**XML**

* eXtensible Markup Language
* which is made up of
  + Tag
  + Content/Data
* Used for
  + Converting meaningless data into meaningful information by adding XML structure
* **NOT** 
  + A programming language
  + Used for designing
  + Used for decoration
* To check the XML
  + Use xmllint utility
  + To install xmllint
    - sudo apt-get install xmllint
  + To use xmllint
    - xmllint page1.xml
    - --noout: do not display the xml contents

**HTML vs XML**

|  |  |
| --- | --- |
| **HTML** | **XML** |
| 1. Designing web pages | 1. Used for adding a structure |
| 1. Tags are pre-defined by W3C | 1. Tags are **user-defined** |
|  |  |

**Tag**

* Word enclosed by < and >
* Also called as element
* E.g.
  + <html>
* Types
  + Opening Tag
  + Closing Tag
  + Empty Tag
  + Root Tag
* Rules to create tag
  + XML element is case sensitive
    - <name> and <Name> are different elements
    - E.g.
      * <name>test</Name> : Error
      * <name>test</name>
      * <Name>test</Name>
  + Every opening element **MUST** be closed
    - E.g.
      * <name>test : Error
      * <name>test</name>
  + Element name **MUST NOT** start with number or element **MUST NOT** have only numbers
    - E.g.
      * <1Name>test</1Name> : Error
      * <1>test</1> : Error
      * <firstName>test</firstName>
  + Element MUST NOT contain any special characters like space (only underscore can be used)
    - E.g.
      * <first name>test</first name> : Error
      * <firstName>test</firstName>
      * <first\_name>test</first\_name>
  + Every XML file MUST contain ONE AND ONLY ONE root tag

**Attribute**

* Used to add extra information in a tag
* Rules
  + It will be always present in the form of name=value format
    - E.g.
      * <phone phoneType>+9134545</phone>: Error
      * <phone phoneType=”special”>+9134545</phone>
  + An element **must not** have multiple attributes with same name
    - E.g.
      * <phone type=”comp” type=”comp2”>+9134</phone>: Error
      * <phone type=”comp”>+9134</phone>
* Attribute only element
  + Element has all the information in the form of attributes
  + Attribute only element does **NOT** contain child element(s)
  + Written as an empty element
  + Limitations
    - Can not have multiple attributes with same name
    - Can not have an attribute for an attribute
  + E.g.

<person

name="person1"

address="pune"

phone="+9134345" />

**Well-formed XML:**

* which follows all the syntactical rules (given by W3C) of element and attribute

**Valid XML**

* A well-formed xml file which follows all logical rules (given by developer)
* Every Valid xml file is a well-formed but every well-formed is NOT a valid xml
* To define the logical rules
  + DTD
  + XML Schema

**DTD**

* Stands for Document Type Definition
* Used to define user’s logical rules
* Has its own syntax different than XML
* Types
  + Internal
    - Same xml will have the DTD declarations
  + External
    - DTD declarations will be written outside the xml file
    - External DTD file must have an extension of .dtd
    - External DTD file MUST NOT contain <!DOCTYPE> declaration
    - <!DOCTYPE> MUST be present in the same xml file
    - Use SYSTEM keyword to load external DTD file

<!DOCTYPE <root tag> SYSTEM “<dtd file path>”>

* Limitations
  + DTD has its own Syntax
  + DTD does NOT have any data type
  + DTD does NOT have any restriction
  + DTD CAN NOT understand XML Namespace
* Rules must start with <!DOCTYPE <type>>
  + DOCTYPE is a root element of the xml file
  + Syntax:

<!DOCTYPE name [

rules here

]>

* Rules to define elements
  + **Simple element**
    - Element which does not have any child element or an attribute
    - E.g.
      * <name>test</name>
    - Rule:

<!ELEMENT <element name> (<type>)>

* + - E.g.

<!ELEMENT name (#PCDATA)>

* + - * Where
        + #PCDATA: element contains characters
        + Parsed Character Data (string)
  + **Parent Element with a specific order:**
    - Syntax:

<!ELEMENT <element name> (<order of child elements>)>

* + - E.g.

<!DOCTYPE person [

<!ELEMENT person (name, address)>

<!ELEMENT name (#PCDATA)>

<!ELEMENT address (#PCDATA)>

]>

<person>

<name>test</name>

<address>Pune</address>

</person>

* + **Parent Element with child elements in any order:**
    - Syntax:

<!ELEMENT <element name> ANY>

* + - E.g.

<!DOCTYPE person [

<!ELEMENT person ANY>

<!ELEMENT name (#PCDATA)>

<!ELEMENT address (#PCDATA)>

]>

<person>

<name>test</name>

<address>Pune</address>

</person>

or

<person>

<address>Pune</address>

<name>test</name>

</person>

* + **Empty element:**
    - Syntax:

<!ELEMENT <element name> EMPTY>

* + - E.g.

**<!ELEMENT testEmptyElement EMPTY>**

<testEmptyElement></testEmptyElement>

<testEmptyElement />

* Rules to define attribute
  + Syntax:

<!ATTLIST <element name>

<attribute name> <data type> <#REQUIRED | #IMPLIED | “<default value>” >

* + Where:
    - #REQUIRED: the attribute has to be present for every element
    - #IMPLIED: the attribute is optionally present for an element
    - “<default value>”: the attribute is optionally present for an element
  + Where
    - CDATA: character data (string)
    - (<value1>|<value2>): enumerated values (one of the values)
  + E.g.

<phone phoneType=”special”>+91343</phone>

<!ATTLIST phone

phoneType CDATA #REQUIRED>

* Wild Characters
  + \*: zero or more
  + +: one or more
  + ?: exactly zero or exactly one

**XML Namespace**

* Group of similar type of elements
* Use xmlns to create a namespace
* Syntax:

<<prefix>:<element name> xmlns:<prefix>=”<namespace name>”>

</<prefix>:<element name>>

* E.g.
  + <t1:table **xmlns:t1**=”<http://htmltable>”>
    - <tr></tr>
  + </t1:table>
  + Where
    - t1: prefix of namespace
    - <http://htmltable>: namespace name
    - t1:table-> fully qualified element name
    - table: local name

**XML Schema**

* has to be written externally
* Steps
  + Create separate file with .xsd extension

<xs:schema xmlns:xs=”[**http://www.w3.org/2001/XMLSchema**](http://www.w3.org/2001/XMLSchema)”>

</xs:schema>

* + Load xsd file in the xml file

<age

xmlns:xsi=”[**http://www.w3.org/2001/XMLSchema-instance**](http://www.w3.org/2001/XMLSchema-instance)”

xsi:noNamespaceSchemaLocation=”<file name.xsd>”>50</age>

* Data Types
  + Built-in Data Types
    - string:
    - integer:
    - positiveInteger:
    - decimal:
    - date:
    - time:
    - datetime:
    - boolean:
  + Custom Data Types
    - Step 1
      * Create a simple type element declaration
    - Step 2
      * Add required restriction(s)
    - Restrictions
      * maxInclusive:
      * minInclusive:
      * maxExclusive:
      * minExclusive:
      * pattern: (<regular expression>)
      * maxLength:
      * minLength:
      * enumeration:
    - E.g.

<xs:element name="age">

<xs:simpleType>

<xs:**restriction** **base="xs:positiveInteger"**>

<xs:minExclusive value="25" />

<xs:maxExclusive value="60" />

</xs:**restriction**>

</xs:simpleType>

</xs:element>

// ERROR

<age>25</age>

<age>60</age>

<age>26</age>

<age>59</age>

* Validate Elements
  + Simple Type Element
    - Which **DOES NOT** have any child element or an attribute
    - Syntax:

<xs:element name=”<element name>” type=”<data type>” />

<xs:element name=”<element name>” type=”<data type>”></xs:element

* + - E.g.

<xs:element name=”age” type=”xs:positiveInteger” />

* + Complex Type Element
    - Which may have
      * At least one child element OR
      * At least one attribute
    - Where order is important (sequence)
      * Every child element MUST be present
      * They MUST appear in specified order

<xs:element name="person">

<xs:complexType>

<**xs:sequence**>

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

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

</**xs:sequence**>

</xs:complexType>

</xs:element>

<person>

<fullname>test</fullname>

<age>40</age>

</person>

// Error

<person>

<age>40</age>

<fullname>test</fullname>

</person>

* + - Where order is NOT important (all)
      * Every child element MUST be present
      * They may appear in any order

<xs:element name="person">

<xs:complexType>

<**xs:all**>

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

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

</**xs:all**>

</xs:complexType>

</xs:element>

<person>

<fullname>test</fullname>

<age>40</age>

</person>

<person>

<age>40</age>

<fullname>test</fullname>

</person>

* + - When only one of the child elements is required (choice)
      * Error when
        + All child elements are present
        + None of child elements is present

<xs:element name="gender">

<xs:complexType>

<xs:**choice**>

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

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

</xs:**choice**>

</xs:complexType>

</xs:element>

<gender>

<male></male>

</gender>

<gender>

<female></female>

</gender>

// ERROR

<gender>

<male></male>

<female></female>

</gender>

// ERROR

<gender>

</gender>

* Occurrence indicators
  + Types
    - maxOccurs
      * upper bound
      * use unbounded to have no limit on upper bound
    - minOccurs
      * lower bound
      * use unbounded to have no limit on lower bound
  + Mapping
    - \* (zero or more): minOccurs = “0” and maxOccurs=”unbounded”
    - + (one or more): minOccurs = “1” and maxOccurs=”unbounded”
    - ? (zero or one): minOccurs = “0” and maxOccurs=”1”
  + E.g.

<xs:element

minOccurs="1"

maxOccurs="unbounded"

name="phone"

type="xs:string" />

* Validate Attributes
  + Syntax:

<xs:attribute name=”<attribute name>” type=”<value type>” />

* + E.g.

<xs:element name="phone">

<xs:**complexType**>

<xs:**attribute** name="phoneType" type="xs:string" />

</xs:**complexType**>

</xs:element>

<phone phoneType=”special” />

* + Use “use” attribute to
    - Mandatory (required)

<xs:attribute

name="firstName"

type="xs:string"

use="required" />

<person firstName=”test” />

// ERROR

<person />

* + - Optional

<xs:attribute

name="firstName"

type="xs:string"

use="optional" />

<person firstName=”test” />

<person />

* + - Prohibited
      * If present then considered as an error

<xs:attribute

name="firstName"

type="xs:string"

use="prohibited" />

// ERROR

<person firstName=”test” />

<person />

* + Attribute having enumerated values

<xs:attribute name="phoneType" use="required">

<xs:simpleType>

<xs:restriction base="xs:string">

<xs:enumeration value="special" />

<xs:enumeration value="company" />

<xs:enumeration value="personal" />

</xs:restriction>

</xs:simpleType>

</xs:attribute>

<phone phoneType=”special” />

<phone phoneType=”company” />

<phone phoneType=”personal” />

// ERROR

<phone phoneType=”myphone” />

* + Use “default” attribute to specify the default value

<xs:attribute

name="address"

type="xs:string"

default="pune"

use="optional" />

// address = Karad

<person address=”Karad”/>

// address = pune

<person />

**XML Path**

* Used to select (get) one or more elements

<movies>

<movie>

<title>Toy Story</title>

<rating>4.5</rating>

</movie>

<movie>

<title>Cars</title>

<rating>4.0</rating>

</movie>

</movies>

* Types
  + - Absolute
      * Always starts with “/” (root)
    - Relative
  + E.g.
    - /movies: <movies>
    - /movies/movie:
      * return an array of movie element
      * array always starts from 1
    - /movies/movie[1]:

<movie>

<title>Toy Story</title>

<rating>4.5</rating>

</movie>

* + - /movies/movie[1]/title: Toy Story
    - /movies/movie[2]/rating: 4.0

**XML Stylesheet (xsl)**

* must be written outside the xml file (externally)
* must have an extension “.xsl”