value="16">
  <xs:maxexclusive value="60">
  </xs:restriction>
  </xs:simpletype>
  </xs:element>.

2) **enumerate** :- To limit the content of xml element to a set of acceptable values.

→ we use enumerate/enumeration constraint

Ex: <xs:restriction base="xs:string">
  <xs:enumeration value="java">
  <xs:enumeration value="python">

3) pattern :

Ex: <xs:restriction base="xs:string">
  <xs:pattern value="[a-z]">
  <xs:pattern value="male|female">

1) enumeration :- define list of i/p values
2) length :- defines exact no. of characters / list of items allowed must be equal to zero / greater than zero
3) maxExclusive :- specifies upper bounds for numeric values means value should be less than its value
4) maxInclusive :- value should be less than / equal to its value.
5) minExclusive :- value should be greater than its value
6) minInclusive :- value should be greater than / equal to its value
7) pattern :- exact sequence of characters.
8) totalDigits :- exact no. of digits allowed
9) whitespace :- allowed white spaces, tabs, space, carriage return, line feed.

Parsers

→ parsers acts as a mediator / interface b/w client application & XML document.

→ By using parsers we access & manipulate data inside XML document.

→ XML uses two parsers for processing:

1) DOM (Document object Model)

2) SAX (simple Access for XML / simple API for XML)

1) DOM.

→ DOM present in XML document as a tree structure

→ DOM document is a collection of nodes. we can add, delete & move the nodes

→ DOM is a "read & write parser"

→ It is slow compared to SAX.

→ complete XML document will be loaded into memory before we read.

2) SAX :

→ It is alternate approach for DOM

→ It is a "read only parser"

→ It will not load complete XML document, It will parse XML document line by line triggering event based.

→ It is faster.

→ "JAX.B" parser is used for java application.