Web Development & Designing (203105353)

Prof. Mosam Patel, Assistant Professor,
Department of Information Technology

CHAPTER-4

XML

Introduction

- XML is an eXtensible Markup Language.
- It is used to store and transport data.
- XML is used for distributing data over the Internet
- XML was designed to be self-descriptive.
- XML was designed to be machine- and human- readable.

HTML V/S XML

➢HTML and XML were designed with below goals:

HTML was designed to display data - focus on how data looks
XML was designed to carry data - focus on what data is

HTML works with predefined elements like <h2>, <div>, <b>,<body>,<p> etc.
XML tags are not predefined like HTML tags are

With XML, the author must define both the tags and the document structure.

Use of XML

XML is used for distributing data over the Internet.
XML is used in many aspects of web development.
XML is often used to separate data from presentation.
XML tags are not predefined like HTML tags are
XML was designed to carry data - with focus on what data is
Most XML applications will work as expected even if new data is added (or removed)
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.

Tree Structure of XML

An XML tree starts at a root (parent) element and branches from the root to child elements.

All elements can have child elements and child elements can have sub child elements:

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

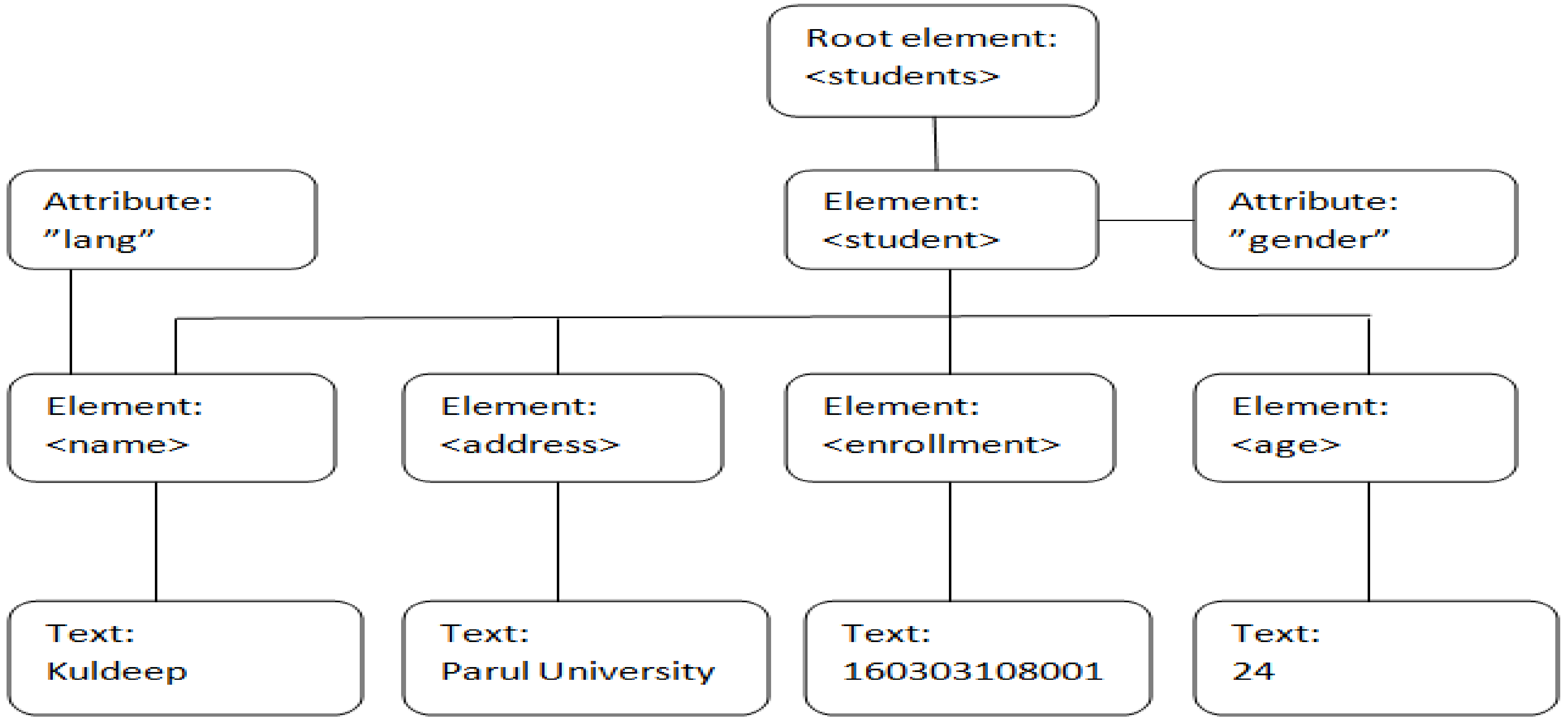
The terms parent, child, and sibling are used to describe the relationships between elements. Parents have children. Children have parents. Siblings are children on the same level (brothers and sisters).

Tree Structure of XML(Contd..)

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

```
<students>
  <student>
    <enrollment>160303108001
</enrollment>
    <name>Kuldeep</name>
     <batch>2016</batch>
    <age>24</age>
  </student>
  <student>
    <enrollment>150303108002</enr
ollment>
    <name>Hiren</name>
     <batch>2015</batch>
    <age>25</age>
  </student>
</students>
```

Tree Structure of XML(Contd..)



Syntax Rules of XML

- XML Documents Must Have one and only one root element.
- Each & Every element must have both tag(start and an end tag)
- XML Tags are Case Sensitive.
- XML Elements Must be Properly Nested
- The XML Prolog
- XML Attribute Values Must Always be Quoted
- We use Entity References.
- We can use Comments in XML.
- All the White-space is Preserved in XML
- Names (used for tags and attributes) must begin with a letter or underscore, and can consist of letters & digits.

XML key components

1. XML Element:

- XML Eléments are part of the XML documents.

  Ex. <name>Kuldeep</name>
- An XML element can contain: attributes, text, other elements or a mix of this.
- It is called Empty Element when an element with no content.

  Ex. <age></age>
- We can also use self-closing elements/tag.

  Ex. <name/>

XML key components(Contd..)

2. XML Attributes:

- XML elements can have attributes, like HTML.
- Attributes are designed to contain data related to a specific element.

- Attribute values must always be quoted. Either single or double quotes can be used.

  Ex. <student gender="male">   or   <student gender='male'>
- If the attribute value itself contains double quotes you can use single quotes, like in this example:

  Ex. <name=‘Hiren “Rameshbhai" Patel'>

XML key components(Contd..)

Attributes v/s Elements:

- Attributes have no structure, It is simple strings.
- Elements may have **attributes**.

  Ex. <name gender="male">.
- As a *thumb rule is* Content into elements & Metadata into attributes

Example:

1)

```
<student gender="female">
   <firstname>Priyanka</firstname>
   <lastname>Patel</lastname>
</student>
```

2)

```
<student>
       <gender>female</gender>
       < firstname>Priyanka</firstname>
       <lastname>Patel</lastname>
</student>
```

- There are no rules about when to use attributes or when to use elements in XML.

XML key components(Contd..)

3.Namespaces:

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

Name Conflicts:

- In XML, element names are defined by the user/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>
```

XML key components(Contd..)

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

```
 <table>
         <name>Tea Table</name>
         <width>70</width>
         <length>100</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.

- A user or an XML application will not know how to handle these differences.

XML key components(Contd..)

Using a name Prefix we can solved Name Conflict

- The below example 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>Tea Table</f:name>
    <f:width>70</f:width>
    <f:length>100</f:length>
  </f:table>
  ```
- So, there will be no conflict because the two <table> elements have different names.

XML key components(Contd..)

xmlns Attribute:

- A namespace for the prefix must be defined, When we use prefixes in XML.
- It can be defined by an **xmlns** attribute in the start tag of an element.

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="https://www.w3schools.com/furniture">
       <f:name>Tea Table</f:name>
       <f:width>70</f:width>
       <f:length>100</f:length></f:table>  </root>
```

XML key components(Contd..)

- Namespaces can also be declared in the XML root element:

```
<root xmlns:h="http://www.w3.org/TR/html4/"
        xmlns:f="https://www.w3schools.com/furniture">
  <h:table>
      <h:tr>
          <h:td>Apples</h:td>
          <h:td>Bananas</h:td>
          </h:tr>
      </h:table>
  <f:table>
      <f:name>Tea Table</f:name>
      <f:width>70</f:width>
     <f:length>100</f:length>
     </f:table>
 </root>
```

XML key components(Contd..)

- The purpose of using an URI is to give the namespace a unique name.
- 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 Uniform Resource Name (URN).

Default Namespaces:

- Defining a default namespace for an element saves us from using prefixes in all the child elements. It has the following

syntax: xmlns="*namespaceURI*"

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

XML key components(Contd..)

4.Processing Instructions:

- Processing instructions allow documents to contain instructions for applications.
- PIs can be used to pass information to applications. PIs can appear anywhere in the document.
- Processing instructions are not part of the character data of the document, but MUST be passed through to the application.
- They can appear in the prolog, including the document type definition (DTD), in textual content, or after the document.

Parul® University

DIGITAL LEARNING CONTENT

XML key components(Contd..)

Syntax:

```
<?target instructions?>
```

Where

- **target** – Identifies the application to which the instruction is directed.
- **instruction** – A character that describes the information for the application to process.
- A PI starts with a special tag **<?** and ends with **?>**. Processing of the contents ends immediately after the string **?>** is encountered.
- A PI can contain any data except the combination **?>**, which is interpreted as the closing delimiter.

XML key components(Contd..)

- There are two examples of valid Processing instructions:

  <?welcome to Parul University?>   or   <?welcome?>

Example:

```
<?xml-stylesheet href = “parul.css" type = "text/css"?>
```

- Here, the *target* is *xml-stylesheet*. *href=“parul.css"* and *type="text/css"* are *data* or *instructions* the target application will use at the time of processing the given XML document.

- In this case, a browser recognizes the target by indicating that the XML should be transformed before being shown;

- The first attribute states that the type of the transform is XSL and the second attribute points to its location.

XML key components(Contd..)

5.XML Parser:

- The Document Object Model (DOM) defines the properties and methods for accessing and editing XML.
- The XML DOM defines a standard way for accessing and manipulating XML documents. It presents an XML document as a tree-structure.
- However, before an XML document can be accessed, it must be loaded into an XML DOM object.
- All modern browsers have a built-in XML parser that can convert text into an XML DOM object.

Parul® University

DIGITAL LEARNING CONTENT

XML key components(Contd..)

Parsing a Text String:<html> <body><p id="parul"></p>

```
  <script>
      var t, p, xDoc;
      t = "<students><student>" +
              "<name>Priyanka Patel</name>" +
              "<batch>2015</batch>" +
              "<age>23</age>" +
              "</student></students>";
      p = new DOMParser();
      xDoc = p.parseFromString(t,"text/xml");
document.getElementById("parul").innerHTML =
                    xDoc.getElementsByTagName("name")[0].childNodes[0].nodeValue;
   </script></body></html>
```

XML key components(Contd..)

6. CDATA Sections [Character Data]

- CDATA is defined as blocks of text that are not parsed by the parser.
- The predefined entities such as **&gt;, &lt;,** and **&amp;** require typing and are generally difficult to read in the markup. In such cases, CDATA section can be used.
- By using CDATA section, you are commanding the parser that the particular section of the document contains no markup and should be treated as regular text.

Syntax:

```
<![CDATA[ characters with markup ]]>
```

XML key components(Contd..)

Example:

- Each character written inside the CDATA section is ignored by the parser.

```
<script>
        <![CDATA[ <note> Welcome to Parul University </note> ]] >
</script >
```

- Everything between <note> and </note> is treated as character data and not as markup.

CDATA Rules:

1. CDATA cannot contain the string "]]>" anywhere in the XML document.
2. Nesting is not allowed in CDATA section.

XML key components(Contd..)

Character Entities

- HTML and XML, have some symbols reserved for their use, which cannot be used as content in XML code.
- There are few special characters or symbols which are not available to be typed directly from the keyboard.
- **Types of Character Entities:**

  A. Predefined Character Entities

  B. Numbered Character Entities

  C. Named Character Entities

XML key components(Contd..)

A. Predefined Character Entities

- They are use to avoid the ambiguity while using some symbols.

For example,

less than (**<**) or greater than (**>**) symbol is used with the angle tag (**<>**).

- Following is a list of pre-defined character entities from XML specification.

Ampersand – **&amp;**

Single quote – **&apos;**

Greater than – **&gt;**

Less than – **&lt;**

Double quote – **&quot;**

XML key components(Contd..)

B. Numeric Character Entities

- The numeric reference is used to refer to a character entity. Numeric reference can either be in decimal or hexadecimal format. As there are thousands of numeric references available, these are a bit hard to remember.
- General syntax for decimal/hexadecimal numeric reference is :

&# decimal/hexadecimal number ;

Entity Name	Character	Decimal reference	Hexadecimal reference
quot	“	&#34	&#x22;
amp	&	&#38	&#x26;
lt	<	&#60	&#x3C;
gt	>	&#62	&#x3E;
apos	‘	&#39	&#x26;

XML key components(Contd..)

3.Named Character Entity

- As it is hard to remember the numeric characters, the most preferred type of character entity is the named character entity.

- Here, each entity is identified with a name.

XML - Validation

- Validation is a process by which an XML document is validated.
- An XML document is called valid if its contents match with the elements, attributes and associated DTD, and if the document complies with the constraints expressed in it.
- Validation is dealt in two ways by the XML parser.

They are –

1. Well-formed XML document
2. Valid XML document

XML – Validation (Contd..)

1.Well-formed XML Document:

- It follow the following rules then it is **well-formed**:
- It must follow the ordering of the tag.
- Each of its opening tags must have a closing tag or it must be a self ending tag.
- It must have only one attribute in a start tag, which needs to be quoted.
- Non DTD XML files must use the predefined character entities.

XML – Validation (Contd..)

Example:

```
<?xml version = "1.0" encoding = "UTF-8" ?>
<!DOCTYPE data
    [ <!ELEMENT data (name,address,mobile)>
       <!ELEMENT name (#PCDATA)>
       <!ELEMENT address (#PCDATA)>
       <!ELEMENT mobile (#PCDATA)>
   ]>
  <data> <name>Hiren Parikh</name>
   <address>vadodara</address>
   <mobile>9473626728</mobile>
   </data>
```

XML – Validation (Contd..)

2.Valid XML document

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

XML – DTD

- A DTD is a Document Type Definition. It is defines the structure and the legal elements and attributes of an XML document.
- DTDs check vocabulary and validity of the structure of XML documents against grammatical rules of appropriate XML language.
- An XML document with correct syntax is called "Well Formed".
- An XML document validated against a DTD is both "Well Formed" and "Valid".

XML – DTD(Contd..)

- A DTD is a Document Type Definition. It is defines the structure and the legal elements and attributes of an XML document.
- DTDs check vocabulary and validity of the structure of XML documents against grammatical rules of appropriate XML language.
- An XML document with correct syntax is called "Well Formed".
- An XML document validated against a DTD is both "Well Formed" and "Valid".

Syntax:

```
<!DOCTYPE element DTD identifier
 [
     declaration1 declaration2 ........
 ]>
```

XML – DTD(Contd..)

- There are two types of DTD:
  1. Internal DTD
  2. External DTD

1. Internal DTD

- DTD is declared inside the XML file it’s called Internal DTD, it must be wrapped inside the <!DOCTYPE> definition:

Syntax of internal DTD :

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

Parul® University

DIGITAL LEARNING CONTENT

XML – DTD(Contd..)

Example:
```
<?xml version="1.0"?>
    <!DOCTYPE student [
    <!ELEMENT student (name, address, age, marks)>
    <!ELEMENT name (#PCDATA)>
    <!ELEMENT address (#PCDATA)>
    <!ELEMENT age (#PCDATA)>
    <!ELEMENT marks (#PCDATA)>
    ]>
    <student>
    <name>Hiren</name>
    <address>Vadodara</address>
    <age>24</age>
    <marks>82</marks>
    </student>
```

Parul® University

DIGITAL LEARNING CONTENT

XML – DTD(Contd..)

2.External DTD :

- If the DTD is declared in an external file, the <!DOCTYPE> definition must contain a reference to the DTD file:

Syntax:

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

- where *file-name* is the file with *.dtd* extension.

XML – DTD(Contd..)

Example:<?xml version="1.0"?>

```
<!DOCTYPE note SYSTEM “parul.dtd">
 <student>
<name>Hiren</name>
<address>Vadodara</address>
<age>24</age>
<marks>82</marks>
</student>
```

- Below file **“parul.dtd",** which contains the DTD:

```
<!ELEMENT student (name, address, age, marks)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT address (#PCDATA)>
<!ELEMENT age (#PCDATA)>
<!ELEMENT marks (#PCDATA)>
```

Parul® University

DIGITAL LEARNING CONTENT

XML – DTD(Contd..)

Using DTD for Entity Declaration

- A doctype declaration can also be used to define special characters and character strings, used in the document:

-
  ```
  <?xml version="1.0" encoding="UTF-8"?>
  <!DOCTYPE student[
  <!ENTITY nbsp "&#xA0;">
  <!ENTITY author“Name: Hiren.">
  < !ENTITY text “Parul University!">
  ]>
  < student>
  <name>Hiren<name>
  <address>Vadodara</address>
  < age>24</age>
  < footer>&author;&nbsp;&text;</footer>
  < /student>
  ```

Parul® University

DIGITAL LEARNING CONTENT

XML – DTD(Contd..)

Using DTD for Entity Declaration

- A doctype declaration can also be used to define special characters and character strings, used in the document:
-
```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE student[
<!ENTITY nbsp "&#xA0;">
<!ENTITY author“Name: Hiren.">
< !ENTITY text “Parul University!">
]>
< student>
<name>Hiren<name>
<address>Vadodara</address>
< age>24</age>
< footer>&author&nbsp;&text;</footer>
< /student>
```

XML – DTD(Contd..)

Output:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE note>
< student>
 <name>Hiren</name>
 <address>Vadodara</address>
 < age>24</age>
 < footer >Name: Hiren. Parul University!</footer>
 < /note>
```

XML – DTD(Contd..)

DTD Elements:

1.Declaring Only One Occurrence of an Element

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

2. **Declaring Minimum One Occurrence of an Element**

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

3. **Declaring Zero or More Occurrences of an Element**

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

4. **Declaring Zero or One Occurrences of an Element**

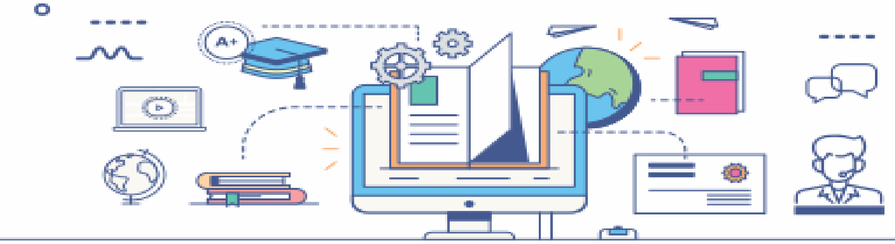
```
<!ELEMENT note (message?)>
```

5. **Declaring either/or Content**

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

6. **Declaring Mixed Content**

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



XML Schema

- It is describes the structure of an XML document.
- The XML Schema language is also known as XML Schema Definition (XSD).
- It is define the legal building blocks of an XML document:
- the elements and attributes that can appear in a document
- the number of (and order of) child elements
- data types for elements and attributes
- default and fixed values for elements and attributes.

Why Learn XML Schema?

- There are hundreds of standardized XML formats are in daily use.
- Many of these XML standards are defined by XML Schemas.
- XML Schema is an XML-based (and more powerful) alternative to DTD.

XML Schema(Contd..)

XML Schemas Support Data Types

- It is easier to describe allowable document content
- It is easier to validate the correctness of data
- It is easier to define data facets (restrictions on data)
- It is easier to define data patterns (data formats)
- It is easier to convert data between different data types

Parul® University

DIGITAL LEARNING CONTENT

XML Schema(Contd..)

The <schema> Element

- The <schema> element is the root element of every XML Schema.

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

- The <schema> element may contain some attributes. A schema declaration often looks something like this: <?xml version="1.0"?>

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
 targetNamespace="https://www.w3schools.com"
 xmlns="https://www.w3schools.com"
 elementFormDefault="qualified">
 ......
</xs:schema>
```

XML Schema(Contd..)

XSD Simple Elements

- A simple element is an XML element that contains only text. It cannot contain any other elements or attributes.
- The syntax for defining a simple element is:

  <xs:element name=“aaaa" type=“bbbb"/>

- where aaaa is the name of the element and bbbb is the data type of the element.
- XML Schema has a lot of built-in data types. They are:
- xs:string
- xs:decimal
- xs:integer
- xs:boolean
- xs:date
- xs:time

XML Schema(Contd..)

Example:

```
<firstname>Kuldeep</firstname>
<age>24</age>
<brithdate>1994-03-22</brithdate>
```

- Below is corresponding simple element definitions:

```
<xs:element name=“firstname" type="xs:string"/>
<xs:element name="age" type="xs:integer"/>
<xs:element name=“birthdate" type="xs:date"/>
```

XML Schema(Contd..)

XSD How To?

- XML documents can have a reference to a DTD or to an XML Schema.
- A Simple XML Document called “parul.xml”

```
<?xml version="1.0"?>
 <student>
 <name>Hiren</name>
 <address>Vadodara</address>
 <age>24</age>
 <marks>82</marks>
 </student>
```

XML Schema(Contd..)

- **A DTD File called “parul.dtd”** that defines the elements of the XML document previous(“parul.xml"):

```
<!ELEMENT student (name, address, age, marks)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT address (#PCDATA)>
<!ELEMENT age (#PCDATA)>
<!ELEMENT marks (#PCDATA)>
```

XML Schema(Contd..)

- **An XML Schema file called “parul.xsd”** that defines the elements of the XML document previous (“parul.xml"):

-
  ```
  <?xml version="1.0"?>  <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  targetNamespace="https://www.w3schools.com"
  xmlns="https://www.w3schools.com"  elementFormDefault="qualified">
  <xs:element name=“student">
    <xs:complexType>
      <xs:sequence>
        <xs:element name=“name" type="xs:string"/>
        <xs:element name=“address" type="xs:string"/>
        <xs:element name=“age" type=" xs:integer "/>
        <xs:element name=“marks" type=" xs:integer "/>
      </xs:sequence>
    </xs:complexType> </xs:element>   </xs:schema>
  ```

Parul® University

XSLT Introduction

- **XSLT = eXtensible Stylesheet Language Transformations**

- XSL is a styling language for XML.
- With XSLT we can transform an XML document into HTML and XHTML.

- We can add/remove elements and attributes to or from the output file using XSLT
- We can also rearrange and sort elements, perform tests and make decisions about which elements to hide and display, and a lot more.

- It is transform **an XML source-tree into an XML result-tree**.

XSLT Introduction(Contd..)

1. CSS = Style Sheets for HTML

- HTML uses predefined tags. The meaning of, and how to display each tag is well understood.
- CSS is used to add styles to HTML elements.

2. XSL = Style Sheets for XML (More Than a Style Sheet Language)

- XML does not use predefined tags, and therefore the meaning of each tag is not well understood.
- A< table> element could indicate an HTML table, a piece of furniture, or something else - and browsers do not know how to display it!
- So, XSL describes how the XML elements should be displayed.

XSLT Introduction(Contd..)

XSL consists of four parts:

1. **XSLT** is a language for transforming XML documents.

2. **XPath** is a language for navigating in XML documents.

3. **XQuery** is a language for querying XML documents.

4. **XSL-FO** is a language for formatting XML documents (discontinued in 2013)

Parul® University

XSLT Introduction(Contd..)

<xsl:template> Element

- An XSL style sheet consists of one or more number 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 templates.
- The **match** attribute is used to associate a template with an XML element.
- The match attribute can also be used to define a template for the entire XML document.
- The value of the match attribute is an XPath expression (i.e. match="/" defines the whole document).

XSLT Introduction(Contd..)

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

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
 <html><body>
 <h2>Parul Database</h2>
 <table border=“2">
  <tr bgcolor="#5acd22">
   <th>Name</th>
   <th>Age</th>
  </tr>
  <tr>
   <td>.</td>
   <td>.</td>
  </tr>
 </table></body></html></xsl:template> </xsl:stylesheet>
```

XSLT Introduction(Contd..)

Finding the message text

- The template **<xsl:template match="/">** is use to select the entire file.
- We can say that it is selecting the *root node* of the XML tree
- Inside this template,

  <xsl:value-of select="message"/> selects the message child

  - Alternative Xpath expressions *also* work like:
    - **./message**
    - **/message/text()**   **(text() is an XPath *function*)**
    - **./message/text()**

XSLT Introduction(Contd..)

<xsl:value-of> Element

- The <xsl:value-of> element is used to extract the value of a selected node.
- It is extract the value of an XML element and add it to the output stream of the transformation.
- **<xsl:value-of select="*XPath expression*"/>** selects the contents of an element and adds it to the output stream
  - The **select** attribute is required
  - Notice that **xsl:value-of** is *not* a container, hence it needs to end with a slash

Example:

- **<h3> <xsl:value-of select="message"/> </h3>**

XSLT Introduction(Contd..)

<xsl:for-each> Element

- **xsl:for-each** is a one type of loop statement.
- The XSL <xsl:for-each> element can be used to select every XML element of a specified node-set:

syntax:

```
<xsl:for-each   select="XPath expression">
    here add text statement & rules
</xsl:for-each>
```

Example: To select every student **(//student**) and make an unordered list **(<ul>)** of their name **(name),** use:

```
<ul>
    <xsl:for-each select="//student">
      <li>   <xsl:value-of select="name"/>   </li>
    </xsl:for-each> </ul>
```

XSLT Introduction(Contd..)

Apply Condition:

- We can filter (restrict) output by adding a condition to the select attribute’s value:

```
<ul>
   <xsl:for-each select="//student">
     <li>
     <xsl:value-of  select=“name[../age=‘23']"/>
     </li>
     </xsl:for-each>
<ul>
```

- It will select student name with age 23

XSLT Introduction(Contd..)

<xsl:if> Element

- **xsl:if** allows us to include content *if* a given condition (in the **test** attribute) is true

Example:

```
<xsl:for-each  select="//student">
  <xsl:if  test="age='23'">
    <li>
      <xsl:value-of  select="name"/>
    </li>
  </xsl:if>
</xsl:for-each>
```

XSLT Introduction(Contd..)

<xsl:choose> Element

- The xsl:choose ... xsl:when ... xsl:otherwise are like as Java's switch ... case ... default statement.

Syntax:

```
<xsl:choose>
    <xsl:when test="some condition">
        ... Code here ...
    </xsl:when>
    <xsl:otherwise>
        ... Code here ...
    </xsl:otherwise>
</xsl:choose>
```

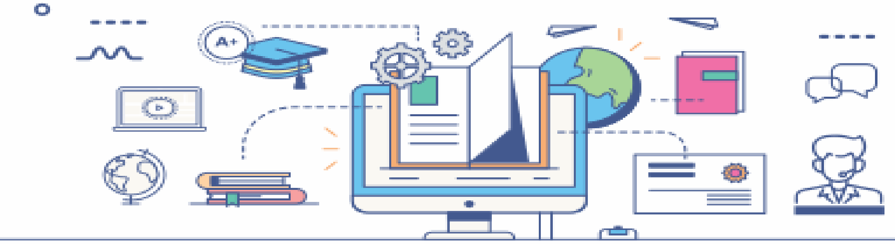
XSLT Introduction(Contd..)

<xsl:sort> Element

- We can place an **xsl:sort** inside an **xsl:for-each**
- The attribute of the sort tells what field to sort on

Example:

```
  <ul>
      <xsl:for-each select="//student">
        <xsl:sort select=“age"/>
        <li>   <xsl:value-of select=“name"/> by
                <xsl:value-of select=“age"/>  </li>
      </xsl:for-each>
    </ul>
```



XSLT Introduction(Contd..)

<xsl:text> Element

- <xsl:text> gives you much better control over whitespace; it acts like the <pre> element in HTML.
- XSL isn’t very careful with whitespace in the document.
- Since XML defines only five entities, you cannot readily put other entities (such as &nbsp;) in your XSL
    - &amp;nbsp; almost works, but &nbsp; is visible on the page
    - Below formula is use for entities:

```
<xsl:text  disable-output-escaping="yes">&amp;nbsp;</xsl:text>
```

Parul® University

DIGITAL LEARNING CONTENT

XSLT Introduction(Contd..)

xsl:apply-templates

- It is applies a template rule to the current element or to the current element’s child nodes.
- If we add a **select** attribute, it applies the template rule only to the child that matches.
- If we have multiple **<xsl:apply-templates>** elements with **select** attributes, the child nodes are processed in the same order as the **<xsl:apply-templates>** elements

XSLT Introduction(Contd..)

Example: File name: parul.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="C:\Users\admin\Desktop\Student.xsl" ?>
<student>
<s>
  <name> Pratik Vaghasiya </name>
  <branch> IT</branch>
  <age>20</age>
  <city> Navsari </city>
 </s>
```

Parul® University

DIGITAL LEARNING CONTENT

XSLT Introduction(Contd..)

```
<s>
<name> Brijal Savaliya </name>
  <branch> CA </branch>
  <age> 20</age>
  <city> Surat </city>
 </s>
 <s>
  <name> Rikita Vaghasiya </name>
  <branch> Micro Bsc</branch>
  <age> 19 </age>
  <city> Navsari </city>
 </s>
<s>
  <name> Vivek Vadher </name>
  <branch> IT </branch>
  <age> 19 </age>
  <city> Rajkot </city>
 </s>
 <s>
  <name> Hardik Vaghasiya </name>
  <branch> Civil </branch>
  <age> 24</age>
  <city> Navsari</city>
 </s>
</student>
```

XSLT Introduction(Contd..)

File Name: Student.xsl

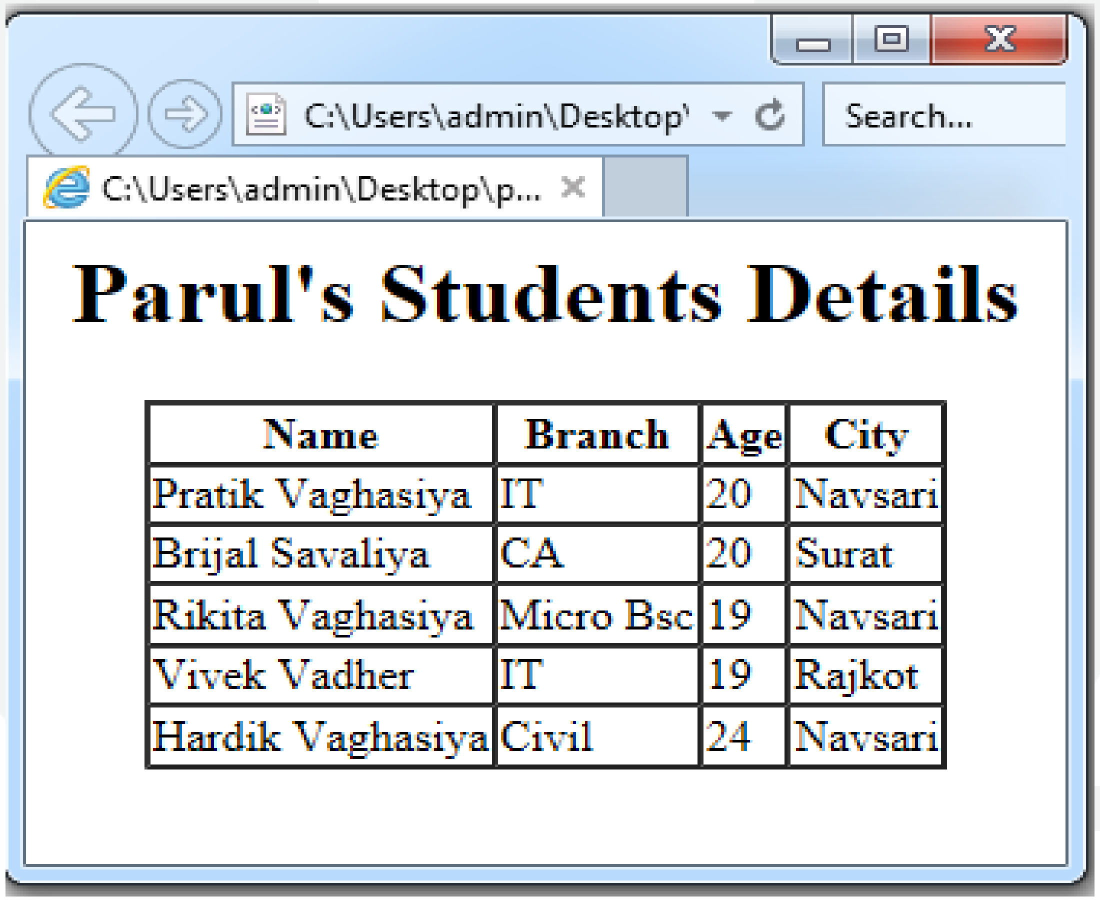
```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
 <html> <body>
  <h1 align="center">Parul's Students Details</h1>
   <table border="1" cellspacing="0" align="center" >
   <tr>
    <th>Name</th>
    <th>Branch</th>
    <th>Age</th>
    <th>City</th>   </tr>
```

Parul® University

DIGITAL LEARNING CONTENT

XSLT Introduction(Contd..)

```
<xsl:for-each select="student/s">
  <tr>
   <td><xsl:value-of select="name"/></td>
   <td><xsl:value-of select="branch"/></td>
   <td><xsl:value-of select="age"/></td>
   <td><xsl:value-of select="city"/></td>
  </tr>
   </xsl:for-each>
   </table>
</body> </html>
</xsl:template>
</xsl:stylesheet>
```



Parul's Students Details

Name	Branch	Age	City
Pratik Vaghasiya	IT	20	Navsari
Brijal Savaliya	CA	20	Surat
Rikita Vaghasiya	Micro Bsc	19	Navsari
Vivek Vadher	IT	19	Rajkot
Hardik Vaghasiya	Civil	24	Navsari

DIGITAL LEARNING CONTENT

Parul® University

www.paruluniversity.ac.in