

XML:eXtensible Mark Up language

Unit 2

Text book :N.P. Gopalan

chapter 8

Faculty: Ms. Fiona Coutinho

Intro XML:

XML and html based on SGML(standard generalized markup language)

XML-meta language as it creates other mark up languages

XML namespaces,schema,XLink,XPath,Xquery,RSS(Really simple syndication) and RDF(resource description framework or semantic web)

Intro XML:

html code:

```
<html>  
<body>  
<h1>harry potter</h1>  
<h2> 1999</h2>  
</body>  
</html>
```

XML code:

```
<book>  
<title>harry potter</title>  
<date>1999</date>  
</book>
```

XML Example1:

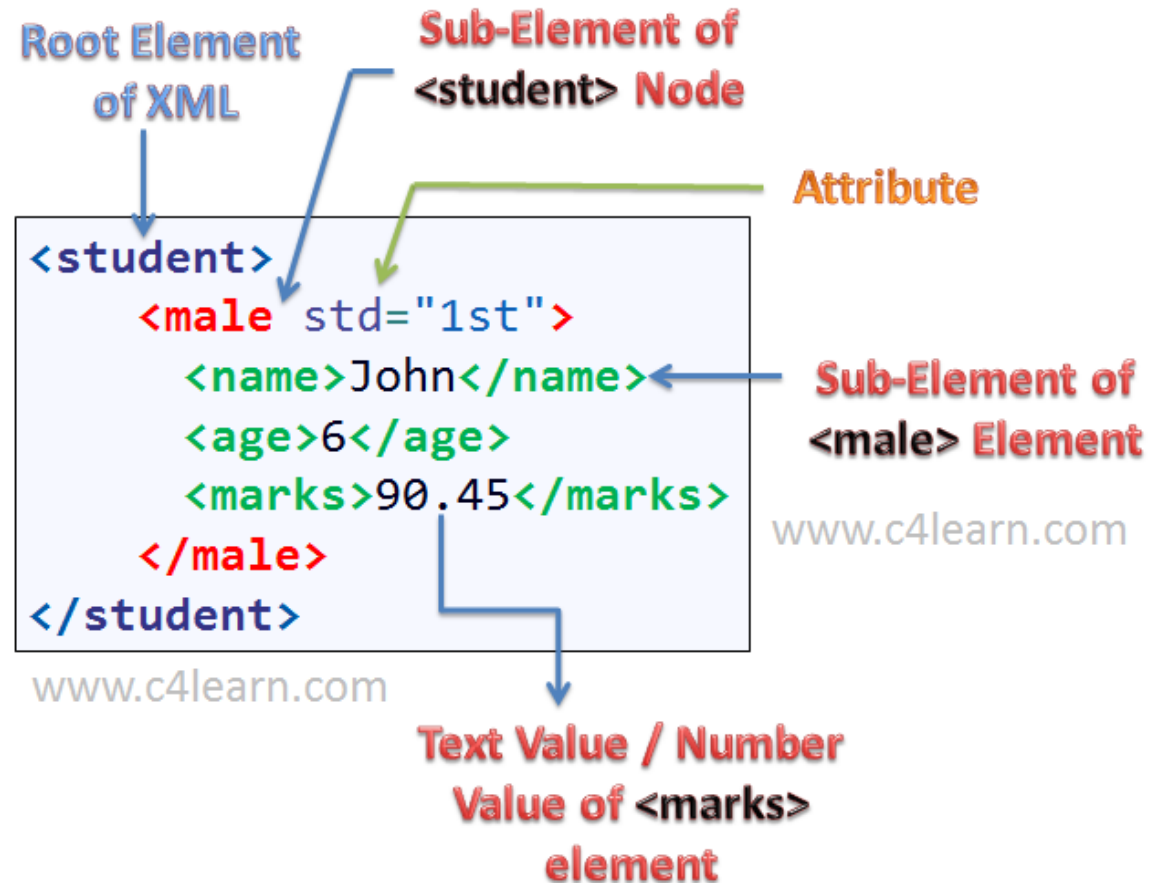
```
<?xml version="1.0" encoding="UTF-8"?>
  <student>
    <male std="1st">
      <name>Pritesh</name>
      <age>6</age>
      <marks>90.45</marks>
    </male>
    <female std="1st">
      <name>Pooja</name>
      <age>5</age>
      <marks>96.67</marks>
    </female>
  </student>
```

Note: Save this file as student.xml

XML Element Consists Of –

1. Other Element : XML Contain **<student>** tag which contain other Elements.
2. Text: Pritesh was text in between the <name> tag
3. Attributes: **<male>** element have attribute **std**
4. Mixing Above Elements

XML Example Explained:



Rules to write XML

1. All XML must have a root element.
2. All tags must be closed.
3. All tags must be properly nested.
4. Tag names have strict limits.
5. Tag names are case sensitive.
6. Tag names cannot contain spaces.
7. Attribute values must appear within quotes ("")

Exercise 1.:Write xml to store bookstore information?

XML Example2

```
<?xml version="1.0" encoding="UTF-8"?>
  <bookstore>
    <book category="cooking">
      <title lang="en">Everyday Italian</title>
      <author>Giada De Laurentiis</author>
      <year>2005</year>
      <price>30.00</price>
    </book>
    <book category="children">
      <title lang="en">Harry Potter</title>
      <author>J K. Rowling</author>
      <year>2005</year>
      <price>29.99</price>
    </book>
```

XML Example2 continued

```
<book category="web">
  <title lang="en">XQuery Kick Start</title>
  <author>James McGovern</author>
  <author>Per Bothner</author>
  <author>Kurt Cagle</author>
  <author>James Linn</author>
  <author>Vaidyanathan Nagarajan</author>
  <year>2003</year>
  <price>49.99</price>
</book>
<book category="web" cover="paperback">
  <title lang="en">Learning XML</title>
  <author>Erik T. Ray</author>
  <year>2003</year>
  <price>39.95</price>
</book>
```

Write xml to store collection of cd
information?

<CATALOG>

<CD>

<TITLE>Empire Burlesque</TITLE>

<ARTIST>Bob Dylan</ARTIST>

<COUNTRY>USA</COUNTRY>

<COMPANY>Columbia</COMPANY>

<PRICE>10.90</PRICE>

<YEAR>1985</YEAR>

</CD>

<CD>

<TITLE>Hide your heart</TITLE>

<ARTIST>Bonnie Tyler</ARTIST>

<COUNTRY>UK</COUNTRY>

<COMPANY>CBS Records</COMPANY>

<PRICE>9.90</PRICE>

<YEAR>1988</YEAR>

</CD>

</CATALOG>

Write xml to store breakfast menu?

<breakfast_menu>

<food>

<name>Belgian Waffles</name>

<price>\$5.95</price>

<description>

Two of our famous Belgian Waffles with plenty of real maple syrup

</description>

<calories>650</calories>

</food>

<food>

<name>Strawberry Belgian Waffles</name>

<price>\$7.95</price>

<description>

Light Belgian waffles covered with strawberries and whipped cream

</description>

<calories>900</calories>

</food>

</breakfast_menu>

XML APPLICATION

- **XMLNews is a specification for exchanging news and other information** :Using a standard makes it easier for both news producers and news consumers to produce, receive, and archive any kind of news information across different hardware, software, and programming languages.
- XML is used in many aspects of web development.

XML APPLICATION

- Transaction Data:Thousands of XML formats exists, in many different industries, to describe day-to-day data transactions(Stocks and Shares,Financial transactions)
- Medical data
- Mathematical data
- Scientific measurements
- Weather services

XML v/s HTML

- XML is often used to describe data and not data presentation.
- Unlike html ,XML does not carry any information about how to be displayed.
- Xml uses XSL whereas html uses CSS
- Xml is case sensitive
- Like with html ,Browsers cannot directly interpret XML data
- XML supports user defined tags unlike html
- Xml is strict with tags compared to html

XML ADVANTAGES

- It simplifies data sharing
- It simplifies data transport
- It simplifies platform changes
- It simplifies data availability
- Many computer systems contain data in incompatible formats. Exchanging data between incompatible systems (or upgraded systems) is a time-consuming task for web developers. Large amounts of data must be converted, and incompatible data is often lost.
- XML stores data in plain text format. This provides a software- and hardware-independent way of storing, transporting, and sharing data.
- XML also makes it easier to expand or upgrade to new operating systems, new applications, or new browsers, without losing data.
- With XML, data can be available to all kinds of "reading machines" like people, computers, voice machines, news feeds, etc.

Disadvantages Attributes:

1.XML code:

```
<book>  
<title>harry potter</title>  
<date>12/11/1999</date>  
</book>
```

2.XML code:

```
<book date="12/11/1999">  
<title>harry potter</title>  
</book>
```

3.XML code:

```
<book>  
<title>harry potter</title>  
<date>  
<day>12</day>  
<month>11</month>  
<year>1999</year>  
</book>
```

Disadvantages Attributes:

- 1. cant contain multiple values**
- 2. not expandable for future changes**
- 3. cant describe structure(child elements)**
- 4. difficult to manipulate by program**
- 5. difficult to test against DTD**

Well formed XML

- Complies to all rules to xml
- Merely well-formed XML is not *necessarily* valid, although it may be

Valid XML

- Valid XML has a DTD or schema associated with it and has been verified against all the rules contained in the DTD or schema
- Valid XML is well-formed as well.
- ```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE note SYSTEM "Note.dtd">
<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
</note>
```

# DTD : Document Type Definition

DTD Defines the structure and the legal elements and attributes of an XML document

- DTD Types:
  1. Internal DTD
  2. External DTD

# 1. Internal DTD

DTD is declared inside the XML file, it must be wrapped inside the <!DOCTYPE> definition

```
<?xml version="1.0"?>
```

```
<!DOCTYPE note [
```

```
<!ELEMENT note (to,from,heading,body)>
```

```
<!ELEMENT to (#PCDATA)>
```

```
<!ELEMENT from (#PCDATA)>
```

```
<!ELEMENT heading (#PCDATA)>
```

```
<!ELEMENT body (#PCDATA)>
```

```
]>
```

```
<note>
```

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

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

```
<heading>Reminder</heading>
```

```
<body>Don't forget me this weekend</body>
```

```
</note>
```



# DTD : Document Type Definition

- !DOCTYPE note defines that the root element of this document is note
- !ELEMENT note defines that the note element must contain four elements: "to,from,heading,body"
- !ELEMENT to defines the “to” element to be of type "#PCDATA"
- !ELEMENT from defines the “from” element to be of type "#PCDATA"
- !ELEMENT heading defines the “heading” element to be of type "#PCDATA"
- !ELEMENT body defines the “body” element to be of type "#PCDATA"

## 2. External DTD

<!DOCTYPE> definition must contain a reference to the DTD file.

```
<?xml version="1.0"?>
```

```
<!DOCTYPE note SYSTEM "note.dtd">
```

```
<note>
```

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

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

```
 <heading>Reminder</heading>
```

```
 <body>Don't forget me this weekend!</body>
```

```
</note>
```

# DTD : Document Type Definition

The Building Blocks of XML Documents:

1. Elements
2. Attributes
3. Entities
4. PCDATA
5. CDATA

# DTD : Document Type Definition

1.Elements:Examples of XML elements could be "note" and "message"

2.Attributes: provide extra information about elements

3.Entities :Some characters having a special meaning in XML, for eg. < defines the start of an XML tag. Entities are expanded when a document is parsed by an XML parser.

1. &lt;      <

2. &gt;      >

3. &amp;      &

4. &quot;    "

5. &apos;    '

# DTD : Document Type Definition

## 4. PCDATA: parsed character data.

- character data found between the start tag and the end tag of an XML element.
- PCDATA is text that WILL be parsed by a parser.
- The text will be examined by the parser for entities and markup.
- Tags inside the text will be treated as markup and entities will be expanded.
- parsed character data should not contain any &, <, or > characters; these need to be represented by the &amp; &lt; and &gt; entities, respectively.

# DTD : Document Type Definition

5.CDATA:character data.

- NOT be parsed by a parser.
- Tags inside the text will NOT be treated as markup
- entities will not be expanded.

# DTD : Document Type Definition

## 1.Declaring Elements

<!ELEMENT element-name category>

<!ELEMENT element-name EMPTY>

Example:

<!ELEMENT br EMPTY>

XML example:

<br />

# DTD : Document Type Definition

## 2.Elements with Parsed Character Data

<!ELEMENT from (#PCDATA)>

3.<!ELEMENT element-name ANY>

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

<!DOCTYPE address [ <!ELEMENT address ANY> ]>

<address> Here's a bit of sample text </address>



# DTD : Document Type Definition

## 4.Elements with Children (sequences)

<!ELEMENT element-name (child1)>

or

<!ELEMENT element-name (child1,child2,...)>

- When children are declared in a sequence separated by commas, the children must appear in the same sequence in the document.
- In a full declaration, the children must also be declared, and the children can also have children

<!ELEMENT note (to,from,heading,body)>

<!ELEMENT to (#PCDATA)>

<!ELEMENT from (#PCDATA)>

<!ELEMENT heading (#PCDATA)>

<!ELEMENT body (#PCDATA)>

# DTD : Document Type Definition

## 5.Declaring Only One Occurrence of an Element

`<!ELEMENT element-name (child-name)>`

Example:

`<!ELEMENT note (message)>`child element "message" must occur once, and only once inside the "note" element.

# DTD : Document Type Definition

6.Declaring Minimum One Occurrence of an Element

```
<!ELEMENT note (message+)>
```

7.Declaring Zero or More Occurrences of an Element

```
<!ELEMENT note (message*)>
```

8.Declaring Zero or One Occurrences of an Element

```
<!ELEMENT note (message?)>
```

9.Declaring either/or Content

```
<!ELEMENT note (to,from,header,(message|body))>
```

10.Declaring Mixed Content

```
<!ELEMENT note (#PCDATA|to|from|header|message)*>
```

The example above declares that the "note" element can contain zero or more occurrences of parsed character data, "to", "from", "header", or "message" elements.

# DTD : Document Type Definition

## 10.Example Mixed Content

```
<!DOCTYPE address [<!ELEMENT address (#PCDATA|name)*>
<!ELEMENT name (#PCDATA)>]>
```

```
<address> Here's a bit of text mixed up with the child element.
<name> Tanmay Patil </name>
</address>
```

# DTD : Document Type Definition

DTD – Attributes declared with an ATTLIST declaration

```
<!ATTLIST element-name attribute-name attribute-type attribute-value>
```

DTD example:

```
<!ATTLIST payment paytype CDATA "cheque">
```

XML example:

```
<payment paytype="check" />
```

# DTD : Document Type Definition

The attribute-value can be one of the following:

1. value                      The default value of the attribute
2. #REQUIRED              The attribute is required
3. #IMPLIED                The attribute is optional
4. #FIXED value            The attribute value is fixed

# DTD : Document Type Definition

## 1.A Default Attribute Value

DTD:

```
<!ELEMENT square EMPTY>
```

```
<!ATTLIST square width CDATA "0">
```

Valid XML:

```
<square width="100" />
```

In the example above, the "square" element is defined to be an empty element with a "width" attribute of type CDATA.

If no width is specified, it has a default value of 0.

# DTD : Document Type Definition

## 2.#REQUIRED

<!ATTLIST element-name attribute-name attribute-type #REQUIRED>

DTD:

<!ATTLIST person number CDATA #REQUIRED>

Valid XML:

<person number="5677" />

Invalid XML:

<person />

Use the #REQUIRED keyword if you don't have an option for a default value, but still want to force the attribute to be present.



# DTD : Document Type Definition

## 3.#IMPLIED

<!ATTLIST element-name attribute-name attribute-type #IMPLIED>

DTD:

<!ATTLIST contact fax CDATA #IMPLIED>

Valid XML:

<contact fax="555-667788" />

Valid XML:

<contact />

Use the #IMPLIED keyword if you don't want to force the author to include an attribute, and you don't have an option for a default value.

# DTD : Document Type Definition

## 4.#FIXED

DTD:

```
<!ATTLIST sender company CDATA #FIXED "Microsoft">
```

Valid XML:

```
<sender company="Microsoft" />
```

Invalid XML:

```
<sender company="WXYZ" />
```

- Use the #FIXED keyword when you want an attribute to have a fixed value without allowing the author to change it.
- If an author includes another value, the XML parser will return an error.

# DTD : Document Type Definition

## 5.Enumerated Attribute Values

DTD:

```
<!ATTLIST payment type (cheque|cash) "cash">
```

XML example:

```
<payment type=" cheque" />
```

or

```
<payment type="cash" />
```

Use enumerated attribute values when you want the attribute value to be one of a fixed set of legal values.

# DTD : Document Type Definition

## Types entities: Internal ,external and parameter

Entities are variables used to define shortcuts to common text.

**1.Internal** : links a name with string of text

XML spec define 5 internal entities

DTD example:**<!ENTITY writer "joana">**

XML:**<author>&writer;</author>**

**2.External** :associate a name with content of another file so that xml can use content of another file

DTD example:

**<!ENTITY writer SYSTEM "http://www.xml.com/entities/entities.xml"**

XML: **<author>&writer;</author>**

**3.Parameter:**can occur only in DOCTYPE declaration

DTD example:

**<!ENTITY % personcontent "#PCDATA">**

**<!ELEMENT allen(%personcontent;)\*>**

XML: **<allen>details of person....</allen>**

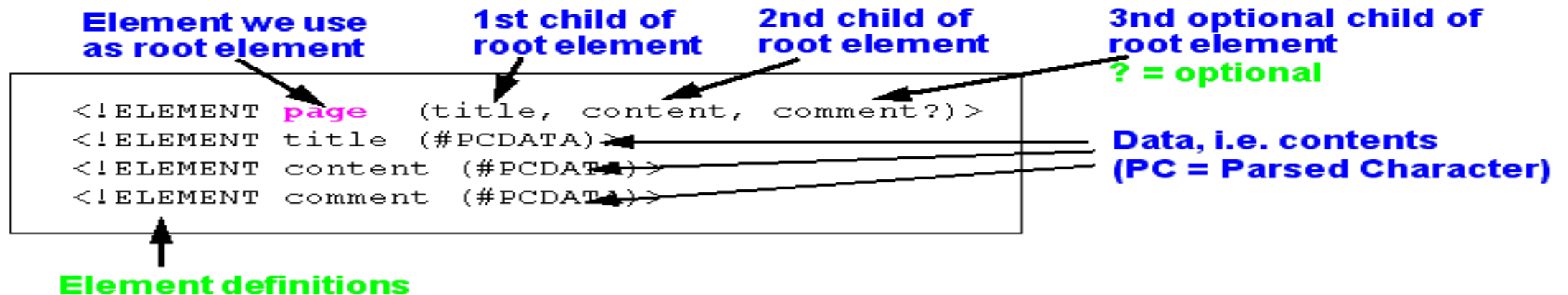
**EXAMPLES**

# DTD : Document Type Definition

A simple XML document of type <page>

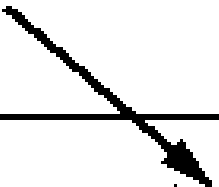
```
<?xml version="1.0"?>
<page>
 <title>Hello friend</title>
 <content>Here is some content :)</content>
 <comment>Written by DKS/Tecfa, adapted from S.M./the Cocoon samples</comment>
</page>
```

A DTD that would validate this "page" document



```
<?xml version="1.0"?>
<!DOCTYPE list SYSTEM "simple_recipe.dtd">
<list>
 <recipe>
 <author>Carol Schmidt</author>
 <recipe_name>Chocolate Chip Bars</recipe_name>
 <meal>Dinner</meal>
 <ingredients>
 <item>2/3 C butter</item> <item>2 C brown sugar</item>
 <item>1 tsp vanilla</item> <item>1 3/4 C unsifted all-purpose flour</item>
 <item>1 1/2 tsp baking powder</item>
 <item>1/2 tsp salt</item> <item>3 eggs</item>
 <item>1/2 C chopped nuts</item>
 <item>2 cups (12-oz pkg.) semi-sweet choc. chips</item>
 </ingredients>
 <directions>
 Preheat oven to 350 degrees.
 Melt butter; combine with brown sugar and vanilla in large mixing bowl.
 Set aside to cool. Combine flour, baking powder, and salt; set aside.
 Add eggs to cooled sugar mixture; beat well.
 Stir in reserved dry ingredients, nuts, and chips.
 Spread in greased 13-by-9-inch pan.
 Bake for 25 to 30 minutes until golden brown; cool. Cut into squares.
 </directions>
 </recipe>
</list>
```

**The list element (root)  
must contain one or more  
recipe elements**  
**+ = at least one**



**Mandatory children (subelements)  
of the recipe element**



**Comma separated elements must appear  
in the same order!**

```
<!ELEMENT list (recipe+)>
<!ELEMENT recipe (author, recipe_name, meal, ingredients, directions)>
<!ELEMENT author (#PCDATA)>
<!ELEMENT recipe_name (#PCDATA)>
<!ELEMENT meal (#PCDATA)>
<!ELEMENT ingredients (item+)>
<!ELEMENT item (#PCDATA)>
<!ELEMENT directions (#PCDATA)>
```



```
<?xml version="1.0" " ?>
<!DOCTYPE STORY SYSTEM "story-grammar.dtd">
<?xml-stylesheet href="story-grammar.css" type="text/css"?>
<STORY>
 <title>The little XMLer</title>
 <context></context>
<problem></problem>
<goal></goal>
<THREADS>
 <EPISODE>
 <subgoal>I have to do it ...</subgoal>
 <ATTEMPT>
 <action></action>
 </ATTEMPT>
 <result></result>
 </EPISODE>
</THREADS>
<moral></moral>
<INFOS>
</INFOS>
</STORY>
```

**The THREADS element  
must contain one or more  
EPISODE elements**

**+ means "at least one"**

**All elements of STORY  
must be present  
in this order**

**Comment**

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!-- DTD to write simple stories - VERSION 1.0 1/2007
 Made by Daniel K. Schneider / TECFA / University of Geneva -->
<!ELEMENT STORY (title, context, problem, goal, THREADS, moral, INFOS)>
<!ATTLIST STORY xmlns:xlink CDATA #FIXED "http://www.w3.org/1999/xlink">
<!ELEMENT THREADS (EPISODE+)>
<!ELEMENT EPISODE (subgoal, ATTEMPT+, result) >
<!ELEMENT ATTEMPT (action | EPISODE) >
<!ELEMENT INFOS ((date | author | a) *) >
<!ELEMENT title (#PCDATA) >
<!ELEMENT context (#PCDATA) >
<!ELEMENT problem (#PCDATA) >
<!ELEMENT goal (#PCDATA) >
<!ELEMENT subgoal (#PCDATA) >
<!ELEMENT result (#PCDATA) >
<!ELEMENT moral (#PCDATA) >
<!ELEMENT action (#PCDATA) >
<!ELEMENT date (#PCDATA) >
<!ELEMENT author (#PCDATA) >
<!ELEMENT a (#PCDATA) >
<!ATTLIST a
 xlink:href CDATA #REQUIRED
 xlink:type CDATA #FIXED "simple" >
```

**Choose one  
| means "or"**

**Choose as  
many as  
you like  
in random  
order**

**These elements  
only contain  
text**

**Empty element  
(has no contents)**

**name attribute  
is mandatory**

**gender attribute  
is optional**

```
<!ELEMENT family (person)+>
<!ELEMENT person EMPTY>
<!ATTLIST person name CDATA #REQUIRED>
<!ATTLIST person gender (male|female) #IMPLIED>
<!ATTLIST person type (mother|father|boy|girl) "mother">
<!ATTLIST person id ID #REQUIRED>
```

**id is required  
and of type ID  
(no elements can have same ID !)**

**type attribute must be  
either mother, father, ...**

```
<?xml version="1.0" ?>
<!DOCTYPE family SYSTEM "family.dtd">
<family>
 <person name="Joe Miller" gender="male"
 type="father" id="123.456.789"/>
 <person name="Josette Miller" gender="female"
 type="girl" id="123.456.987"/>
</family>
```

# For the given xml write the DTD?

```
<empinfo>
```

```
 <employee id="1">
```

```
 <name>Opal Kole</name>
```

```
 <designation>Senior Engineer</designation>
```

```
 <email>email@myemail.com</email>
```

```
 </employee>
```

```
 <employee id="2">
```

```
 <name from="CA">Opal Kole</name>
```

```
 <designation discipline="DBA">Senior Engineer</designation>
```

```
 <email>email@email.com</email>
```

```
 </employee>
```

```
</empinfo>
```

# DTD Example

```
<?xml version="1.0" standalone="yes"?>
<!DOCTYPE empinfo [
 <!ELEMENT empinfo (employee)>
 <!ELEMENT employee (name, designation, email)>
 <!ATTLIST employee id CDATA #REQUIRED>
 <!ELEMENT name (#PCDATA)>
 <!ATTLIST name from CDATA #IMPLIED>
 <!ELEMENT designation (#PCDATA)>
 <!ATTLIST designation discipline CDATA #FIXED "Web developer">
 <!ELEMENT email (#PCDATA)>
 <!ATTLIST email domain CDATA "personal">
]>
```

# Validate

- <https://www.xmlvalidation.com>

# XML Schema



# XML Schema

## why schema?

- DTD's not xml
- Don't support namespaces
- no limit on character data
- don't support inheritance

## Building blocks schema:

### **1.Simple Element:**

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

- string,decimal,integer,boolean,date,time,complex,attributes

### **2.Attributes:**

```
<xs:attribute name="lang" type="xs:string" default="English"/>
```

**OR**

```
<xs:attribute name="lang" type="xs:string" fixed="English"/>
```

### **3.Complex elements:**

4 types:Empty, contain other element's ,text, elements and text together

`<xs:sequence>` - define child elements

# XML Schema

- An XML Schema describes the structure of an XML document, just like a DTD
- 2 types elements complex type and Simple type
- note is a complex type of element

```
<?xml version="1.0" encoding="UTF-8" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
 <xs:element name="note">
 <xs:complexType>
 <xs:sequence>
 <xs:element name="to" type="xs:string"/>
 <xs:element name="from" type="xs:string"/>
 <xs:element name="heading" type="xs:string"/>
 <xs:element name="body" type="xs:string"/>
 </xs:sequence>
 </xs:complexType>
 </xs:element>
</xs:schema>
```

# XML Schema

- `<xs:element name="note">` defines the element called "note"
- `<xs:complexType>` the "note" element is a complex type
- `<xs:sequence>` the complex type is a sequence of elements
- `<xs:element name="to" type="xs:string">` the element "to" is of type string (text)
- `<xs:element name="from" type="xs:string">` the element "from" is of type string

# XML Schemas :Advantages over DTD

- XML Schemas are written in XML
- XML Schemas are extensible to additions
- XML Schemas support data types
- XML Schemas support namespaces

# XML Schema :syntax of element

- <element  
  id=ID  
  name=NCName  
  type=QName  
  default=string  
  fixed=string  
  maxOccurs=nonNegativeInteger|unbounded  
  minOccurs=nonNegativeInteger  
  nillable=true|false//if true cannot contain nulls  
  abstract=true|false  
  >  
</element>

# XML Schema

```
<xs:element name="age" type="xs:nonNegativeInteger"/>
```

```
<xs:element name="dateborn" type="xs:date"/>
```

```
<xs:element name="OrderID" type="xs:int" />
```

```
<xs:element name="Customer_name" type="xs:string"
 default="unknown" />
```

```
<xs:element name="Customer_location" type="xs:string" fixed=" UK"
 />
```

Save schema as note.xsd

```
<xs:element name="Customer">
 <xs:complexType>
 <xs:sequence>
 <xs:element name="Dob" type="xs:date" />
 <xs:element name="Address" type="xs:string" />
 </xs:sequence>
 </xs:complexType>
</xs:element>
<xs:element name="Supplier">
 <xs:complexType>
 <xs:sequence>
 <xs:element name="Phone" type="xs:integer" />
 <xs:element name="Address" type="xs:string" />
 </xs:sequence>
 </xs:complexType>
</xs:element>
```

# Sample XML

<Customer>

<Dob>2000-01-12T12:13:14Z</Dob>

<Address> 34 thingy street, someplace, sometown, ww1 8uu </Address>

</Customer>

<Supplier>

<Phone>0123987654</Phone>

<Address>22 whatever place, someplace, sometown, ss1 6gy </Address>

</Supplier>



# XML schema attributes

```
<xs:attribute name="ID" type="xs:string" use="optional" />
```

Example:

```
<xs:element name="Order">
```

```
<xs:complexType>
```

```
<xs:attribute name="OrderID" type="xs:int" use="optional" />
```

```
</xs:complexType>
```

```
</xs:element>
```

OR

```
<xs:element name="Order"> <xs:complexType> <xs:attribute
 name="OrderID" type="xs:int" use="required" /> </xs:complexType>
</xs:element>
```

- The default and fixed attributes can be specified within the XSD attribute specification (in the same way as they are for elements).

EXAMPLE1

# XSD

```
<xs:element name="Shape">
 <xs:complexType>
 <xs:sequence>
 <xs:element name="Point" minOccurs="1" maxOccurs="unbounded">
 <xs:complexType>
 <xs:attribute name="x" type="xs:int" />
 <xs:attribute name="y" type="xs:int" />
 </xs:complexType>
 </xs:element>
 </xs:sequence>
 <xs:attribute name="Colour" type="xs:string" />
 </xs:complexType>
</xs:element>
```

# Sample XML

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

```
<Shape Colour="Black">
```

```
<Point x="0" y="0" />
```

```
<Point x="100" y="0" />
```

```
<Point x="50" y="50" />
```

```
</Shape>
```

EXAMPLE2

- Write XML For the given schema

# Example schema

```
<?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>
```

# Solution XML

<contact>

<name>abc</name>

<company>xyz</company>

<phone>9727377333</phone>

</contact>



EXAMPLE3

# Question

- Write schema for the given xml document?

# XML document using a schema

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

<shiporder orderid="889923"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="shiporder.xsd">
 <orderperson>John Smith</orderperson>
 <shipto>
 <name>Ola Nordmann</name>
 <address>Langgt 23</address>
 <city>4000 Stavanger</city>
 <country>Norway</country>
 </shipto>
 <item>
 <title>Empire Burlesque</title>
 <note>Special Edition</note>
 <quantity>1</quantity>
 <price>10.90</price>
 </item>
 <item>
 <title>Hide your heart</title>
 <quantity>1</quantity>
 <price>9.90</price>
 </item>
</shiporder>
```

SOLUTION

```
<?xml version = "1.0" encoding = "UTF-8"?>
 <xs:schema xmlns:xs = "http://www.w3.org/2001/XMLSchema">
 <xs:element name = " shiporder ">
 <xs:complexType>
 <xs:sequence>
 <xs:element name = "orderperson" type = "xs:string" />
 <xs:element name = "shipto">
 <xs:complexType>
 <xs:sequence>
 <xs:element name = "name" 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>
```

```
<xs:element name = "item" minOccurs="1" maxOccurs="unbounded" >
<xs:complexType>
<xs:sequence>
<xs:element name = "title" type = "xs:string" />
<xs:element name = "note" type = "xs:string" minOccurs="0" />
<xs:element name = "quantity" type = "xs:int" />
<xs:element name = "price" type = "xs:decimal" />
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:sequence>
<xs:attribute name="OrderId" type="xs:int" use="required" />
</xs:complexType>
</xs:element>
</xs:schema>
```

# Sample XML file

```
<catalog>
<cd>
<title>Empire Burlesque</title>
<artist>Bob Dylan</artist>
<country>USA</country>
<company>Columbia</company>
<price>10.90</price>
<year>1985</year>
</cd>
<cd>
<title>Hide your heart</title>
<artist>Bonnie Tyler</artist>
<country>UK</country>
<company>CBS Records</company>
<price>9.90</price>
<year>1988</year>
</cd>
:
:
</catalog>
```

# XSL : Extensible stylesheet language

- XSL describes how to display xml doc
- To declare the document an XSL style sheet use <xsl:stylesheet> or <xsl:transform> element
- `<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">`
- To get access to the XSLT elements, attributes and features we must declare the XSLT namespace at the top of the document.
- The `xmlns:xsl="http://www.w3.org/1999/XSL/Transform"` points to the official W3C XSLT namespace.
- If you use this namespace, you must also include the attribute `version="1.0"`
- XSL has 3 parts:
  1. XSLT-language for transforming xml documents
  2. xpath-language for navigating xml documents
  3. xsl-fo-language for formatting xml documents

XSL can add new elements,remove elements,sort,rearrange,test,make decisions about elements to be displayed ,etc.



# XSL : Extensible stylesheet language

- **xml namespaces**: collection of xml elements and attributes identified by IRI(internationalized resource identifier).
- Avoids naming conflicts by associating URL with every element
- xmlns is keyword used

```
<Book xmlns:lib="http://www.library.com">
```

```
<lib:title>holmes</lib:title>
```

```
<lib:Author>Arthur</lib:Author>
```

```
</Book>
```

## **default namespace:**

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"?
```

```
<xsl:template match="/">
```

```
....
```

```
</xsl:template>
```

# XSLT Process steps

# Step1:Start with a Raw XML Document or write your xml document

```
<catalog>
<cd>
<title>Empire Burlesque</title>
<artist>Bob Dylan</artist>
<country>USA</country>
<company>Columbia</company>
<price>10.90</price>
<year>1985</year>
</cd>
:
</catalog>
```

Note: Save file as catalog.xml

## Step2:Write XSLT

```
<?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>My CD Collection</h2>
 <table border="1">
 <tr bgcolor="#9acd32">
 <th>Title</th>
 <th>Artist</th>
 </tr>
```

## Step2:Write XSLT

```
<xsl:for-each select="catalog/cd">
 <tr>
 <td><xsl:value-of select="title"/></td>
 <td><xsl:value-of select="artist"/></td>
 </tr>
</xsl:for-each>
</table>
</body>
</html>
</xsl:template>
</xsl:stylesheet>
```

Note: Save this file as cdcatalog.xsl

## Step3:Link XSL Style Sheet to XML

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

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

```
<catalog>
```

```
.....
```

Output:

## My CD Collection

Title	Artist
Empire Burlesque	Bob Dylan
Hide your heart	Bonnie Tyler
Greatest Hits	Dolly Parton
Still got the blues	Gary Moore
Eros	Eros Ramazzotti
One night only	Bee Gees
Sylvias Mother	Dr.Hook
Maggie May	Rod Stewart
Romanza	Andrea Bocelli
When a man loves a woman	Percy Sledge
Black angel	Savage Rose
1999 Grammy Nominees	Many
For the good times	Kenny Rogers
Big Willie style	Will Smith
Tupelo Honey	Van Morrison
Soulsville	Jorn Hoel
The very best of	Cat Stevens
Stop	Sam Brown
Bridge of Spies	T`Pau
Private Dancer	Tina Turner
Midt om natten	Kim Larsen
Pavarotti Gala Concert	Luciano Pavarotti

# XSLT Elements

- An XSL style sheet consists of one or more set of rules that are called templates.
- A template contains rules to apply when a specified node is matched
- 1.<xsl:template>** element defines a template.
- The **match** attribute is used to associate a template with an XML element
- The **match="/"** attribute associates the template with the root of the XML source document.
- The content inside the <xsl:template> element defines some HTML to write to the output



# XSLT Element

## 2. <xsl:value-of>:

- extract the value of a selected node and add it to the output stream of the transformation
- <td><xsl:value-of select="catalog/cd/title"/></td>

# XSLT Element

3. `<xsl:for-each>` : allows you to do looping

- used to select every XML element of a specified node-set
- `<xsl:for-each select="catalog/cd">`

```
 <tr>
```

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

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

```
 </tr>
```

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

# XSLT Element

- **<xsl:for-each select="catalog/cd[artist='Bob Dylan']">**
- Legal filter operators are:
  1. = (equal)
  2. != (not equal)
  3. &lt; less than
  4. &gt; greater than

# XSLT Element

4. <xsl:sort> used to sort output

Its used inside the <xsl:for-each> element in the XSL file

```
<xsl:for-each select="catalog/cd">
 <xsl:sort select="artist"/>
 <tr>
 <td><xsl:value-of select="title"/></td>
 <td><xsl:value-of select="artist"/></td>
 </tr>
</xsl:for-each>
```

- The **select** attribute indicates what XML element to sort on.

# XSLT Element

5. <xsl:if> : used to put a conditional test against the content of the XML file

```
<xsl:for-each select="catalog/cd">
 <xsl:if test="price > 10">
 <tr>
 <td><xsl:value-of select="title"/></td>
 <td><xsl:value-of select="artist"/></td>
 <td><xsl:value-of select="price"/></td>
 </tr>
 </xsl:if>
</xsl:for-each>
```

# XSLT Element

## 6. <xsl:choose> :

### SYNTAX:

```
<xsl:choose>
 <xsl:when test="expression">
 ... some output ...
 </xsl:when>
 <xsl:otherwise>
 ... some output
 </xsl:otherwise>
</xsl:choose>
```

# XSLT Element

## 6. <xsl:choose> :EXAMPLE

```
<xsl:for-each select="catalog/cd">
 <tr>
 <td><xsl:value-of select="title"/></td>
 <xsl:choose>
 <xsl:when test="price > 10">
 <td bgcolor="#ff00ff">
 <xsl:value-of select="artist"/></td>
 </xsl:when>
 <xsl:otherwise>
 <td><xsl:value-of select="artist"/></td>
 </xsl:otherwise>
 </xsl:choose>
 </tr>
 </xsl:for-each>
```

# XSLT Element

6.<xsl:choose>:EXAMPLE with more then one <xsl:when> is used

```
<xsl:choose>
 <xsl:when test="price > 10">
 <td bgcolor="#ff00ff">
 <xsl:value-of select="artist"/></td>
 </xsl:when>
 <xsl:when test="price > 9">
 <td bgcolor="#cccccc">
 <xsl:value-of select="artist"/></td>
 </xsl:when>
 <xsl:otherwise>
 <td><xsl:value-of select="artist"/></td>
 </xsl:otherwise>
 </xsl:choose>
```



# Difference between XSL and XSLT

- XSL :extensible stylesheet language
- XSLT : XSL transformation which is a part of XSL

# Semester questions

## 1. Difference between XSL and CSS

- XSL used to format and display XML document on browser whereas CSS is used to format html document
- XSL can add new elements into the generated file or remove existing elements or sort and rearrange(filter) elements as required unlike CSS XSL uses xml notations unlike CSS XSL not supported by modern browsers but CSS is.
- XSL is complex to write compared to CSS
- **Note: Always write code example as one of the point of difference irrespective of the technology being discussed**

# Semester questions

## 2.Difference between XML and html

- XML is data description language whereas html is data presentation language
- XML uses XSL to be formatted and displayed on the browser whereas html uses CSS
- XML is case sensitive unlike html
- In XML we can define custom tags unlike html

# Semester questions

## 3. Difference between Xhtml and html

- XHTML is HTML redesigned as XML
- In XHTML DOCTYPE is **mandatory**
- The xmlns attribute in <html> is **mandatory**
- <html>, <head>, <title>, and <body> are **mandatory**
- Follows rules as XML:
- XHTML elements must be **properly nested**
- XHTML elements must always be **closed**
- XHTML elements must be in **lowercase**
- XHTML documents must have **one root element**

# Semester questions: sample XHTML document

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

```
<html xmlns="http://www.w3.org/1999/xhtml">
```

```
<head>
 <title>Title of document</title>
</head>
```

```
<body>
 some content
</body>
```

```
</html>
```