Unit-5 [eXtensible Markup Language]

❖ Introduction to XML

- XML stands for extensible markup language.
- A markup language is a set of codes, or tags, which describes the text in a digital document.
- It is designed to store and transport data.
- It was designed to be self-descriptive
- XML became a W3C Recommendation on February 10, 1998.
- It is not a replacement for HTML.
- It is designed to carry data, not to display data.
- XML tags are not predefined. You must define your own tags.
- XML is platform independent and language independent.

❖ Features OR Advantage of XML:

➢ XML can be used with existing protocols:

- Many current XML messaging proposals use existing application layer protocols such as SMTP, HTTP.

➢ XML simplifies data sharing:

- In the real world, computer systems and databases contain data in incompatible formats. XML data is stored in plain text format. This provides a software- and hardware-independent way of storing data.

➢ XML simplifies data transport:

- One of the most time-consuming challenges for developers is to exchange data between incompatible systems over the Internet.
- Exchanging data as XML greatly reduces this complexity, since the data can be read by different incompatible applications.

➢ XML simplifies Platform change:

- Upgrading to new systems (hardware or software platforms), is always time consuming. Large amounts of data must be converted and incompatible data is often lost.
- XML data is stored in text format. This makes it easier to expand or upgrade to new operating systems, new applications, or new browsers, without losing data.

➢ Support a wide variety of applications:

- XML has found wide application. Today, various programs and devices use it to handle, structure, and store, transmit, and display data.

➢ Compatible with SGML(Standard Generalized Markup Language):

- Since XML is simply a subset of SGML

➢ XML documents are reasonably clear to the lay person

- It is easy to use, read, write and understand to the people.

❖ HTML Vs XML

HTML	XML
HTML stands for Hypertext Markup Language	XML stands for Extensible Markup Language
It is a predefined markup language.	It is a framework for specifying markup languages.
It is used to display data.	It is used to transport data.
It is static.	It is dynamic.
Tags are predefined	Tags are user defined
A limited number of tags are available.	Tags are extensible.
It is not necessary to use closing tags (but recommended to use closing tags).	Closing tags are mandatory.
Namespace not supported.	Namespace supported.
Tags are not case-sensitive.	Tags are case-sensitive.
White space cannot preserve (can ignore white space).	White space preserved (cannot ignore white space).
Filename extension **.html or .htm**	File extension **.xml**

❖

❖ Structure of XML:

➢ Logical Structure:

- The logical structure of an XML document consists following things:
  - Declarations
  - Tags and Elements
  - Attributes
  - Comment
- **Declaration:**
  - XML declaration consists of the XML version, character encoding or/and standalone status. The declaration is optional.
  - Syntax:

```
<?xml version="version_number," encoding="character_encoding" standalone="yes_or_no" ?>
```

  - **XML Declaration Rules**
    - ♦ If the XML declaration is present, it must be the first thing that appears.
    - ♦ The XML declaration is case sensitive, and it must start with the lowercased <?xml.
    - ♦ It has no closing tag.
  - **Example:**

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

- **Tags & Elements:**

- Tags work as pairs except for declarations. Every tag pair consists of an **opening tag** (also known as the **start tag**) and a **closing tag** (also known as the **end tag**).
- Tag names are enclosed in <>. For a particular tag pair, the start and end tags must be identical except the end tag has / after the <.

```
<name>...</name>
```

- Anything between the opening and closing tags is referred to as content.
- Opening tag, content, and closing tag, altogether, is referred to as an element.

```
Opening tag + content + closing tag = an element
```

- Example:

```
<age>20</age>
```

  - In the above element,
    - **age** is the name of the element.
    - **<age>** – opening tag
    - **25** – content
    - **</age>** – closing tag.
- **XML Tag and Element Rules**
  - Tags are case sensitive.
  - All XML documents must contain a single root element.
  - All elements must have a closing tag (except for declarations).
  - A tag name must begin with a letter or an underscore, and it cannot start with the **XML**.
  - A tag name can contain letters, digits, hyphens, underscores, and periods. Hyphens underscore, and periods are the only punctuation marks allowed.
  - A tag name cannot contain spaces.
  - All elements must be nested properly.

Attributes

- Attribute for an element is placed after the tag name in the start tag. You can add more than one attribute for a single element with different attribute names.
- **Let's consider the below XML document.**

```
<company name="ABC Holdings" location="London">
    <chairman>Mr. John</chairman>
    <gm>Mr. Wood</gm>
</company>
```

  - There are two attributes in the **company** element, i.e. **name** and **location**.
  - Let's study the **name** attribute,

- **name** – attribute name
- **ABC Holdings** – attribute value
- An **attribute name** is also known as an **attribute**.

- **XML Attribute Rules:**
  - Attribute values must be within quotes.
  - An element cannot contain several attributes with the same name.

- **Comment:**
  - Comments are optional. Adding comments help to understand the document content.
  - **Syntax for XML Comments**
    - A comment begins with **<!–** and ends with **–>.**

```
<!-- Add your comment here -->
```

❖ What is DTD?

- DTD stands for **Document Type Definition**.
- It defines the legal building blocks of an XML document.
- It is used to define document structure with a list of legal elements and attributes.
- XML DTD is used to check whether an XML document is “well formed” and “valid”.
- Declaration of DTD
  - Internal DTD Declaration
  - External DTD Declaration

➢ Internal DTD Declaration

- If the DTD is declared inside the XML file, it must be wrapped inside the <!DOCTYPE> definition:
- **Syntax:**

```
<!DOCTYPE root-element [element-declarations]>
```

- **Example:**

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

<address>
   <name>SK SINGH</name>
   <company>Monarch Softtech</company>
   <phone>(011) 123-4567</phone>
</address>
```

- **The DTD above is interpreted like this:**
  - !DOCTYPE **address** defines that the **root** element of this document is address.
  - !ELEMENT **address** defines that the "**address"** element must contain three elements: "name, company, phone".

- !ELEMENT **name** defines the "**name"** element to be of type "#PCDATA"
- !ELEMENT **company** defines the "**company"** element to be of type "#PCDATA"
- !ELEMENT **phone** defines the "**phone"** element to be of type "#PCDATA"

- **Rules:**
  - The document type declaration must appear at the start of the document (preceded only by the XML header) - it is not permitted anywhere else within 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 Declaration**

- 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 reference it as external DTD, standalone attribute in the XML declaration must be set as no.
- This means, declaration includes information from the external source.
- Syntax:

```
<!DOCTYPE root-element SYSTEM "file-name">
```

- **Example:**
  - address.xml file

```
<?xml version = "1.0" encoding = "UTF-8" standalone = "yes" ?>
<!DOCTYPE address SYSTEM "address.dtd">

<address>
   <name>SK SINGH</name>
   <company>Monarch Softtech</company>
   <phone>(011) 123-4567</phone>
</address>
```

  - address.dtd

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

- **Types:**
  - You can refer to an external DTD by either using **system identifiers** or **public identifiers**.
  - **System Identifiers:**
    - ♦ A system identifier enables you to specify the location of an external file containing DTD declarations. **Syntax is as follows:**

```
<!DOCTYPE name SYSTEM "address.dtd" [...]>
```

    - ♦ As you can see it contains 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 "-//Beginning XML//DTD Address Example//EN">
```

  - As you can see, it begins with keyword PUBLIC, followed by a specialized identifier.

❖ Entity

- Entities are used to define shortcuts to special characters.
- Entities can be declared internal or external.
- **Types of Entity are:**
  - Built-in Entity
  - Character Entity
  - General Entity
- **Built-in Entity:**
  - All XML parsers must support built-in entities.
  - In general, you can use these entity references anywhere.
  - You can also use normal text within the XML document, such as in element contents and attribute values.
  - here are five built-in entities that play their role in well-formed XML, they are:
    - ampersand: **&**
    - Single quote: **'**
    - Greater than: **>**
    - Less than: **<**
    - Double quote: **"**
  - **Example:**

```
<?xml version = "1.0"?>
<note>
   <desc>I'm a Teacher &amp; Programmer</desc>
<note>
```

    -

- **Character entities**
  - Character Entities are used to name some of the entities which are symbolic representation of information i.e characters that are difficult or impossible to type can be substituted by Character Entities.
  - Example:

```
<?xml version = "1.0" encoding = "UTF-8" standalone = "yes"?>
<!DOCTYPE author [
   <!ELEMENT author (#PCDATA)>
   <!ENTITY writer "Tanmay patil">
   <!ENTITY copyright "&#169;">
]>
<author>&writer;&copyright;</author>
```

  - You will notice here we have used **©** as value for copyright character. Save this file as sample.xml and open it in your browser and you will see that copyright is replaced by the character ©.

➢ General Entity:

- General entities must be declared within the DTD before they can be used within an XML document.
- Instead of representing only a single character, general entities can represent characters, paragraphs, and even entire documents.
- Example:

```
<?xml version = "1.0"?>
<!DOCTYPE note [
   <!ENTITY source-text "tutorialspoint">
]>

<note> &source-text; </note>
```

- Whenever an XML parser encounters a reference to source-text entity, it will supply the replacement text to the application at the point of the reference.

❖ Elements Content Model

➢ Element Sequences:

- The sequence element specifies that the child elements must appear in a sequence. Each child element can occur from 0 to any number of times.
- **Syntax:**

```
<sequence
id=ID
maxOccurs=nonNegativeInteger|unbounded
minOccurs=nonNegativeInteger
any attributes
>

(annotation?,(element|group|choice|sequence|any)*)

</sequence>
```

- Example: This example shows a declaration for an element called "personinfo", which must contain the following five elements in order; "firstname", "lastname", "address", "city", and "country":

```
<xs:element name="personinfo">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="firstname" type="xs:string"/>
      <xs:element name="lastname" type="xs:string"/>
      <xs:element name="address" type="xs:string"/>
      <xs:element name="city" type="xs:string"/>
      <xs:element name="country" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

➢ Elements Choices:

- DTDs can support choices. By using a choice, we can specify one of a group of items.

- For example, if you want to specify that one (and only one) of either <x>, <y>, or <z> will appear, use a choice like this:

```
(x | y | z)
```

- **Example:** If you want, each product is allowed to contain either a **<price>** element or a **<discountprice>** element.

```
<!ELEMENT project (product, id, (price | discountprice))>
```

❖ Elements occurrence indicators:

➢ Discussion of three occurrence Indicators

- Question Mark (?)
- Asterisk Sign (*)
- Plus Sign (+)

➢ Question Mark (?):

- It means element can appear one or more times.
- You can use ? to specify zero or one child elements.
- Using ? indicates that a particular child element may be present once in the element you're declaring, but it need not be.
- Example:

```
<!ELEMENT company (employee)?>
```

➢ Asterisk Sign (*)

- It means element can appear zero or more times.
- Asterisk Sign * symbol to specify that you want an element to contain any number of child elements that is, zero or more child elements.
- Example:

```
<!ELEMENT company (employee)*>
```

➢ Plus Sign (+):

- It means element can appear zero or just once.
- You can use + symbol to add zero or one child element.
- Example:

```
<!ELEMENT company (employee)+>
```