XML TECHNOLOGIES

3.1 Introduction to XML
3..2 XML structure – Elements - Well-formed XML -XML Namespaces
3.3 Working with DTD - Adding DTDs to documents
3.4 Defining DTD entities - Defining Parameter entities
3.5 Working with attributes - Defining attributes - Defining multiple attributes - Using predefined attributes
3.6 CSS basics - Adding CSS to document
3.7 Need of XSL - XSL basics - XSL transformations
3.8 Introduction to Schemas - Defining simple elements and types
3.9 XML query.

## 3.1 Introduction to XML

### What is xml?

- ✓XML stands for e**X**tensible **M**arkup **L**anguage.
- ✓XML is designed to transport and store data.
- ✓XML is important to know, and very easy to learn.
- ✓XML is a markup language much like HTML.
- ✓XML was designed to carry data, not to display data.
- ✓XML tags are not predefined. You must define your own tags.
- ✓XML is designed to be self-descriptive.
- ✓XML is a W3C Recommendation.

### The Difference Between XML and HTML

| XML | HTML |
|---|---|
| XML was designed to transport and store data | HTML was designed to display data |
| With focus on what data is | With focus on how data looks |

### How Can XML be Used?

- ✓XML Separates Data from HTML
- ✓XML Simplifies Data Sharing
- ✓XML Simplifies Data Transport
- ✓XML Simplifies Platform Changes
- ✓XML Makes Your Data More Available
- ✓XML is Used to Create New Internet Languages

**XML Tree**

- ✓XML documents form a tree structure that starts at "the root" and branches to "the leaves".
- ✓XML Documents Form a Tree Structure
- ✓XML documents must contain a root element. This element is "the parent" of all other elements

The elements in an XML document form a document tree. The tree starts at the root and branches to the lowest level of the tree.

**Syntax:**

```
<root>
<child>
<subchild>.....</subchild>
</child>
</root>
```

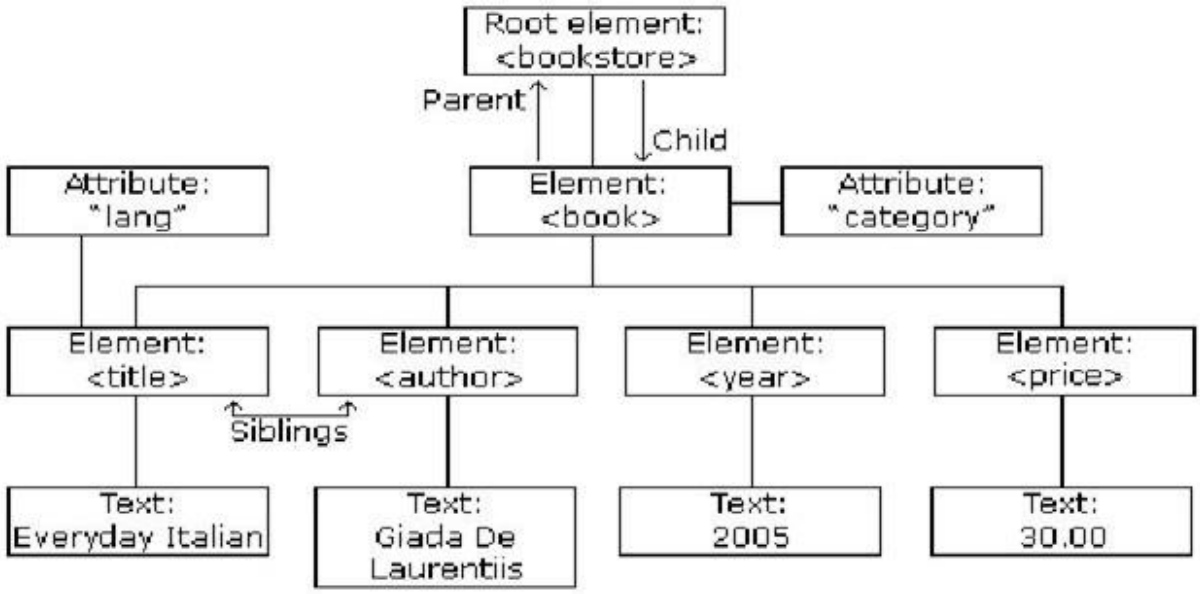
**Example:**

```
<?xml version="1.0" encoding="UTF-8"?>
<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
</note>
```

The terms parent, child, and sibling are used to describe the relationships between elements. Parent elements have children. Children on the same level are called siblings

**Example: book.xml**

```
<?xml version="1.0"?> <bookstore>
<book category="cooking">
<title lang="english">Everyday Italian</title>
<author>Giada De Laurentiis</author> <year>2005</year>
<price>30.00</price>
</book>
<book category="children">
<title lang="english">Harry Potter</title> <author>J K. Rowling</author>
<year>2005</year> <price>29.99</price>
</book>
<book category="web">
<title lang="english">Learning XML</title>
<author>Erik T. Ray</author>
<year>2003</year>
<price>39.95</price>
</book>
</bookstore>
```




**XML Rules**

- ✓All XML Elements Must Have a Closing Tag
- ✓XML Tags are Case Sensitive
- ✓XML Elements Must be Properly Nested
- ✓XML Documents Must Have a Root Element
- ✓XML Attribute Values Must be Quoted
- ✓Character and Entity References
- ✓Comments in XML
- ✓White-space is Preserved in XML

### 3.2 XML Elements

- ✓An **XML element** is everything from (including) the element's **start tag** to (including) the element's **end tag**.
- ✓An element can contain other elements, simple text or a mixture of both. Elements can also have attributes

```
<bookstore>
    <book category="Prog">
        <title>JAVA</title>
        <author>James Gosling</author>
        <year>2005</year>
        <price>RS. 500.00</price>
    </book>
</bookstore>
```

- In the example above, <bookstore> and <book> have **element contents**, because they contain other elements. <author> has **text content** because it contains text.
- In the example above only <book> has an attribute (category="Prog").

**XML Naming Rules**

- **XML elements must follow these naming rules:**
  - Names can contain letters, numbers, and other characters
  - Names cannot start with a number or punctuation character
  - Names cannot start with the letters xml (or XML, or Xml, etc)
  - Names cannot contain spaces
  - Any name can be used, no words are reserved

- **Best Naming Practices**

  - Make names descriptive. Names with an underscore separator are nice
  - <first_name>, <last_name>.
  - Names should be short and simple, <book_title>
  - Avoid "-" characters, Avoid "." characters, Avoid ":" characters

**XML Attributes**

- XML elements can have attributes in the start tag.
- Attributes provide additional information about elements.
- XML Attributes Must be Quoted.

✓ Attribute values must always be enclosed in quotes, but either single or double quotes can be used.

✓ For a person's sex, the person tag can be written like this

<person sex="female"> or <person sex='female'>

Example

```
<person sex="female">
    <firstname>Anna</firstname>
    <lastname>Smith</lastname>
</person>
```

**XML Attributes vs Elements**

✓ Take a look at these examples:

```
<person sex="female">
    <firstname>Anna</firstname>
    <lastname>Smith</lastname>
</person>
```

Example for XML element

```
<person>
    <sex>female</sex>
    <firstname>Anna</firstname>
    <lastname>Smith</lastname>
</person>
```

**Avoid XML Attributes**

- **Some of the problems with using attributes are:**
  - Attributes cannot contain multiple values (elements can)
  - Attributes cannot contain tree structures (elements can)
  - Attributes are not easily expandable (for future changes)
  - Attributes are difficult to read and maintain. Use elements for data. Use attributes for information that is not relevant to the data.

**Well Formed XML Documents**

- XML with correct syntax is "Well Formed" XML.
- XML validated against a DTD is "Valid" XML.
- A "Well Formed" XML document has correct XML syntax

XML documents must have a root element

XML elements must have a closing tag

XML tags are case sensitive

XML elements must be properly nested

XML attribute values must be quoted

**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>
```

- If these XML fragments were added together, there would be a name conflict. Both contain a <table> element, but the elements have different content and meaning.
- An XML parser will not know how to handle these differences

**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>
```

- In the example above, there will be no conflict because the two <table> elements have different names.

**XML Namespaces - The xmlns Attribute**

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

```
<root>
<h:table xmlns:h="http://www.w3.org/TR/html4/"> <h:tr>
 <h:td>Apples</h:td>
 <h:td>Bananas</h:td>
 </h:tr>
```

```
</h:table>

<f:table xmlns:f="http://www.w3schools.com/furniture">
<f:name>African Coffee Table</f:name>
<f:width>80</f:width>
<f:length>120</f:length>
</f:table>
</root>
```

- Namespaces can be declared in the elements where they are used or in the XML root element: <root xmlns:h="http://www.w3.org/TR/html4/" xmlns:f="http://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>
```

**Uniform Resource Identifier (URI)**

- A **Uniform Resource Identifier** (URI) is a string of characters which identifies an Internet Resource.
- The most common URI is the **Uniform Resource Locator** (URL) which identifies an Internet domain address. Another, not so common type of URI is the **Universal**

**Resource Name**(URN)

**3.3Introduction to DTD**

- A **Document Type Definition (DTD)** defines the legal building blocks of an XML document
- It defines the document structure with a list of legal elements and attributes.
- The purpose of a DTD is to define the structure of an XML document

- Types

❖·Internal DTD ❖·External DTD

**Internal DTD Declaration**

- The DTD is declared inside the XML file
- It should be wrapped in a DOCTYPE definition
  - Syntax

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

**Example**

```
<?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>
```

Explanation

The DTD above is interpreted like this:

- **!DOCTYPE note** defines that the root element of this document is note
- **!ELEMENT note** defines that the note element contains 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"

**External DTD Declaration**

- The DTD is declared in an external file
- It should be wrapped in a DOCTYPE definition
  - **Syntax:**
    - <!DOCTYPE root-element SYSTEM "filename">

Eg

```
<?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>
```

- The file "note.dtd" which contains the DTD

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

- With a DTD, each of your XML files can carry a description of its own format.
- With a DTD, independent groups of people can agree to use a standard DTD for interchanging data.
- Your application can use a standard DTD to verify that the data you receive from the outside world is valid.
- You can also use a DTD to verify your own data.

### 3.4 Defining DTD entities - Defining Parameter entities

**ENTITY**

ENTITYs are used to reference data that act as an abbreviation or can be found at an external location. The first character of an ENTITY value must be a letter, '_', or ':' .

Example

```
<?xml version="1.0" ?>
<!DOCTYPE experiment_a [
<!ELEMENT experiment_a (results)*>
<!ELEMENT results EMPTY>
<!ATTLIST results image ENTITY #REQUIRED>
<!ENTITY a SYSTEM "http://www.university.com/results/experimenta/a.gif"> ]>
<experiment_a> <results image="a"/>
<experiment_a>
```

**ENTITIES**

Allows multiple ENTITY names separated by whitespace.

Example:

```
<?xml version="1.0“ ?>
<!DOCTYPE experiment_a [
<!ELEMENT experiment_a (results)*> <!ELEMENT results EMPTY>
<!ATTLIST results images ENTITIES #REQUIRED>
<!ENTITY a1 SYSTEM "http://www.university.com/results/experimenta/a1.gif">
<!ENTITY a2 SYSTEM "http://www.university.com/results/experimenta/a2.gif">

<experiment_a>
<results images="a1 a2 "/>
</experiment_a>
```

## 3.5 Working with attributes - Defining attributes - Defining multiple attributes - Using predefined attributes

- Attributes are additional information associated with an element type
- They are intended for interpretation by an application.
- The ATTLIST declarations identify which element types may have attributes, what type of attributes they may be, and what the default value of the attributes are.

- Syntax:

```
<!ATTLIST element_name attribute_name attribute_type default_value>
        .
        .
        .
<element attribute_name="attribute_value">
```

**where:-**

element_name: name of the element to which the attribute applies.

- **·Example:**

```
<?xml version="1.0"?>
<!DOCTYPE image [ <!ELEMENT image EMPTY>
<!ATTLIST image height CDATA #REQUIRED> <!ATTLIST image width CDATA
#REQUIRED>
]>
<image height="32" width="32"/>
```

- **Rules:**

✓All attributes used in an XML document must be declared in the Document Type Definition (DTD) using an Attribute- List Declaration .

✓Attributes may only appear in start or empty tags .

✓The keyword ATTLIST must be in upper case

**Default values**

✓The "default_value" signifies whether an attribute is required or not, and if not, what default value should be displayed.

✓The possible default values are listed in below.

| Default Value: | Description: |
|---|---|
| #REQUIRED | The attribute must always be included . |
| #IMPLIED | The attribute does not have to be included. |
| #FIXED "Default_Value" | The attribute must always have the default value that is specified . If the attribute is not physically added to the element tag in the XML document, the XML processor will behave as though the default value does exist. |

**Attribute types**

- **There are three main attribute types.**

1.A string type

2.A tokenized types

3.A enumerated types

**1.A string type:-**

**CDATA :-**

- ·CDATA stands for character data, that is, text that does not form markup.

Example:

```
<?xml version="1.0"?>
<!DOCTYPE image [
<!ELEMENT image EMPTY>
<!ATTLIST image height CDATA #REQUIRED>

<!ATTLIST image width CDATA #REQUIRED> ]>
<image height="32" width="32"/>
```

**2. Tokenized Attribute Type**

**ID :-**

ID is a unique identifier of the attribute. IDs of a particular value should not appear more than once in an XML document .

An element type may only have one ID attribute . An ID attribute can only have an #IMPLIED or #REQUIRED default value . The first character of an ID value must be a letter, '_', or ':' .

Example

```
<?xml version="1.0"?> <!DOCTYPE student_name [
<!ELEMENT student_name (#PCDATA)>
<!ATTLIST student_name student_no ID #REQUIRED>
]>
<student_name student_no="a9216735">Jo Smith</student_name>
```

**IDREF**

IDREF is used to establish connections between elements. The IDREF value of the attribute must refer to an ID value declared elsewhere in the document . The first character of an ID value must be a letter, '_', or ':' .

Example:

```
<?xml version="1.0" standalone="yes"?>
<!DOCTYPE lab_group [
<!ELEMENT lab_group (student_name)*>
<!ELEMENT student_name (#PCDATA)>
<!ATTLIST student_name student_no ID #REQUIRED> <!ATTLIST
student_name tutor_1 IDREF #IMPLIED> <!ATTLIST student_name tutor_2
IDREF #IMPLIED> ]>
<lab_group>
<student_name student_no="a8904885">Alex Foo</student_name>
<student_name student_no="a9011133">Sarah Bar</student_name>
```

```
<student_name student_no="a9216735"
tutor_1="a9011133" tutor_2="a8904885">Jo
Smith</student_name> </lab_group>
```

**IDREFS**

Allows multiple ID values separated by whitespace .

**NMTOKEN**

The first character of an NMTOKEN value must be a letter, digit, '.', '-', '_', or ':' . Example:-

```
<?xml version="1.0"?> <!DOCTYPE student_name [
<!ELEMENT student_name (#PCDATA)>
<!ATTLIST student_name student_no NMTOKEN #REQUIRED>
]>
<student_name student_no="9216735">Jo Smith</student_name>
```

**NMTOKENS**

Allows multiple NMTOKEN names separated by whitespace .

**3.Enumerated attribute type**

**NOTATION**

NOTATIONs are useful when text needs to be interpreted in a particular way, for example, by another application. The first character of a NOTATION name must be a letter, '_', or ':' .

Example:-

```
<?xml version="1.0"?> <!DOCTYPE code [
<!ELEMENT code (#PCDATA)> <!NOTATION vrml PUBLIC "VRML 1.0">
<!ATTLIST code lang NOTATION (vrml) #REQUIRED> ]>
<code lang="vrml">Some VRML instructions</code>
```

**Enumerated**

Enumerated attribute types allow you to make a choice between different attribute values. The first character of an Enumerated value must be a letter, digit, '.', '-', '_', or ':' .

Example

```
<?xml version="1.0"?>
<!DOCTYPE payment [ <!ELEMENT payment EMPTY>
<!ATTLIST payment mode (cash|cheque) “cash” #REQUIRED> ]>
<payment mode=”cash”>
                Or
<payment mode=”cheque”>
```

## 3.6 CSS basics - Adding CSS to document

What is CSS?

- CSS stands for Cascading Style Sheets
- Most popular way of formatting HTML files so that they can be easily maintained and updated
- CSS also well supported method of formatting XML document for web browsers

Basic CSS statement

- For creating style sheet, Identify the specific rules and values for each element's available properties

Syntax

: Selector { property : value}

- Selector is the element identifier
- Property is the name of one of the predefined CSS properties.
- Value is one of the predefined value types for specific property.

Example:

- company { background.color:LIGHTBLUE; font.size:12pt; }

- **To apply a series of properties values to a single element**

Syntax:

Selector { property : value; property: value;property: value;…..property: value}

Example

company { background.color:LIGHTBLUE; font.size:12pt; }

- **Setting multiple properties values for a multiple elements**

Syntax:

Selector1,selector2,selector3 {property:value; property: value}

Example:

head, company {font-family: “comic sans MS”, sans-serif; font-size:16pt; font-style:normal}

Adding css to your document

- Style sheets can only be added to xml documents through the use of a processing instruction that references the external CSS file
- CSS using a system of pattern matching to determine which CSS rules are applied to which elements that are found within the document

Syntax:

<?xml-stylesheet type=“text/css” href=“stylesheet.css”?>

- A CSS style contains a list of rules.
- Each rules gives the names of the element
- It applies the styles to those elements
- Cascading Style Sheets(CSS) is a very simple and straight forward language for applying styles to xml documents

- You can choose fonts, font weight, font size, background color, spacing between paragraph and etc…
- All style information is placed in a separate document called a style sheet
- A single xml document can be formatted different ways by charging the style sheet
- CSS is a declarative language in 1996 is a standard adding information about style properties such as fonts and borders to html elements

Example:

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/css" href="catalog.css"?> <company>
<IT>
<name>wipro</name>
<location>sholliganallur</location>
<city>chennai</city> <state>tamil nadu</state>
</IT>
<IT>
<name>TCS</name>
<location>Vadapalani</location>
<city>chennai</city> <state>tamil nadu</state>
</IT>
</company>
```

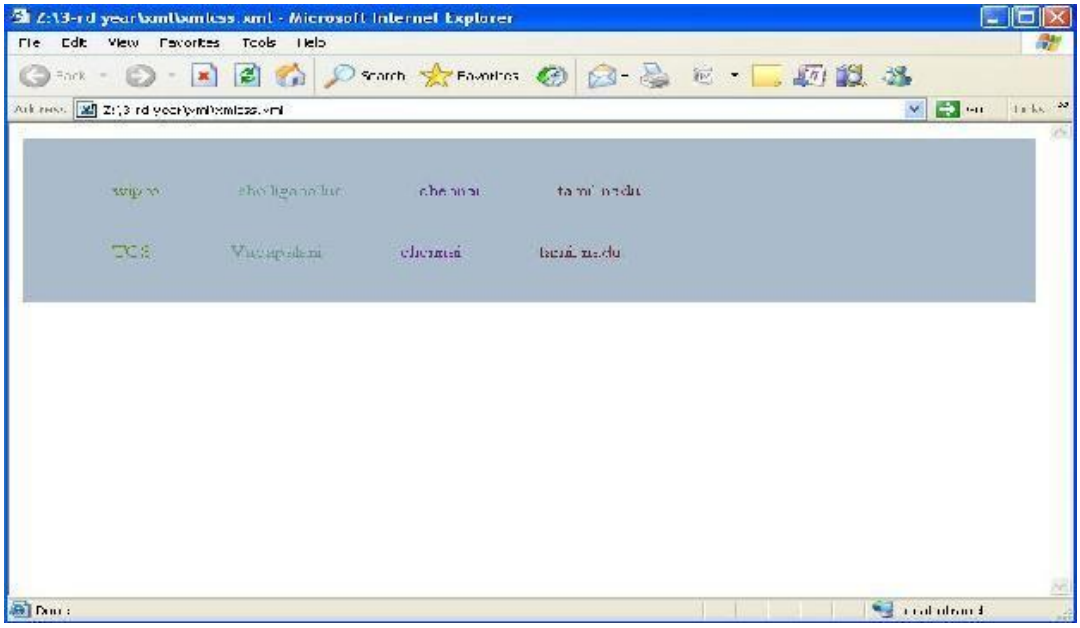
**File Name :Catalog.css**

```
company
{
background.color:#a
abbcc; width=100%
}
```

```
 IT
{
display:block;
margin.bottom:30pt;
margin.left:30pt
}
name
{
color:#678910;
margin.left:20pt;
font.size:12pt
}

Location
{
color:#689888;
margin.left:20pt
; font.size:12pt
}
city
{
color:#682391;
margin.left:20pt;
font.size:12pt
}
state
{
color:#692931;
margin.left=20pt;
font.size=12pt
```

}

Output:-




CSS SELECTOR

- CSS rule specifies which element it applies to is called a selector The simple selector is name of the element

Example:

Title { display:block; font-size:16pt;font-weight:bold } Selectors also specify multiple elements

CONTROLLING FONTS

- Font styles – control all aspects of your text appearance from font style, face, family and variants.
- Font-family Used to specify a prioritized list of font family names and / or generic family names
- Allows to identify a specific list of fonts, generally same style and size

Syntax:

Font-family: [ [ <family-name> | <generic-family> ], ]* [<family-name> | <generic-family>] | inherit

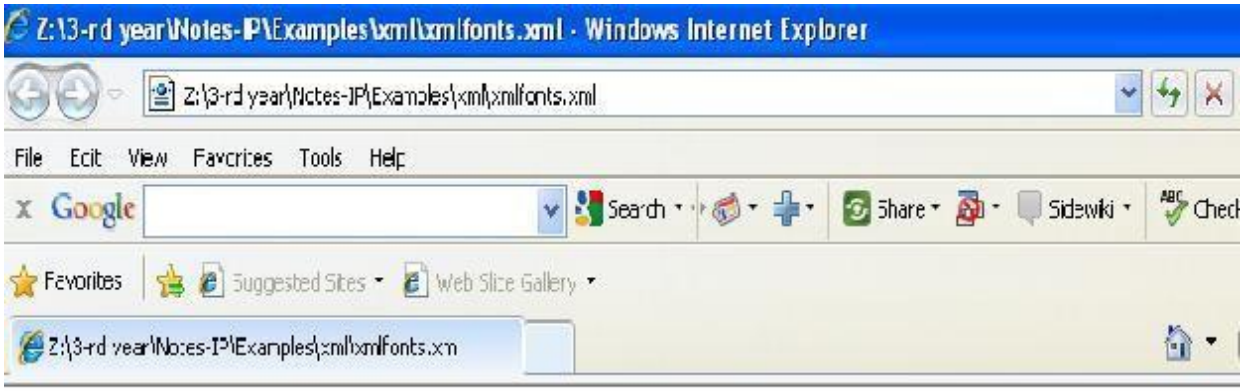
Family-name – name of the font family to use. font-family include : bookman, Lucida, and Times New Roman. All font family names within double quotes.
Generic-family – serif, sans-serif, cursive, fantasy, and monospace. because they are keywords, they cannot be included within quotation marks.

Example: xmlfonts.xml

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/css" href="font.css"?> <book>
<intro>XML complete Refrence</intro> <author>Williamson</author> <publisher>Tata Magrawhills</publisher>
<sample>See index</sample>
</book>
```

Filename : font.css

```
book { font-family: "comic sans"; } intro {font-family: "Monotype corsiva"; }
```




*XML complete Refrence* Williamscr Tata Magrawh lls See index

author { font-family: sans-serif; } publisher { font-family: "Helvetica"; } sample { font-family: serif; }

Font-size used to identify the size of the font
This setting based upon xml browsers default font-size settings

Syntax:

font-size: <absolute-size> | <relative-size> | <length> | <percentage> | inherit

Description:
<absolute-size> - (xx-small | x-small | small | medium | large | x-large). Medium-12pt ,small-10pt, scaling factor- 1.2 between each size

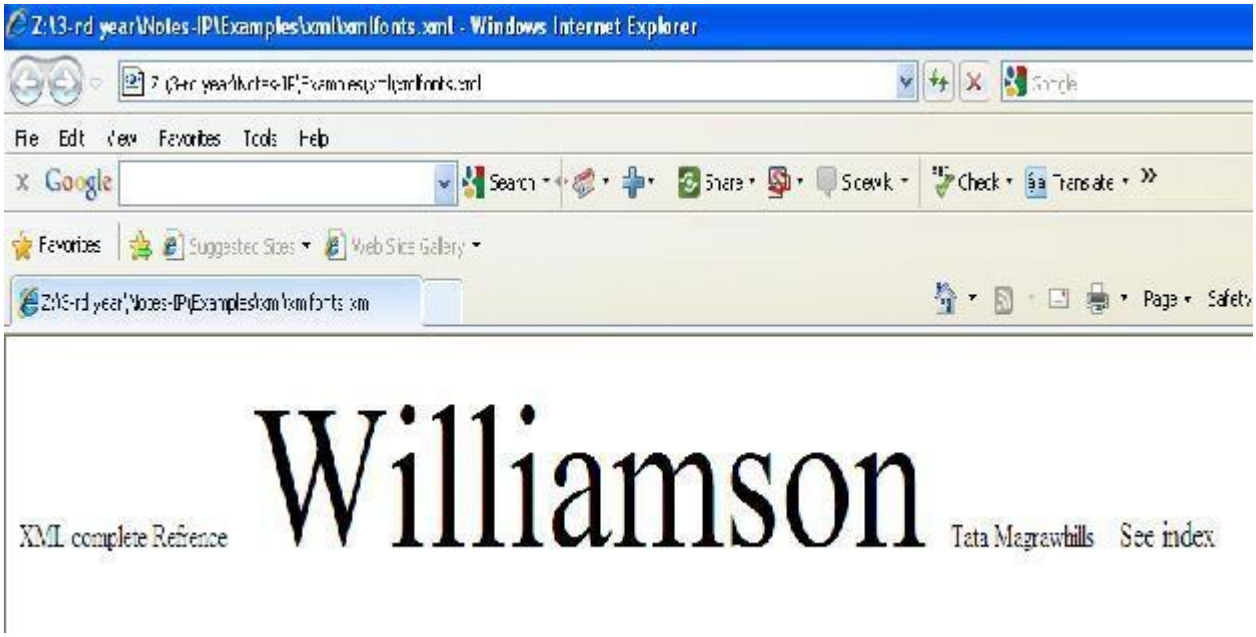
If the <relative-size> is specified (larger | smaller)
<length> - used to specify positive integer representing an absolute font size
<percentage> - to identify an absolute font size in relation to the
parent element's font size

Example:-

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/css" href="fontsize.css"?> <book>
<intro>XML complete Refrence</intro>
<author>Williamson</author>
<publisher>Tata Magrawhills</publisher>
<sample>See index</sample>
</book>
```

**fontsize.css**

**b**ook { font-size: "150"; }
intro {font-size: small }
author { font-size: larger; }
publisher { font-size: "12pt"; }
sample{ font-size: medium; }



Setting font-stretch

- Font-stretch property is used to control the kerning of your font.
- It control the amount of space found between each individual character of the font.

Syntax:

font-stretch: normal | wider | narrower | ultra-condensed | xtra-condensed | condensed | semi-condensed | semi- expanded | expanded | extra-expanded | ultra-expanded | inherit

Description

ultra-condensed , xtra-condensed , condensed , semi-condensed , semi-expanded,expanded , extra-expanded
, ultra-expanded – these values are organized from most condensed to least condensed. Each of them is a small change in horizontal spacing of text.
Wider – this value is relative to parent element and expands spacing. Without increasing the ultra-expanded level. Narrower – decreasing spacing one step without decreasing below ultra-condensed level

Example:

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/css" href="fontstretch.css"?>
<book>
<intro>XML complete Refrence</intro>
<author>Williamson</author>
<publisher>Tata Magrawhills</publisher>
<sample>See index</sample>
</book>
```

**fontstretch.css**

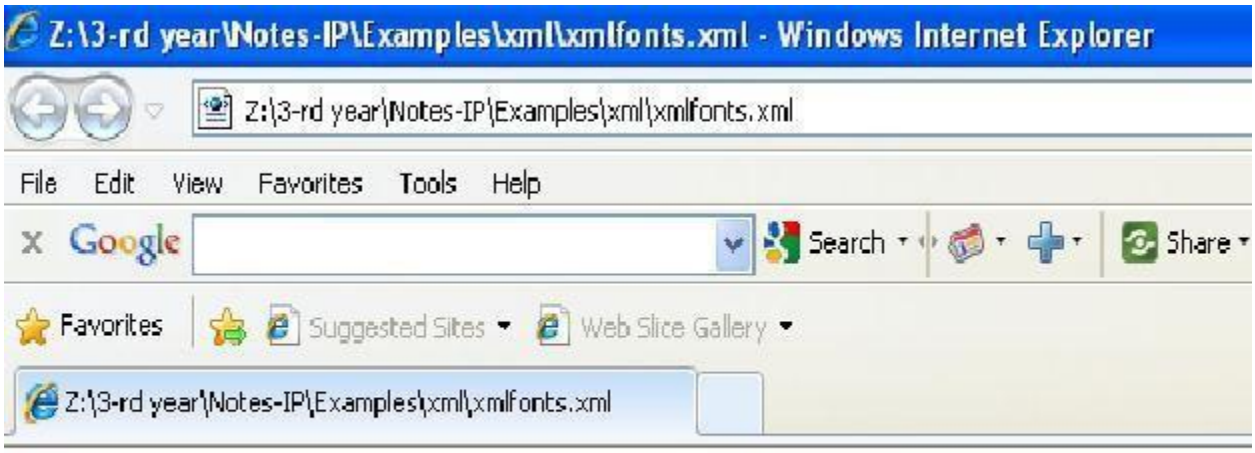
**b**ook { font-stretch: "ultra-expanded";}

intro {font-stretch: narrower; }

author { font-stretch: "expanded"; }

publisher { font-stretch: "normal"; }

sample { font-stretch: "wider"; } Output:-




XML complete Refrence Williamson Tata Magrawhills See index

Setting font-style

- Font-style property specifies the style of font being used
- You can specify normal opaque, or italic fonts

Syntax:

font-style: normal | italic | oblique | inherit

Description:-

normal →·no style change

Italic →·displays with an italic or cursive slant to each character oblique →·displays with an oblique, slanted, or inclined font

Example:

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/css" href="fontstyle.css"?> <book>
<intro>XML complete Refrence</intro> <author>Williamson</author> <publisher>Tata Magrawhills</publisher>
<sample>See index</sample>
</book>
```

fontstyle.

css

```
intro {font-style: "oblique"; } author { font-style: "italic"; } publisher { font-style: "normal"; }
```

Setting font-variant

- Font-variant property writes fonts using small caps for fonts supporting this style.
- Small caps are achieved by replacing all lower case letters with lowercase height capital letters.

Syntax:-
font-variant: normal | small-caps | inherit

Description:-
normal – the value does not change from the standard font character. small-caps – this value specifies to use small capital letters in place of standard lowercase letters within the specified text.

Example:-

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/css" href="fontvariant.css"?> <book>
<intro>XML complete Refrence</intro>
<author>Williamson</author>
<publisher>Tata Magrawhills</publisher>
<sample>See index</sample>
</book>
```

**fontvariant.css**

```
intro {font-variant: "small-caps"; }
```

Setting font color

color property identifies the foreground color for the text content of an element.

Syntax
color:<colorname> | <RGBcolor>

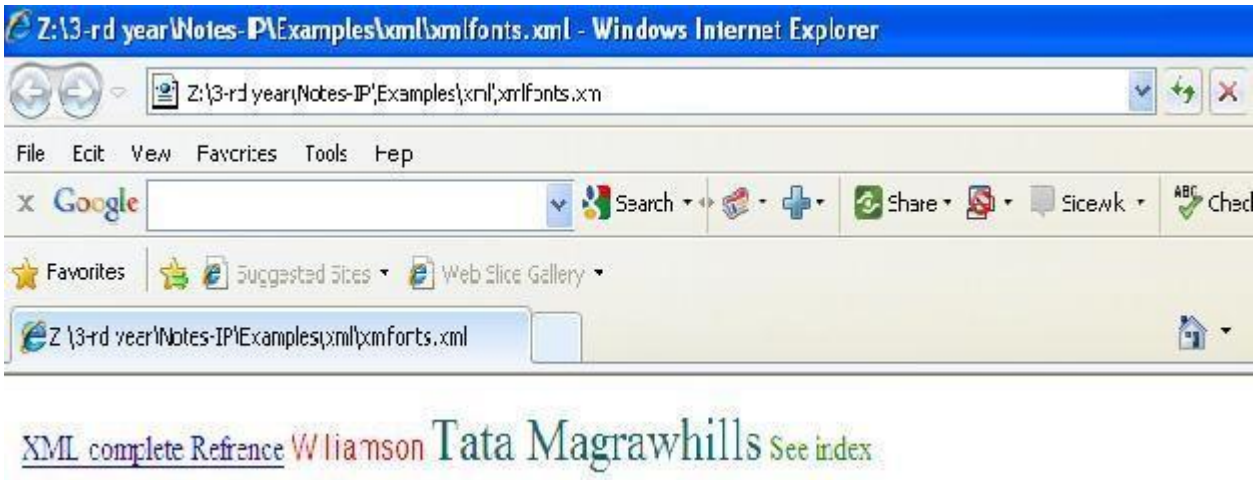
Description:-
colorname →·one of the valid color names.
RGBcolor→·name of the color returned in hexadecimal format representing Red- Green-Blue(RGB) value of the color

Example:-

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/css" href="fontcolor.css"?> <book>
<intro>XML complete Refrence</intro>
<author>Williamson</author>
<publisher>Tata Magrawhills</publisher>
<sample>See index</sample>
</book>
```

**fontcolor.css**

```
author {font-family: arial, helvetica, sans-serif; color:#FF0000 }
intro { text-decoration: underline; color: Blue}
publisher { font-family: "times new roman"; font-size:"20pt"; color: rgb(10,100,100)}
sample { color: rgb(25,150,0)} Output:-
```



Font-weight

Used to specify the weight or thickness of font to display a specified text Syntax:

font-weight: 100 | 200 | 300 | 400 | 500 | 700 | 800 | 900 | normal |bold | bolder | bolder | lighter | inherit

Description:-

Normal →·value is typically same as value of

400 Bold→·same as value 700

Bolder →darker than the inherited one , increased by 100.

Lighter→·lighter than inherited one

Example:

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/css" href="fontweight.css"?> <book>
<intro>XML complete Refrence</intro> <author>Williamson</author>
<publisher>Tata Magrawhills</publisher>
<sample>See index</sample>
</book>
```

**Fontweight.css**

```
book { font-weight: "400"; } intro { font-weight:"lighter";}sample { font-weight:"bold";}
```

3.7 Need of XSL - XSL basics - XSL transformations

XSL(EXTENSIBLE STYLESHEET LANGUAGE)

XSL is a stylesheet technology which address two important aspects

1.Transform the document from one type to another (XSLT)
2.Styling the document according to the formatting rules

XSLT stands for XSL Transformation XSLT is the most important part of XSL
Transform the document from one type to another

Syntax:- (root element)
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl=http://www.w3.org/TR/WD-XSL>

The transformation language provides element that define rules for how one xml document is transformed into another xml element

**XSL consists of three parts:**

- XSLT - a language for transforming XML documents
- XPath - a language for navigating in XML documents
- XSL-FO - a language for formatting XML documents

**What is XSLT?**

- XSLT stands for XSL Transformations
- XSLT is the most important part of XSL
- XSLT transforms an XML document into another XML document
- XSLT uses XPath to navigate in XML documents
- XSLT is a W3C Recommendation

XSLT Uses XPath

- XSLT uses XPath to find information in an XML document. XPath is used to navigate through elements and attributes in XML documents.

**XSLT - Transformation**

The root element that declares the document to be an XSL style sheet is <xsl:stylesheet> or <xsl:transform>.
Note: <xsl:stylesheet> and <xsl:transform> are completely synonymous and either can be used!

The correct way to declare an XSL style sheet according to the W3C XSLT Recommendation is:

<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"> (or)
<xsl:transform 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.

<xsl:template> Element

- 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.
- The <xsl:template> element is used to build template
- The match attribute is used to associate a template with an xml element
- Match attribute can also be used to define a template for entire xml document
- Value of match attribute is an xpath expression ie. match="/" defines the whole document

Structure(Syntax)

```
<xsl:template match=“ ”>
…….
</xsl:templ
ate>
```

Example:-

```
<?xml version="1.0" ?>
<?xml-stylesheet type=”text/xsl”href=”books.xsl”?> <books>
<book>
<title>java</title>
<author>pattrick</author> <publisher>tata mcgraw hill</publisher> <price>500</price>
</book>
<book>
<title>asp</title>
<author>micheal</author> <publisher>tech media</publisher> <price>450</price>
</book>
</book
s>
```

books.

xsl

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

<body>

    <table border="10" bgcolor="green" cellpading="20" cellspacing="15"
    align="center">
<caption>books details</caption> <tr>
```

<th>title</th>

```
<th>author</th>
<th>publisher</th>
<th>price</th>
</tr>
</table>
</body>
</html>
</xsl:template>
</xsl:stylesheet>
```

<xsl:value-of> Element

- This is most frequently used structure
- This is used to retrieve the value of the element and displaying the request
- The <xsl:value-of> element can be used to extract the value of an XML element and add it to the output stream of the transformation

Syntax:

<xsl:value-of select=“”>

Example:-

```
<?xml version="1.0" ?>
<?xml-stylesheet type=”text/xsl”href=”books.xsl”?> <books>
<book>
<title>java</title>
<author>pattrick</author> <publisher>tata mcgraw hill</publisher> <price>500</price>
</book>
<book>
<title>asp</title>
<author>micheal</author> <publisher>tech media</publisher> <price>450</price>
</book>
</books>
```

books.xsl

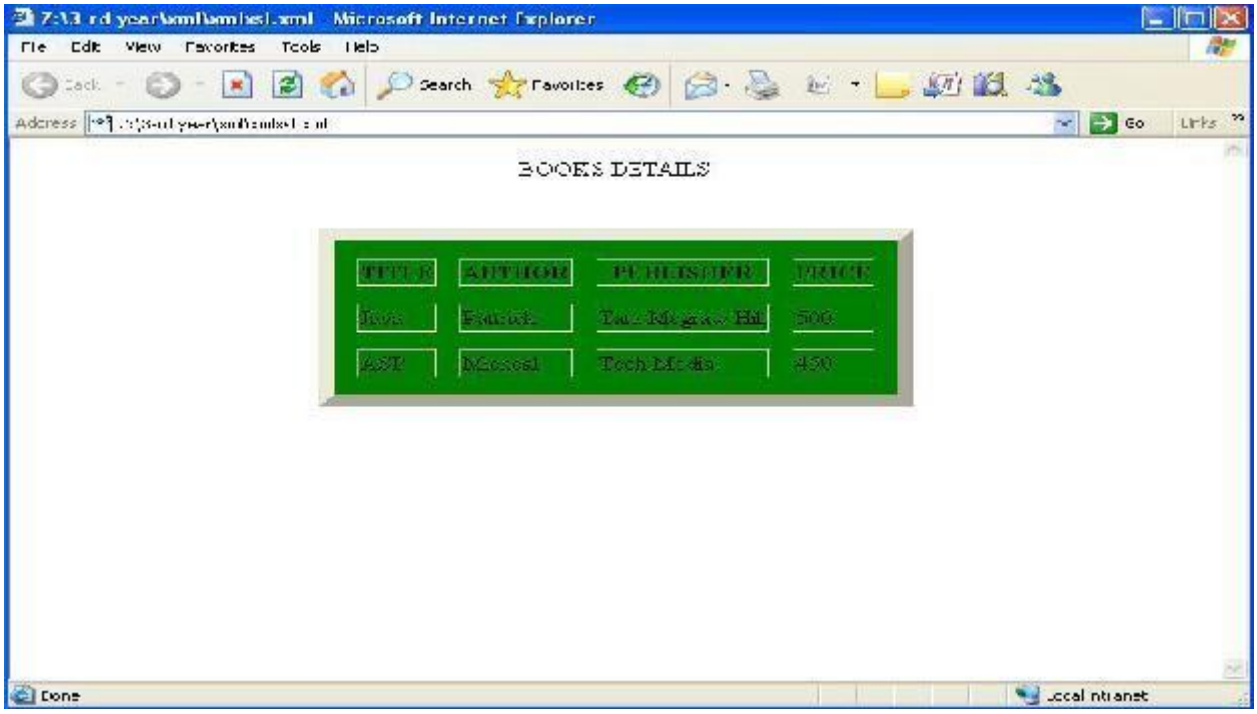
```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl"><xsl:template match="/">
<html>
<body>

<table border="10" bgcolor="green" cellpading="20" cellspacing="15" align="center">

<caption>books details</caption> <tr>
<th>title</th>
<th>author</th>
<th>publisher</th>
<th>price</th>
</tr>
<tr>
<td><xsl:value-of select ="books/book/title"/></td> <td><xsl:value-of select ="
books/book/author"/></td> <td><xsl:value-of select ="
books/book/publisher"/></td> <td><xsl:value-of select =" books/book/price"/></td>
</tr>
</table>
</body>
</html>
</xsl:template>
</xsl:stylesheet>
```

**<xsl:for-each> Element**

The <xsl:for-each> element allows you to do looping in XSLT.
The XSL <xsl:for-each> element can be used to select every XML element of a specified node-set:

Syntax:
<xsl:for-each select=“”>
…….
</xsl:for-
each>
Example:-

```
<?xml version="1.0" ?>
<?xml-stylesheet type=”text/xsl”href=”books.xsl”?> <books>
<book>
<title>java</title>
<author>pattrick</author> <publisher>tata mcgraw hill</publisher> <price>500</price>
</book>
<book>
<title>asp</title>
<author>micheal</author> <publisher>tech media</publisher> <price>450</price>
</book>
</books>
```




books.xsl

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl"><xsl:template match="/">
<html>
<body>
    <table border="10" bgcolor="green" cellpading="20" cellspacing="15"
    align="center">
<caption>books details</caption> <tr>
<th>title</th>
<th>author</th>
<th>publisher</th>
```

```
<th>price</th>
</tr>
<xsl:for-eachselect="books/book"> <tr>
<td><xsl:value-of select ="title"/></td> <td><xsl:value-of select =" author"/></td>
<td><xsl:value-of select =" publisher"/></td> <td><xsl:value-of select ="
price"/></td>
        </tr> </xsl:for-each></table>
</body>
</html>
</xsl:template>
</xsl:stylesh
eet>
```

Output:-

<xsl:if> Element

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

Syntax:

```
<xsl:if select=“ Condition”>
…..
</xsl:if
>
```

Exampl

e:-

```
<?xml version="1.0" ?>
<?xml-stylesheet type=”text/xsl”href=”books.xsl”?> <books>
<book>
<title>java</title>
<author>pattrick</author> <publisher>tata mcgraw hill</publisher> <price>500</price>
</book>
<book>
<title>asp</title>
```

<author>micheal</author> <publisher>tech media</publisher> <price>450</price>
</book>
</book
s>
books.
xsl
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl"><xsl:template match="/">
<html>
<body>
<table border="10" bgcolor="green" cellpading="20" cellspacing="15"
align="center">
<caption>books details</caption> <tr>
<th>title</th>
<th>author</th>
<th>publisher</th>
<th>price</th>
</tr>
<xsl:for-eachselect="books/book"> <xsl:if test="price > 451">
<tr>
<td><xsl:value-of select ="title"/></td> <td><xsl:value-of select =" author"/></td>
<td><xsl:value-of select =" publisher"/></td>

<td><xsl:value-of select =" price"/></td>
</tr> </xsl:if >

</xsl:for-each></table> </body> </html> </xsl:template> </xsl:stylesheet>

<xsl:sort> Element

The <xsl:sort> element is used to sort the output.

Syntax:

```
<xsl:sort select=“ Condition”>
…..
</xsl:sort>
```

Example:-

```
<?xml version="1.0" ?>
<?xml-stylesheet type=”text/xsl”href=”books.xsl”?> <books>
<book>
<title>java</title>
<author>pattrick</author> <publisher>tata mcgraw hill</publisher> <price>500</price>
</book>
<book>
<title>asp</title>
<author>micheal</author> <publisher>tech media</publisher> <price>450</price>
</book>
</book
s>
books.
xsl
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl"><xsl:template match="/">
<html>
<body>
    <table border="10" bgcolor="green" cellpading="20" cellspacing="15"
    align="center">

<caption>books details</caption> <tr>
```

```
<th>title</th>
<th>author</th>
<th>publisher</th>
<th>price</th>
</tr>
<xsl:for-eachselect="books/book"> <xsl:sort select="title "/>
<tr>
<td><xsl:value-of select ="title"/></td> <td><xsl:value-of select =" author"/></td>
<td><xsl:value-of select =" publisher"/></td> <td><xsl:value-of select ="
price"/></td>
</tr> </xsl:sort >

</xsl:for-each></table> </body> </html> </xsl:template> </xsl:stylesheet>
```

### 3.8 Introduction to Schemas - Defining simple elements and types

XML SCHEMA

- An XML Schema describes the structure of an XML document. XML Schema is an XML-based alternative to DTD. An XML schema describes the structure of an XML document.
- The XML Schema language is also referred to as XML Schema Definition (XSD).

**What is an XML Schema?**

The purpose of an XML Schema is to define the legal building blocks of an XML document, just like a DTD.

An XML Schema:

- defines elements that can appear in a document
- defines attributes that can appear in a document
- defines which elements are child elements
- defines the order of child elements
- defines the number of child elements
- defines whether an element is empty or can include text
- defines data types for elements and attributes
- defines default and fixed values for elements and attributes

Advantages

- XML Schemas are extensible to future additions
- XML Schemas are richer and more powerful than DTDs
- XML Schemas are written in XML
- XML Schemas support data types
- XML Schemas support namespaces

Why use XML schema ?

- Schema support data types
- Easier to describe allowable document content
- Easier to validate the correctness of data
- Easier to work with data from database
- Easier to define data facets(restriction on data)
- Easier to define data pattern (data formats)
- Easier to convert data between different data types

Well-Formed XML schema

A well-formed XML document is a document that conforms to the XML syntax rules, like:

- it must begin with the XML declaration
- it must have one unique root element
- start-tags must have matching end-tags
- elements are case sensitive
- all elements must be closed
- all elements must be properly nested
- all attribute values must be quoted
- entities must be used for special characters

Schema element is the root element of every xml schema

```
<?xml version="1.0"?>
<xs:schema>
……
</xs:schema>
```

Schema declaration

```
<xs:schema
xmlns:xs=http://www.w3.org/2001/xmlschema targetNamespace=http://w3school.com
xmlns=http://www.w3school.co m
elementFormDefault="qualified">
……
……
</xs:schema>
```

Description:-

- xmlns:xs=http://www.w3.org/2001/XMLSchema

indicates that the elements and data types used in the schema come from the "http://www.w3.org/2001/XMLSchema" namespace should be prefixed with xs:

- targetNamespace=http://www.w3schools.com

indicates that the elements defined by this schema (note, to, from, heading, body.) come from the "http://www.w3schools.com" namespace.

- xmlns=http://www.w3schools.com

indicates that the default namespace is "http://www.w3schools.com".

- elementFormDefault="qualified"

indicates that any elements used by the XML instance document which were declared in this schema must be namespace qualified.

XML SIMPLE ELEMENT

XML Schemas define the elements of your XML files.

What is a Simple Element?

A simple element is an XML element that contains only text. It cannot contain any other elements or attributes.

However, the "only text" restriction is quite misleading. The text can be of many different types. It can be one of the types included in the XML Schema definition (boolean, string, date, etc.), or it can be a custom type that you can define yourself.

syntax :

<xs:element name="xxx" type="yyy"/>

Description;-

xxx- the name of the element

yyy- the data type of the element.

XML Schema has a lot of built-in data types. The most common types are:

- xs:string
- xs:decimal
- xs:integer
- xs:boolean
- xs:date
- xs:time

Example:-

```
<lastname>Refsnes</lastname>
<age>36</age>
<dob>1970-03-27</dob><per>97.5</per>
```

*And here are the corresponding simple element definitions:*

```
<xs:element name="lastname" type="xs:string"/>
<xs:element name="age" type="xs:integer"/>
<xs:element name="dob" type="xs:date"/>
 <xs:element name="per" type="xs:decimal"/>
```

*Default and Fixed Values for Simple Elements*

<xs:element name="color" type="xs:string" default="red"/>

A default value is automatically assigned to the element when no other value is specified. In the following example the default value is "red"

<xs:element name="color" type="xs:string" fixed="red"/>

A fixed value is also automatically assigned to the element, and you cannot specify another value. In the following example the fixed value is "red"

Example:-

<?xml version="1.0"?>

<note xmlns="http://www.w3schools.com"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"xsi:schemaLocation="http://www.w3schools.com
note.xsd">
<to>Tove</to>
<from>Jani<</from>
<body>Don't forget me this weekend!</body>
</note>

**note.xsd**

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

XSD ATTRIBUTES

All attributes are declared as simple types.

What is an Attribute?

Simple elements cannot have attributes. If an element has attributes, it is considered to be of a complex type.

But the attribute itself is always declared as a simple type.

syntax :

```
<xs:attribute name="xxx" type="yyy"/>
```

Description:-

xxx- the name of the attribute

yyy- specifies the data type of the attribute.

XML Schema has a lot of built-in data types. The most common types are:

- xs:string
- xs:decimal
- xs:integer
- xs:boolean
- xs:date
- xs:time

Example:-

*Here is an XML element with an attribute:*

<lastname lang="EN">Smith</lastname>

*And here is the corresponding attribute definition:*

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

Default and Fixed Values for Attributes

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

A default value is automatically assigned to the attribute when no other value is specified. In the following example the default value is "EN":

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

A fixed value is also automatically assigned to the attribute, and you cannot specify another value.In the following example the fixed value is "EN":

XSD Complex Elements

A complex element contains other elements and/or attributes.

What is a Complex Element?

A complex element is an XML element that contains other elements and/or attributes. There are four kinds of complex elements:

- empty elements
- elements that contain only other elements
- elements that contain only text
- elements that contain both other elements and text

Note: Each of these elements may contain attributes as well!
Example:
A complex XML element, "product", which is empty:
<product pid="1345"/>

A complex XML element, "employee", which contains only other elements:

```
<employee>
<firstname>John</firstname>
<lastname>Smith</lastname>
</employee>
```

A complex XML element, "food", which contains only text:

<food type="dessert">Ice cream</food>

A complex XML element, "description", which contains both elements and text:

<description>
It happened on <date lang="norwegian">03.03.99</date> ....
</description>

Example Program:

Simple XML Element

<employee>
<firstname>John</firstname>
<lastname>
</employee>

XSD format

<xs:element name="employee">
<xs:complexType>
<xs:sequence>
<xs:element name="firstname" type="xs:string"/>
<xs:element name="lastname" type="xs:string"/> </xs:sequence>
</xs:complexType>
</xs:element>

## 3.9 XML QUERY (XQUERY)

- Xquery is for XML like SQL for database
- Xquery is used to finding & Extracting elements and attributes from XML document
- XQuery is *the* language for querying XML data
- XQuery for XML is like SQL for databases
- XQuery is built on XPath expressions
- XQuery is supported by all major databases
- XQuery is a W3C Recommendation

### XQuery Basic Syntax Rules

- XQuery is case-sensitive
- XQuery elements, attributes, and variables must be valid XML names
- An XQuery string value can be in single or double quotes

- An XQuery variable is defined with a $ followed by a name

Ex. $booklist XML Query (Xquery) – booklist.xml

```
<booklist>
<book>
<title>XML Complete Reference</title> <author>Williamson</author> <price>499</price>
</book>
```

<book>
<title>JavaScript Complete Reference</title> <author>Thomas Pawel</author>
<price>400</price>
</book>
<book>
<title>Java Complete Reference</title> <author>Pattrick</author> <price>440</price>
</book>
<booklist>

For opening a particular file we use doc() function extract data from xml document

Path expression used to navigate to elements in XML document

Syntax:

doc("booklist.xml")/booklist/book/title this will extract

<title>XML Complete Reference</title> <title>JavaScript Complete Reference</title>
<title>Java Complete Reference</title>

Predicates

This is used to limit the extracted information from the document.

Syntax:

doc("booklist.xml")/booklist/book[price<450]

Output:

<title>JavaScript Complete Reference</title>
<title>Java Complete Reference</title>

FLWOR expression

- for - (optional) binds a variable to each item returned by the in expression
- let - (optional)
- where - (optional) specifies a criteria
- order by - (optional) specifies the sort-order of the result
- return - specifies what to return in the result

Example:-
FLWOR – acronym for “For”, “Let”, “Where”, “orderedby”, “Return”

How to select the node using FLWOR function for $x in doc("booklist.xml")/booklist/book where $x/[price<450] order by $x/title return $x/title

- The for clause selects all book elements under the booklist element into a variable called $x
- The where clause selects only book elements with a price element with a value less than 450
- The order by clause defines the sort-order. Will be sort by the title element
- The return clause specifies what should be returned

Here it returns the title elements

<title>Java Complete Reference</title>
<title>JavaScript Complete Reference</title>

Present the result in a html list
Xquery

```
<ul>
{
for $x in doc("booklist.xml")/booklist/book/title order by $x
return<li>{$x}</li>
```

```
}</ul>
```

The result of the above will be

```
<ul>
<li> <title>XML Complete Reference</title></li> <li><title>Javascript Complete
Reference</title></li>
<li><title>Java Complete Reference</title></li>
</ul>
```

Now we want to eliminate the title element, and show only the data inside the title element:

```
<ul>
{
for $x in doc("booklist.xml")/booklist/book/title order by $x
return<li>{data($x)}</li>
}
</ul>
```

The result will be (an HTML list):

```
<ul>
<li> XML Complete Reference</li> <li>Javascript Complete Reference</li>
<li>Java Complete Reference</li>
</ul>
```