XML Module

# Session 01

# 1. Introduction to XML

* XML stands for eXtensible Markup Language.  
  It is used to store and transport data in a simple and structured way.
* Self descriptive
* Used to carry data(not used to display data)
* Self defined tags
* XML uses a document type defination
* Helps in Easy communication between two platform

Application

|  |  |
| --- | --- |
| HTML | XML |
| HTML was designed to display data. | **XML was designed to carry data.** |
| HTML displays data and focuses on how data looks. | **XML describes data and focuses on what data is.** |
| HTML displays information. | **XML describes information.** |
| Not case sensitive | **Case sensitive** |
| Pre-defined tags | **Can create own tags** |

XML Exaample:  
hierarchical structure:

<?xml version="1.0" encoding="iso-8859-1" ?>

<Watch>

<name>Titan</name>

<price>$50</price>

Declaration

</watch>

Syntax:

<?xml version="1.0" encoding="iso-8859-1" ?>

- <FlowerPlanet>

<Name>Rose</Name>

<Price>$1</Price>

<Description>Red in color</Description>

<Number>700</Number>

</FlowerPlanet>

**where,**  
<Name>, <Price>, <Description> and <Number> tags are **elements**  
<FlowerPlanet> and </FlowerPlanet> are the **root elements**

Unlike HTML (which shows data on web pages), XML focuses on describing and organizing data.  
Example:  
<student>  
 <name>Ali</name>  
 <course>Web Development</course>  
 <marks>90</marks>  
</student>

Two Types Of XML :

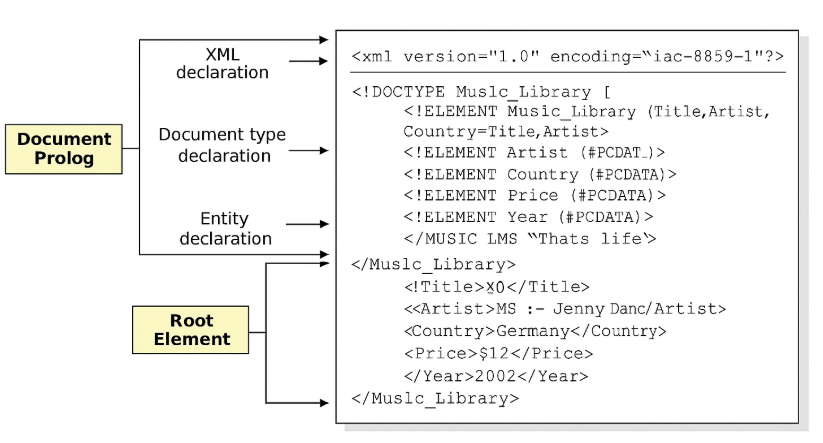
1. Structure (DTD- Document type Definition)

2. Content(XSD – XML Schema Definition)

# Lesson 02. Exploring XML

**Structure of an XML document:**

- XML uses tags which are user-defined (not pre-defined like HTML).  
- It is both human-readable and machine-readable.  
- XML is widely used in:  
 \* Web services (data exchange between applications)  
 \* Configuration files  
 \* Storing structured data



**XML DTD(Document type declaration/definition)**

#### 🔹 ****XML DTD****:

* XML Document Type Definition/Declaration
* Used to describe XML language precisely.
* Used to define structure of an XML document.
* Contains list of legal elements.
* Used to perform validation.

#### ****DTD Syntax****:

<!DOCTYPE element DTD Identifier [

declaration 1

declaration 2

]>

**Types of DTD**

#### 🟥 ****Internal****

* Elements are declared **within** the XML file.
* **Syntax**:

<!DOCTYPE root-element [

element-declaration

]>

#### 🟦 ****External****

* Elements are declared **outside** the XML file.
* **Syntax**:

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

### Internal DTD Example:

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

<!DOCTYPE Address [

<!ELEMENT Address (Name, Company, Phone)>

<!ELEMENT Name (#PCDATA)>

<!ELEMENT Company (#PCDATA)>

<!ELEMENT Phone (#PCDATA)>

]>

<Address>

<Name>...</Name>

<Company>...</Company>

<Phone>...</Phone>

</Address>

### External DTD Example:

* **DTD File**: Add.dtd

<!ELEMENT Address (Name, Company, Phone)>

<!ELEMENT Name (#PCDATA)>

<!ELEMENT Company (#PCDATA)>

<!ELEMENT Phone (#PCDATA)>

* **XML File**:

<!DOCTYPE Address SYSTEM "Add.dtd">

## What is XML Schema?

* **XML Schema** (also called **XSD – XML Schema Definition**) is a way to describe the structure and rules of an XML document.
* It defines:
  + What elements and attributes can appear.
  + The **order** of elements.
  + The **data types** (string, integer, date, etc.).
  + Default and fixed values.

it like a **blueprint** for an XML file, ensuring data follows the correct forma

## XML Schema Syntax (Basic Rules)

An XML Schema file usually:

* Starts with <?xml version="1.0"?>
* Uses <xs:schema> as the root element.
* Defines elements using <xs:element>.
* Defines complex structures using <xs:complexType> and <xs:sequence>.
* Uses xs: prefix (XML Schema namespace).

**Common Syntax Example:**

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

<!-- Define an element -->

<xs:element name="student" type="xs:string"/>

</xs:schema>

* It is like **DTD** but provides **more control** on XML structure.

### Definition Types:

#### ✅ ****Simple Type****

* Used only in the content of the text.

**Example:**

* xs:int, xs:string
* <xs:element name="Phone" type="xs:int" />

#### ✅ ****Complex Type****

* It is the container for other element definitions.
* Allows you to specify which child elements an element can contain & to provide some structure within your XML documents.

### ****e.g. of Complex Type****

# Case 1: ****Without Namespace****

### ****student.xsd****

<?xml version="1.0"?>

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

<xs:element name="student">

<xs:complexType>

<xs:sequence>

<xs:element name="id" type="xs:integer"/>

<xs:element name="name" type="xs:string"/>

<xs:element name="age" type="xs:integer"/>

<xs:element name="email" type="xs:string"/>

</xs:sequence>

</xs:complexType>

</xs:element>

</xs:schema>

### ****student.xml****

<?xml version="1.0"?>

<student xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:noNamespaceSchemaLocation="student.xsd">

<id>101</id>

<name>Zara</name>

<age>22</age>

<email>zara123@gmail.com</email>

</student>

 **complexType** → means the element has a **structure inside it** (not just text, but other child elements or attributes).

 **sequence** → means the child elements must appear in a **specific order** (for example: first <id>, then <name>, then <age>)

# Case 2: ****With Namespace**** (used in bigger systems, multiple schemas)

### ****student.xsd****

<?xml version="1.0"?>

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

targetNamespace="http://example.com/student"

xmlns="http://example.com/student"

elementFormDefault="qualified">

<xs:element name="student">

<xs:complexType>

<xs:sequence>

<xs:element name="id" type="xs:integer"/>

<xs:element name="name" type="xs:string"/>

<xs:element name="age" type="xs:integer"/>

<xs:element name="email" type="xs:string"/>

</xs:sequence>

</xs:complexType>

</xs:element>

</xs:schema>

### ****student.xml****

<?xml version="1.0"?>

<student xmlns="http://example.com/student"

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

xsi:schemaLocation="http://example.com/student student.xsd">

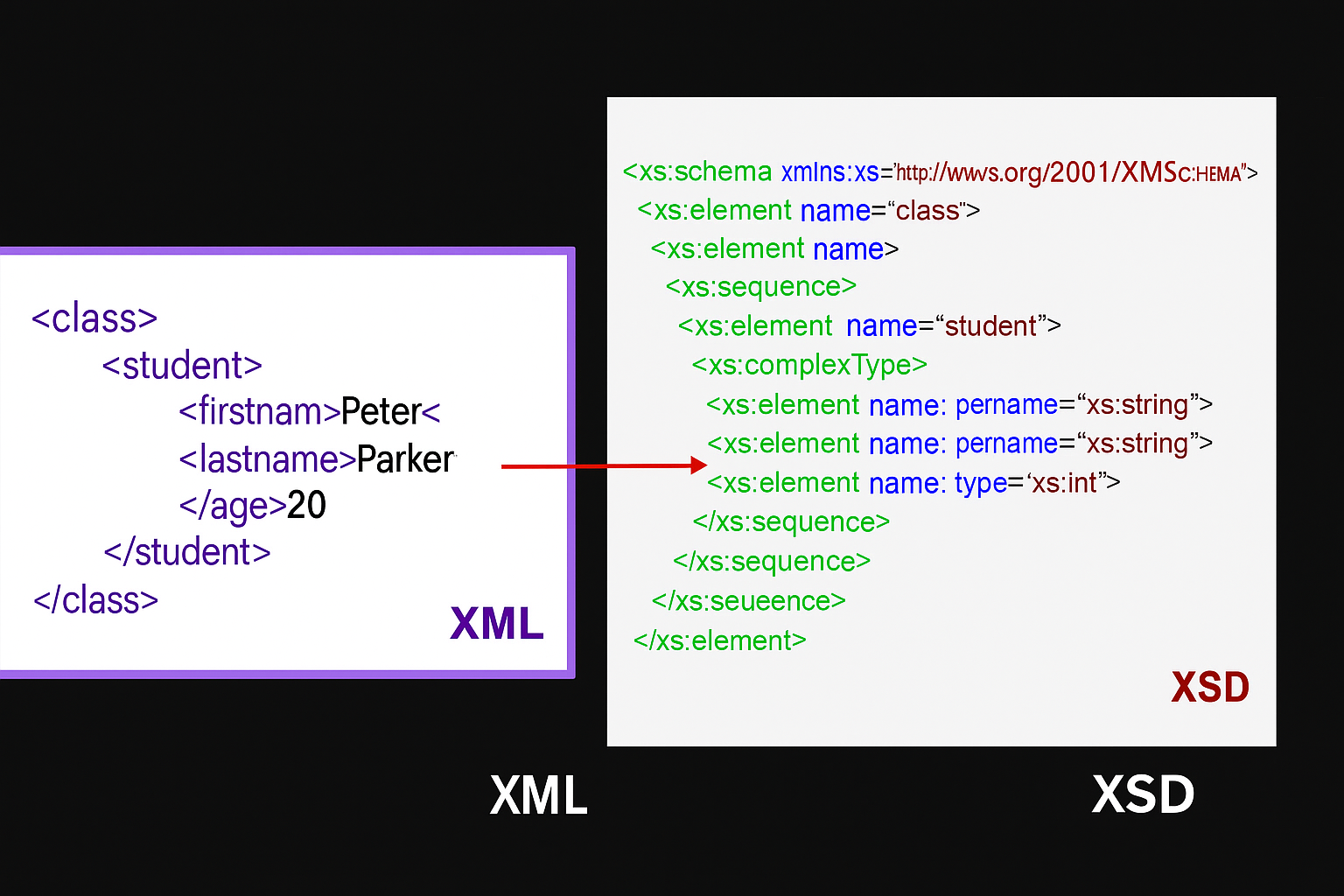
<id>101</id>

<name>Zara</name>

<age>22</age>

<email>zara123@gmail.com</email>

</student>

****

**<class>**

### <student>

### <firstname>Graham</firstname>

### <lastname>Bell</lastname>

### <age>20</age>

### </student>

### </class>

Create XML Schema:

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

<xs:element name="class">

<xs:complexType>

<xs:sequence>

<xs:element name="student">

<xs:complexType>

<xs:sequence>

<xs:element name="firstname" type="xs:string"/>

<xs:element name="lastname" type="xs:string"/>

<xs:element name="age" type="xs:int"/>

</xs:sequence>

</xs:complexType>

</xs:element>

</xs:sequence>

</xs:complexType>

</xs:element>

</xs:schema>

### <https://www.liquid-technologies.com/online-xsd-validator>

### XML Namespaces

#### 🔹 ****Purpose****:

* Used to avoid element name conflict in XML document.
* In XML, element names are defined by the developer. This often results in a conflict when trying to mix XML documents from different XML applications.
* In XML, elements are distinguished by using namespaces.
* It is a set of unique names.
* Identified by URI (Uniform Resource Identifier).
* Attribute name must start with xmlns.

#### Syntax:

<element xmlns:prefix = "URI">

* **Element and attribute prefix names** belong to the **URI**.

#### 🔹 Conflict:

* Generally, conflict occurs when we try to mix XML documents from different XML applications.

### ****CLASS 9ᵀᴴ XML FILE****

<Student>

<name> Ali </name>

<age> 22 </age>

<class> 9 </class>

</Student>

### ****CLASS 10ᵀᴴ XML FILE****

<Student>

<name> Ali </name>

<age> 23 </age>

<class> 10 </class>

</Student>

### ****Solving the Name Conflict Using a Prefix****

Name conflicts in XML can easily be avoided using a name **prefix**.

#### ****CLASS 9ᵀᴴ XML FILE****

<c9:Student>

<c9:name> Ali </c9:name>

<c9:age> 22 </c9:age>

<c9:class> 9 </c9:class>

</Student>

#### ****CLASS 10ᵀᴴ XML FILE****

<c10:Student>

<c10:name> Ali </c10:name>

<c10:age> 23 </c10:age>

<c10:class> 10 </c10:class>

</c10:Student>

***Merged XML Example (With Prefixes):***

<AllStudents xmlns:c9="http://example.com/class9" xmlns:c10="http://example.com/class10">

<c9:Student>

<c9:name>Ali</c9:name>

<c9:age>22</c9:age>

<c9:class>9</c9:class>

</c9:Student>

<c10:Student>

<c10:name>Ali</c10:name>

<c10:age>23</c10:age>

<c10:class>10</c10:class>

</c10:Student>

</AllStudents>

These prefixes (c9, c10) would typically be bound to namespace URIs using the xmlns attribute in a full XML document, like:

❌ Conflict occurs due to **same element name**.

### *🔹 Example of Namespace Usage:*

### *1.xml:*

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

<c1:class xmlns:c1="class1...">

<c1:name>Aman</c1:name>

</c1:class>

✅ Now there will be **no conflict due to namespace**.

### *2.xml:*

<c2:class xmlns:c2="class2...">

<c2:name>Aman</c2:name>

</c2:class>

**Working with XML**

- This means using XML in real applications:  
 \* Reading XML files in programming (Java, Python, PHP, etc.)  
 \* Writing or creating XML documents  
 \* Using XML in databases or with APIs  
- Example: Many websites send/receive data in XML (or JSON) when connecting apps.

# 4. XML Syntax

- Rules that must be followed while writing XML:  
 \* XML documents must have a root element (like <student>).  
 \* All tags must be properly closed (<name>…</name>).  
 \* Tags are case-sensitive (<Name> ≠ <name>).  
 \* Attributes must be in quotes (<student id="101">).  
Example of correct syntax:  
<book id="B1">  
 <title>XML Basics</title>  
 <author>John Smith</author>  
</book>

**5. Features of Markup Language:**

* Separate data from HTML
* Simplifies data sharing
* Simplifies data transport
* Increases data availability
* Simplifies platform change

**6. XML DTD : xml document type definition/declaration**

**Used to define structure of a XML document**

**SESSION 05:**

### ****XML with CSS****

**CSS** stands for **Cascading Style Sheets**. It can be used to add style and display info to an XML document.

#### ****Syntax:****

<?xml-stylesheet type="text/css" href="xml1.css"?>

#### ****Example:****

**collegeData.css**

college {

background-color: blue;

}

Name {

color: black;

}

Roll {

color: green;

}

**XML:**

<?xml-stylesheet type="text/css" href="collegeData.css"?>

<college>

<class>

<Name>Aman</Name>

<RollNo>1</RollNo>

</class>

</college>

### ****XML with XSLT:****

**XSL** → Extensible Stylesheet Language  
**XSL** is a styling language for XML.  
**XSLT**: XML Transformations  
→ It is used to transform an XML document into another XML document or another document type.

### ****What You Can Do with XSLT:****

i) Add/Remove Elements  
ii) Add/Remove Attributes  
iii) Rearrange/Sort Elements

#### ****Eg: Transforming XML into XHTML using XSLT****

<?xml version="1.0"?>

<?xml-stylesheet type="text/xsl" href="collegeData.xsl"?>

<college>

<class>

<Name>Aman</Name>

<RollNo>1</RollNo>

</class>

<class>

<Name>Zara</Name>

<RollNo>2</RollNo>

</class>

</college>

### College.xsl.xsl ****Example****

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

<xsl:stylesheet version="1.0"

xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/">

<html>

<body>

<h2>College Data</h2>

<table border="1">

<tr>

<th>Name</th>

<th>RollNo</th>

</tr>

<xsl:for-each select="college/class">

<tr>

<td><xsl:value-of select="Name"/></td>

<td><xsl:value-of select="RollNo"/></td>

</tr>

</xsl:for-each>

</table>

</body>

</html>

</xsl:template>

</xsl:stylesheet>

### XML with CSS and XSLT | How to Transform XML into XHTML using XSLT: ****XML with CSS:****

**CSS** stands for **Cascading Style Sheets**.  
It can be used to **add style** and **display info** to an XML document.

### ✅ ****Syntax:****

<?xml-stylesheet type="text/css" href="xml1.css"?>

### ✅ ****Example:**** xml1.css

College {

background-color: blue;

}

Name {

color: black;

}

Roll {

color: green;

}

### ✅ XML Example Structure:

<College>

<Name>\_\_\_</Name>

<Roll>\_\_\_</Roll>

</College>

## ****📗 XML with XSLT:****

* **XSLT** → eXtensible Stylesheet Language
* **XSL** is a styling language for XML
* **XSLT**: XML Transformations

It is used to **transform** an XML document into **another XML document**, or **another document type**.

### 🔧 ****What You Can Do with XSLT:****

i) Add / Remove Elements  
ii) Add / Remove Attributes  
iii) Rearrange / Sort Elements

Transforming XML into **XHTML** using **XSLT  
  
XML STYLE SHEET**

## Why We Use XML Style Sheet:

We use an **XML style sheet** (XSL / XSLT) because **XML only stores data, not design**.

1. **Presentation of XML Data**
   * XML is only for **data storage and transfer**.
   * A style sheet (XSLT) converts that data into a **readable and well-formatted output** (HTML, table, list, etc.).

There are mainly **two types of stylesheets** :

# 1. Internal Style Sheet in XML

* The **stylesheet is written inside the XML file itself** (embedded).
* Usually, we embed XSLT directly in the XML file.

### XML File (students.xml):

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

<?xml-stylesheet type="text/xsl" href="students.xsl"?>

<students>

<student>

<name>Tuba</name>

<course>IT</course>

</student>

<student>

<name>Ahmed</name>

<course>CS</course>

</student>

</students>

### 🔹 External XSL File (students.xsl):

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

<xsl:stylesheet version="1.0"

xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/">

<html>

<body style="font-family: Verdana;">

<h2>Students Info</h2>

<table border="1" cellpadding="5">

<tr bgcolor="#7BADD7">

<th>Name</th>

<th>Course</th>

</tr>

<xsl:for-each select="students/student">

<tr>

<td><xsl:value-of select="name"/></td>

<td><xsl:value-of select="course"/></td>

</tr>

</xsl:for-each>

</table>

</body>

</html>

</xsl:template>

</xsl:stylesheet>

| **Type** | **Meaning** |
| --- | --- |
| **Internal XML Style Sheet** | Style (XSLT) is written directly **inside the XML file** |
| **External XML Style Sheet** | Style is defined in a separate .xsl file and **linked to the XML** |