

**XML**

# XML Introduction

- XML stands for eXtensible Markup Language.
- XML was designed to store and transport data.
- XML was designed to be both human- and machine-readable.
- XML 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.

# Why xml?

- **Platform Independent and Language Independent:** The main benefit of xml is that you can use it to take data from a program like Microsoft SQL, convert it into XML then share that XML with other programs and platforms. You can communicate between two platforms which are generally very difficult.
- Many corporation use XML interfaces for databases, programming, office application mobile phones and more. It is due to its platform independent feature.

# The Difference Between XML and HTML

XML and HTML were designed with different goals:

- XML was designed to carry data - with focus on what data is
- HTML was designed to display data - with focus on how data looks
- XML tags are not predefined like HTML tags are

# XML Syntax Rules

- The syntax rules of XML are very simple and logical. The rules are easy to learn, and easy to use.

<root>

<child>

<subchild>.....</subchild>

</child>

</root>

- XML Attribute Values Must Always be Quoted

```
<note date="12/11/2007">
```

```
<to>Tove</to>
```

```
<from>Jani</from>
```

```
</note>
```

- Entity References

```
<message>salary < 1000</message>
```

```
<message>salary &lt; 1000</message>
```

- Comments in XML

The syntax for writing comments in XML is similar to that of HTML

`<!-- This is a comment -->`

- White-space is Preserved in XML

XML:	Hello	Tove
HTML:	Hello Tove	

# The XML Prolog

This line is called the XML **prolog**:

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

- The XML prolog is optional. If it exists, it must come first in the document.
- UTF-8 is the default character encoding for XML documents.

All XML Elements Must Have a Closing Tag

XML Tags are Case Sensitive

XML Elements Must be Properly Nested



- **Message:**

- I have several friends. Hari, Ramu, Latha are my friends.
  - Geetha is my best Friend.

- **XML:**

```
<friends-list>
  <friend>
    <name> Hari </name>
  </friend>
  <friend>
    <name> Ramu</name>
  </friend>
  <friend>
    <name> Latha </name>
  </friend>
  <bestfriend>
    <name>Geetha</name>
  </bestfriend>
</friends-list>
```

- **2 types of information is present:**

- Markup like <friend-list>, <friend>....
  - text or character like Hari, Ramu

# Example:

```
<?xml version="1.0" encoding="UTF-8"?>
<bookstore>
  <book category="COOKING">
    <title>Everyday Italian</title>
    <author>Giada De Laurentiis</author>
    <year>2005</year>
    <price>30.00</price>
  </book>
  <book category="CHILDREN">
    <title>Harry Potter</title>
    <author>J K. Rowling</author>
    <year>2005</year>
    <price>29.99</price>
  </book>
  <book category="WEB">
    <title>Learning XML</title>
    <author>Erik T. Ray</author>
    <year>2003</year>
    <price>39.95</price>
  </book>
</bookstore>
```

- XML based standard defines what are valid elements, using
  - XML type specification languages to specify the syntax
    - DTD (Document Type Descriptors)
    - XML Schema

# XML Validator

**Validation** is a process by which an XML document is validated. An XML document is said to be valid if its contents match with the elements, attributes and associated document type declaration(DTD), and if the document complies with the constraints expressed in it. Validation is dealt in two ways by the XML parser. They are –

- Well-formed XML document
- Valid XML document

# Well-formed XML Document

- An XML document is said to be **well-formed** if it adheres to the following rules –
- Non DTD XML files must use the predefined character entities for **amp(&)**, **apos(single quote)**, **gt(>)**, **lt(<)**, **quot(double quote)**.
- It must follow the ordering of the tag. i.e., the inner tag must be closed before closing the outer tag.
- Each of its opening tags must have a closing tag or it must be a self ending tag.(<title>....</title> or <title/>).
- It must have only one attribute in a start tag, which needs to be quoted.
- **amp(&)**, **apos(single quote)**, **gt(>)**, **lt(<)**, **quot(double quote)** entities other than these must be declared.

```
<?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>Tanmay Patil</name>
  <company>TutorialsPoint</company>
  <phone>011 123-4567</phone>
</address>
```

# Valid XML Document

If an XML document is well-formed and has an associated Document Type Declaration (DTD), then it is said to be a valid XML document.

# XML DTD

- DTD stands for **D**ocument **T**ype **D**efinition. It defines the structure of an XML document using some legal elements. XML DTD is optional.

## DTD Rules

- If DTD is present, it must appear at the start of the document (only the XML declaration can appear above the DTD).
- The element declaration must start with an ! mark.
- The DTD name and element type of the root element must be the same.



# Example of an internal DTD

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE student [
  <!ELEMENT student (firstname,lastname,school)>
  <!ELEMENT firstname (#PCDATA)>
  <!ELEMENT lastname (#PCDATA)>
  <!ELEMENT school (#PCDATA)> ]>
<student>
  <firstname>Mark</firstname>
  <lastname>Wood</lastname>
  <school>Hills College</school>
</student>
```

- **!DOCTYPE student** indicates the beginning of the DTD declaration. And the **student** is the root element of the XML document.
- **!ELEMENT student** indicates the **student** element must contain **firstname**, **lastname** and **school** elements.
- **!ELEMENT firstname** indicates the **firstname** element is of type **#PCDATA** (**P**arsed **C**haracter **D**ata).
- **!ELEMENT lastname** indicates the **lastname** element is of type **#PCDATA**.
- **!ELEMENT school** indicates the **school** element is of type **#PCDATA**.

# Example of an external DTD

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE student SYSTEM "student.dtd">
<student>
    <firstname>Mark</firstname>
    <lastname>Wood</lastname>
    <school>Hills College</school>
</student>
```

**The DTD file content (student.dtd) as follows.**

```
<!ELEMENT student (firstname,lastname,school)>
<!ELEMENT firstname (#PCDATA)>
<!ELEMENT lastname (#PCDATA)>
<!ELEMENT school (#PCDATA)>
```

# XML Schema

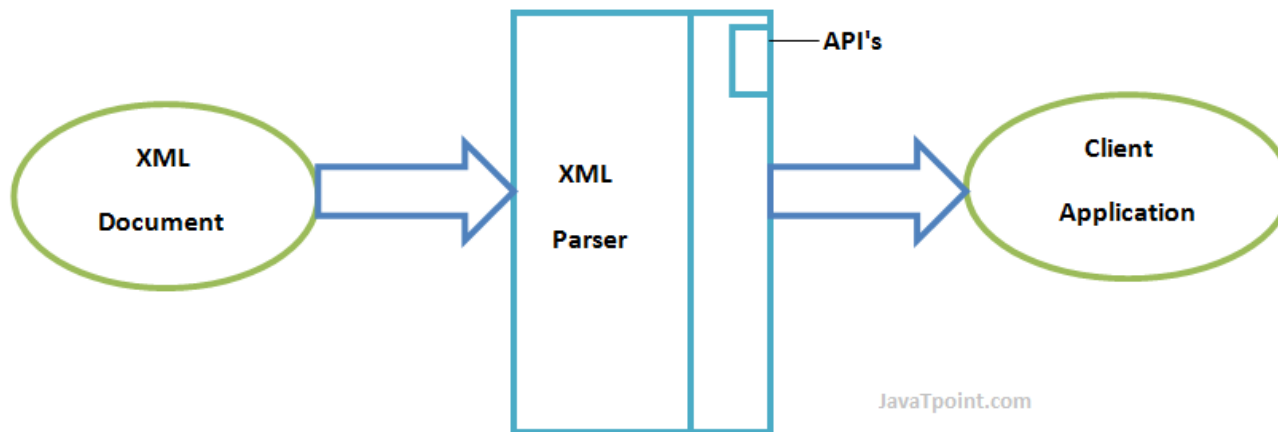
XML Schema is commonly known as **XML Schema Definition (XSD)**. It is used to describe and validate the structure and the content of XML data.

- XML schema defines the elements, attributes and data types.
- Schema element supports Namespaces. It is similar to a database schema that describes the data in a database.

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

# XML Parsers

- An XML parser is a software library or package that provides interfaces for client applications to work with an XML document. The XML Parser is designed to read the XML and create a way for programs to use XML.
- XML parser validates the document and check that the document is well formatted.



# Types of XML Parsers

- These are the two main types of XML Parsers:
  - DOM
  - SAX

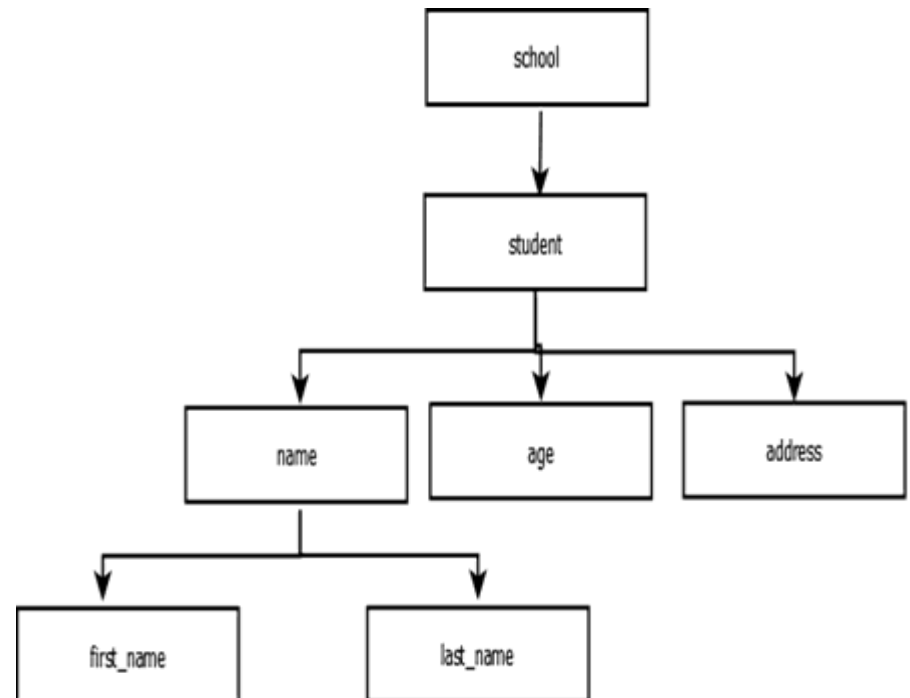
# **XML DOM**

DOM stands for **D**ocument **O**bject **M**odel. It defines a standard manner of accessing and manipulating XML documents. DOM has a (hierarchical) tree structure.



# Example of DOM

```
<?xml version="1.0" encoding="UTF-8" ?>
<school>
  <student>
    <name>
      <first_name>Alex</first_name>
      <last_name>Clarke</last_name>
    </name>
    <age>14</age>
    <address>No. 35, Flower Road,
      Hyderabad</address>
  </student>
</school>
```



# SAX (Simple API for XML)

- A SAX Parser implements SAX API. This API is an event based API and less intuitive.

## **Features of SAX Parser**

- It does not create any internal structure.
- Clients does not know what methods to call, they just overrides the methods of the API and place his own code inside method.
- It is an event based parser, it works like an event handler in Java.

## **Advantages**

- 1) It is simple and memory efficient.
- 2) It is very fast and works for huge documents.

# XML Namespaces

- XML Namespaces provide a method to avoid element name conflicts.

## **Name Conflicts**

- 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.
- This XML carries HTML table information:

```
<table>
  <tr>
    <td>Apples</td>
    <td>Bananas</td>
  </tr>
</table>
```

This XML carries information about a table (a piece of furniture):

```
<table>  
  <name>African Coffee Table</name>  
  <width>80</width>  
  <length>120</length>  
</table>
```

# Solving the Name Conflict Using a Prefix

- Name conflicts in XML can easily be avoided using a name prefix.
- This XML carries information about an HTML table, and a piece of furniture:

```
<h:table>  
  <h:tr>  
    <h:td>Apples</h:td>  
    <h:td>Bananas</h:td>  
  </h:tr>  
</h:table>
```

```
<f:table>  
  <f:name>African Coffee Table</f:name>  
  <f:width>80</f:width>  
  <f:length>120</f:length>  
</f:table>
```

# XML Namespaces - The xmlns Attribute

- When using prefixes in XML, a **namespace** for the prefix must be defined.
- The namespace can be defined by an **xmlns** attribute in the start tag of an element.
- The namespace declaration has the following syntax. *xmlns:prefix="URI"*.

- `<root xmlns:h="http://www.w3.org/TR/html4/"  
xmlns:f="https://www.w3schools.com/furniture">`

`<h:table>`

`<h:tr>`

`<h:td>Apples</h:td>`

`<h:td>Bananas</h:td>`

`</h:tr>`

`</h:table>`

`<f:table>`

`<f:name>African Coffee Table</f:name>`

`<f:width>80</f:width>`

`<f:length>120</f:length>`

`</f:table>`

`</root>`

# XSLT

- **E**Xtensible **S**tylesheet **L**anguage **T**ransformation commonly known as XSLT is a way to transform the XML document into other types of document.

## Advantages

- Independent of programming. Transformations are written in a separate xsl file which is again an XML document.
- Output can be altered by simply modifying the transformations in xsl file. No need to change any code. So Web designers can edit the stylesheet and can see the change in the output quickly.



## Student.xml

```
<?xml version = "1.0"?>
<?xml-stylesheet type = "text/xsl" href = "students.xsl"?>
<class>
  <student rollno = "393">
    <firstname>Dinkar</firstname>
    <lastname>Kad</lastname>
    <nickname>Dinkar</nickname>
    <marks>85</marks>
  </student>
  <student rollno = "493">
    <firstname>Vaneet</firstname>
    <lastname>Gupta</lastname>
    <nickname>Vinni</nickname>
    <marks>95</marks>
  </student>
  <student rollno = "593">
    <firstname>Jasvir</firstname>
    <lastname>Singh</lastname>
    <nickname>Jazz</nickname>
    <marks>90</marks>
  </student>
</class>
```

```

<?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>Students</h2>

    <table border = "1">
      <tr bgcolor = "#9acd32">
        <th>Roll No</th>
        <th>First Name</th>
        <th>Last Name</th>
        <th>Nick Name</th>
        <th>Marks</th>
      </tr>
<xsl:for-each select="class/student">
      <tr>
        <td>
<xsl:value-of select = "@rollno"/>

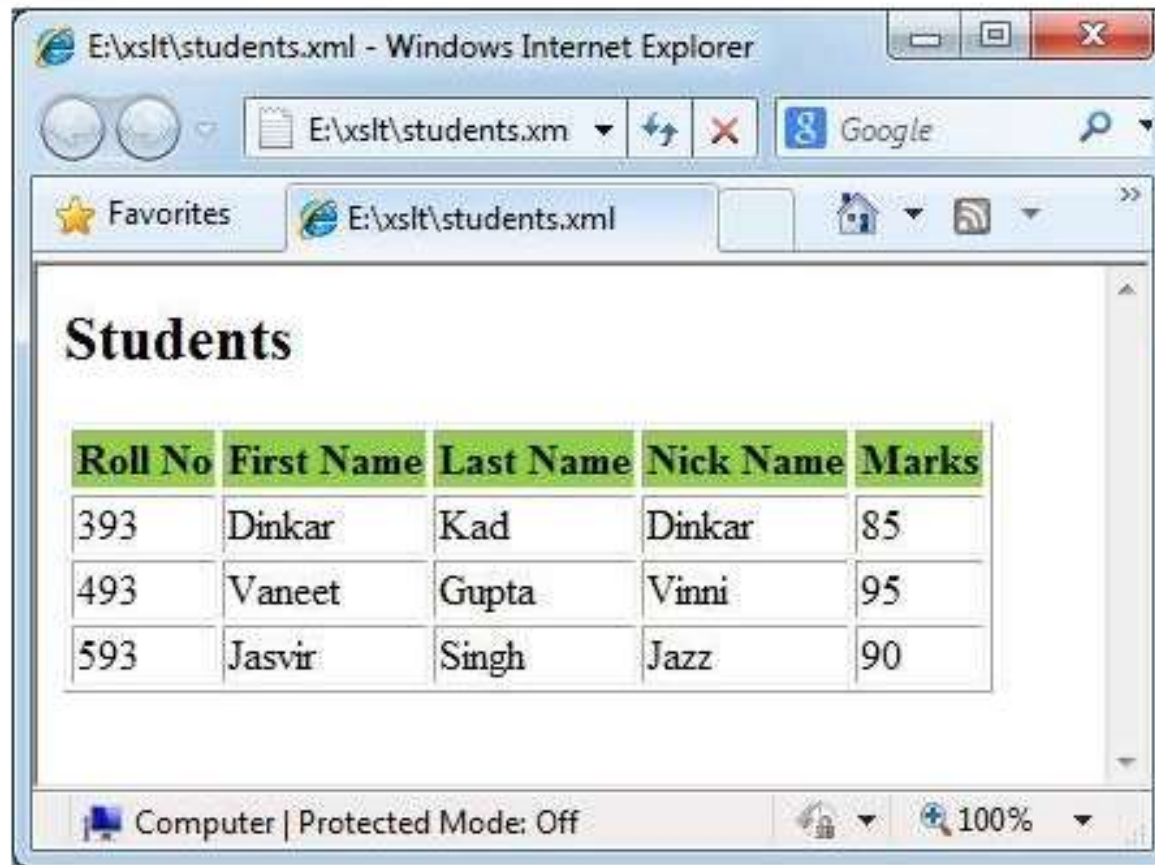
        <td><xsl:value-of select = "firstname"/></td>
<td><xsl:value-of select = "lastname"/></td>
        <td><xsl:value-of select = "nickname"/></td>
        <td><xsl:value-of select = "marks"/></td>

      </tr>
</xsl:for-each>

    </table>
  </body>
</html>
</xsl:template>
</xsl:stylesheet>

```

OUTPUT:



**Students**

Roll No	First Name	Last Name	Nick Name	Marks
393	Dinkar	Kad	Dinkar	85
493	Vaneet	Gupta	Vinni	95
593	Jasvir	Singh	Jazz	90