XML

XML

- XML stands for **EX**tensible **M**arkup **L**anguage.
- XML was designed to **describe data**.
- XML is a software and hardware-independent tool for carrying information.

- **Difference Between XML and HTML**
  - XML was designed to describe data, with focus on what data is
  - HTML was designed to display data, with focus on how data looks

- XML language has no predefined tags.

```
<note>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

- With XML you invent your own tags
- XML is not a replacement for HTML

XML as Mode of Transfer

- XML Simplifies Data Sharing
  - Many databases contains heterogeneous data
  - XML data is stored in plain text format. So this provides s/w and h/w independent way of storing data.
- XML Makes Your Data More Available
- Users define their own tags in XML

Example XML Document

- XML documents form a tree structure that starts at "the root" and branches to "the leaves".

```
<?xml version="1.0" encoding="UTF-8"?>
<note>
  <to>Alan</to>
  <from>Alex</from>
  <heading>Reminder</heading>
  <body>Let's meet this weekend!</body>
</note>
```

- The first line is the XML declaration. It defines the XML version (1.0).
- `<note>` is the root element
- `<to>,<from>,<heading>,<body>` are child elements

- XML Documents Form **aTree Structure**
- 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.
- All elements can have sub elements (child elements):

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

XML Syntax Rules

- All XML Elements Must Have a Closing Tag
- XML Tags are Case Sensitive

```
<Message>This is incorrect</message>
<message>This is correct</message>
```

- XML Elements Must be Properly Nested

```
<b><i>This text is bold and italic</i></b>
```

- XML Attribute Values Must be Quoted

```
<note date="12/11/2007">
  <to>Tove</to>
  <from>Jani</from>
</note>
```

XML Syntax Rules(2)

Entity	Symbol	Description
&lt;	<	less than
&gt;	>	greater than
&amp;	&	ampersand
&apos;	'	apostrophe
&quot;	"	quotation mark

<message>if salary < 1000 then</message>

<message>if salary < 1000 then</message>

Comments in XML

<!--This is a comment -->

White-space is Preserved in XML

XML Elements

- An XML document contains XML Elements.
- An element can contain:
  - Other elements
  - Text
  - Attributes or
  - Mix of all of the above…

```
<bookstore>
  <book category="CHILDREN">
    <title>Harry Potter</title>
    <author>J K. Rowling</author>
    <year>2005</year>
  </book>
  <book category="WEB">
    <title>Learning XML</title>
    <author>Erik T. Ray</author>
    <year>2003</year>
  </book>
</bookstore>
```

XML Naming Rules

- XML elements must follow these naming rules:
  - Element names are **case-sensitive**
  - Element names must start with a **letter** or **underscore**
  - Element names **cannot start** be as **xml** (or XML, or Xml, etc)
  - Element names **can contain letters, digits, hyphens, underscores, and periods**
  - Element names **cannot contain spaces**
- NOTE: No words are reserved (except xml).

Naming Styles

There are no naming styles defined for XML elements. But here are some commonly used:

Style	Example	Description
Lower case	<firstname>	All letters lower case
Upper case	<FIRSTNAME>	All letters upper case
Underscore	<first_name>	Underscore separates words
Pascal case	<FirstName>	Uppercase first letter in each word
Camel case	<firstName>	Uppercase first letter in each word except the first

XML Elements are Extensible

- XML elements can be extended to carry more information.

```
<note>
  <to>Tove</to>
  <from>Jani</from>
  <body>Don't forget me this weekend!</body>
</note>
```

- If application is reading <to> <from> and <body> elements.

```
<note>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

- The application will still read the added element <heading> with out any crash..

XML Attributes

- XML elements can have attributes, and it provides extra information about an element but not the data.

```
<file type="gif">computer.gif</file>
```

- XML attribute values must be quoted

```
<person gender="female">
```

```
<person gender='female'>
```

- If the attribute value itself contains double quotes you can use single quotes, like in this example:

```
<gangster name='George "Shotgun" Ziegler'>
```

```
<gangster name="George &quot;Shotgun&quot; Ziegler">
```

XML Elements and Attributes

- Consider following examples

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

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

- Both the codes gives the same information. There are no rules in writing attributes.

XML Elements and Attributes

```
<note date="2008-01-10">
    <to>Tove</to>
    <from>Jani</from>
    <heading>Reminder</heading>
    <body>Don't forget</body>
</note>
```

```
<note>
  <date>
      <year>2008</year>
      <month>01</month>
      <day>10</day>
  </date>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
   <body>Don't forget</body>
  </note>
```

```
<note>
    <date>2008-01-10</date>
    <to>Tove</to>
    <from>Jani</from>
    <heading>Reminder</heading>
    <body>Don't forget</body>
  </note>
```

XML Attributes for Metadata

Sometimes ID references are assigned to elements. These IDs can be used to identify XML elements

```
<messages>
    <note id="501">
      <to>Tove</to>
      <from>Jani</from>
      <heading>Reminder</heading>
      <body>Don't forget</body>
    </note>
    <note id="502">
      <to>Jani</to>
      <from>Tove</from>
      <heading>Re: Reminder</heading>
      <body>I will not</body>
    </note>
</messages>
```

The id attributes above are for identifying the different <note> elements

XML Namespaces

- XML Namespaces provide a method to avoid element name conflicts.
- Name Conflicts
  - Since element names are given by authors, it is difficult when XML files from diff sources are mixed if element names are matched.
- The following XML's contains table information (HTML <table> element and table information respectively)

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

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

XML Namespaces (2)

- If the above 2 XML's are merged to single XML, there would be name (table) conflict.
- **Solving the Name Conflict Using a Prefix**
- Name conflicts in XML can easily be avoided **using a name prefix.**

```
<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>
```

- Now two elements names <table> has different names….**h:table** and **f:table**

XML Namespaces (2)

- **XML Namespaces - The xmlns Attribute**
  - When using prefixes in XML, a so-called **namespace** for the attribute 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/html5/">
    <h:tr>
      <h:td>Apples</h:td>
      <h:td>Bananas</h:td>
    </h:tr>
  </h:table>

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

XML Namespaces (2)

- **Default Namespaces**
  - Default Namespaces doesn't require any prefixes to the elements.

```
xmlns="namespaceURI"
```

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

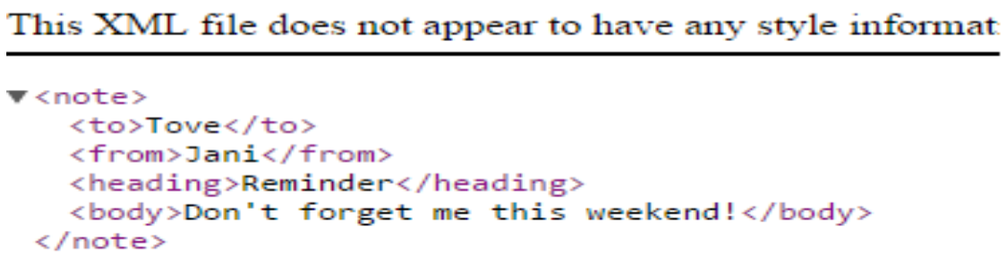
```
<table xmlns="http://www.pepperfry.com/furniture">
    <name>African Coffee Table</name>
    <width>80</width>
    <length>120</length>
  </table>
```

```
<root xmlns:h="http://www.w3.org/TR/html4/"
  xmlns:f="http://www.pepperfry.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>
```

Viewing XML Files

- Raw XML files can be viewed in all major browsers.

```
<?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>
```



- Since XML tags are "invented" by the author of the XML document, browsers do not know if a tag like <table> describes an HTML table or a dining table.

Displaying XML with CSS

- CSS (Cascading Style Sheets) can add display information to an XML document.

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/css" href="cd_catalog.css"?>
<CATALOG>
     <CD>
       <TITLE>Empire Burlesque</TITLE>
       <ARTIST>Bob Dylan</ARTIST>
       <COUNTRY>USA</COUNTRY>
       <COMPANY>Columbia</COMPANY>
       <PRICE>10.90</PRICE>
       <YEAR>1985</YEAR>
</CATALOG>
```

```
CATALOG
{ background-color:#ffffff;width: 100%;}
CD { display: block; margin-bottom: 30pt;
    margin-left: 0;}
TITLE { color:#FF0000; font-size: 20pt;}
ARTIST { color:#0000FF; font-size: 20pt;
    }
COUNTRY,PRICE,YEAR,COMPANY  {
    display: block; color:#000000;
    margin-left: 20pt;}
```

Empire Burlesque Bob Dylan
USA
Columbia
10.90
1985

XML Validation

- An XML document with **correct syntax** is called "**Well Formed**".
- WellFormed XML are documents with correct 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**
- Valid XML must be **wellFormed + well document type definition**

Document Type Definitions (DTD)

- DTD are **rules** that defines **elements** and **attributes of the elements** for XML
- Two different document type definitions that can be used with XML:
  - DTD - The original Document Type Definition
  - XML Schema - An XML-based alternative to DTD

XML DTD

- A "Valid" XML document is a "Well Formed" XML document, which also conforms to the rules of a DTD
- Consider following code

```
<?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 this weekend!</body>
  </note>
```

- **Note.dtd** is the external reference for the DTD

XML DTD (2)

- DTD is to define the structure of an XML document.

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

  - !DOCTYPE **note** defines that the root element of the 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"

#PCDATA means parse-able text data.

XML DTD (3)

- **Internal DTD Declaration**
  - DTD is declared inside the XML file, it must be wrapped **inside the <!DOCTYPE> definition**.

- **External DTD Declaration**

```
<?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>
```

XML DTD (4)

- **DTD - XML Building Blocks**
- Most of the XML documents are made up by the following building blocks
  - Eléments &Attributes
  - Entities
  - PCDATA & CDATA
- **Elements**
  - Elements are the main building blocks of both XML and HTML documents.

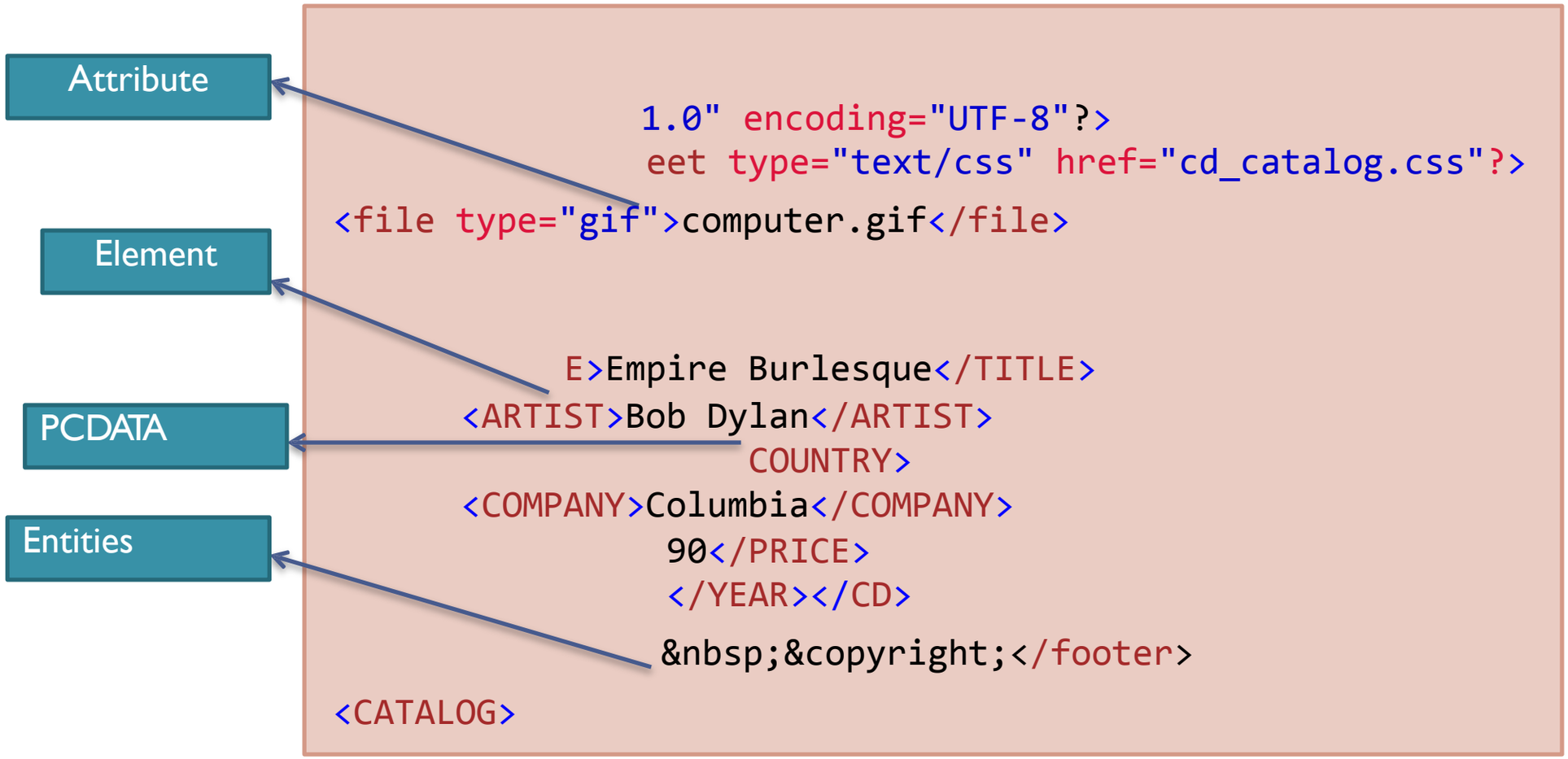
XML DTD (5)

- **Attributes**
  - Attributes provide extra information about elements.
  - Attributes are always placed inside the opening tag of an element
- **Entities**
  - Some characters have a special meaning in XML, like the less than sign (<) that defines the start of an XML tag.

Entity References	Character
&lt;	<
&gt;	>
&amp;	&
&quot;	"
&apos;	'

XML DTD (6)

- **PCDATA**
  - PCDATA means parsed character data.
  - PCDATA is text **thatWILL be parsed by a parser**. The text will be examined by the parser for entities and markup (tags).
  - It is found between start and end tags.
  - However, parsed character data **should not contain any &, <, or > characters**; these need to be represented by the **& < and > entities**, respectively.
- **CDATA**
  - CDATA means character data.
  - Not parsed by the parsers



An entity has three parts: an ampersand (&), an entity name, and a semicolon (;).

DTD - Elements

- **Declaring Elements**
  - Elements are declared with an ELEMENT declaration
    - **Syntax**

```
<!ELEMENT element-name category>
<!ELEMENT element-name (element-content)>
```

  - Elements with Parsed Character Data

```
<!ELEMENT element-name (#PCDATA)>
```

  - Elements with Children

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

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

DTD - Elements (2)

When children are declared in doctypes, they should also mention in **sequence**.
For example…

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

- Declaring Only One Occurrence of an Element

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

- Declaring either/or Content

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

- Declaring Mixed Content

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

* Match 0 or more times
+ Match 1 or more times
? Match 1 or 0 times

DTD -Attributes

- In a DTD, attributes are declared with an ATTLIST declaration
- **Declaring Attributes**
- **Syntax**

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

- **Example**

```
DTD example:
<!ATTLIST payment type CDATA "cheque">
XML example:
<payment type="cheque" />
```

DTD - Attributes values

Value	Explanation
value	The default value of the attribute
#REQUIRED	The attribute is required
#IMPLIED	The attribute is optional
#FIXED *value*	The attribute value is fixed

A Default Attribute Value

DTD

```
<!ELEMENT square EMPTY>
<!ATTLIST square width CDATA “100">
```

XML

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

DTD - Attributes values (2)

- **#REQUIRED**
  - **Syntax**

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

  - DTD

```
<!ATTLIST person number CDATA #REQUIRED>
```

  - Valid XML

```
<person number="5677" />
```

  - Invalid XML

```
<person />
```

- #REQUIRED keyword is used if the attribute value is forcibly present.

DTD - Attributes values (3)

- **#IMPLIED**
- **Syntax**

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

  - DTD:

```
<!ATTLIST contact fax CDATA #IMPLIED>
```

  - Valid XML

```
<contact fax="555-667788" />
```

```
<contact />
```

DTD - Attributes values (4)

- **#FIXED**
- **Syntax**

```
<!ATTLIST element-name attribute-name attribute-type #FIXED "value">
```

  - DTD

```
<!ATTLIST sender company CDATA #FIXED “CDAC">
```

  - Valid XML

```
<sender company=“CDAC" />
```

  - Invalid XML

```
<sender company=“ACTS" />
```

DTD -Attributes values (5)

- **Enumerated AttributeValues**
- **Syntax**

```
<!ATTLIST element-name attribute-name (en1|en2|..) default-value>
```

  - DTD

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

  - Valid XML

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

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

DTD - Entities

- Entities are used to define shortcuts to special characters.
- Entities can be declared internal or external.
- **Syntax**

```
<!ENTITY entity-name "entity-value">
```

- Example
  - DTD

```
<!ENTITY writer "Donald Duck.">
<!ENTITY copyright "Copyright CDAC.">
```

  - XML

```
<author>&writer;&copyright;</author>
```

DTD - Entities(2)

- External Entity
- **Syntax**

```
<!ENTITY entity-name SYSTEM "URI/URL">
```

- Example
  - DTD

```
<!ENTITY writer SYSTEM "http://www.xyz.com/entities.dtd">
<!ENTITY copyright SYSTEM "http://www.xyz.com/entities.dtd">
```

  - XML

```
<author>&writer;&copyright;</author>
```

XML Schema

- An XML Schema describes the structure of an XML document, just like a DTD.
- XML Schema is an XML-based alternative to DTD

```
<note>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me!</body>
</note>
```

```
<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>
```

XML Schema (2)

- Schema is interpreted as
  - `<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
  - `<xs:element name="heading" type="xs:string">` the element "heading" is of type string
  - `<xs:element name="body" type="xs:string">` the element "body" is of type string

XML Schema (3)

- **XML Schema defines following elements in XML document**
  - 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**

XML Schema : Data Communication

- XML Schemas Secure Data Communication
- **For example**
  - A date like: "03-11-2004" will, in some countries, be interpreted as 3.November and in other countries as 11.March.
  - However, an XML element with a data type like this:

    <date type="date">2004-03-11</date>
  - Ensures a mutual understanding of the content, because the XML data type "date" requires the format "YYYY-MM-DD".

XML & DTD &XML Schema example

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

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

```
<?xml version="1.0"?>
<xs:schema>
<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 (4)

- XML Schema language is also referred to as **XML Schema Definition (XSD).**
- **Reference to a DTD**

```
<?xml version="1.0"?>
<!DOCTYPE note SYSTEM note.dtd>
<note>
  <to>Tove</to>……
```

- **Reference to a XML Schema**

```
<note
xsi:schemaLocation="note.xsd">
……
</note>
```

XML Schema (5)

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

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

- Schema Declaration

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace=http://www.xyz.com
elementFormDefault="qualified">
...
...
</xs:schema>
```

Schema Declaration

- It indicates elements and data types used in the schema come from the www.w3.org

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

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

```
targetNamespace="http://www.xyz.com"
```

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

```
elementFormDefault="qualified"
```

XML Schema Elements

- **Simple element**
  - XML element that contains only text. It cannot contain any other elements or attributes.
- **Defining a Simple Element**

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

  - xxx is the name of the element and yyy is the 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

XML Schema Elements (2)

- XML elements and its corresponding simple element definitions

```
<lastname>Refsnes</lastname>
<age>36</age>
<dateborn>1970-03-27</dateborn>
```

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

- Default and FixedValues for Simple Elements
  - Simple elements may have a default value OR a fixed value specified.

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

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

XML Schema Attributes

- **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.
- **Defining the attribute**

```
<xs:attribute name="x1" type="y1"/>
```

`x1` is the name of the attribute and `y1` is the type of the attribute

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

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

XML Schema Attributes (2)

- XML attributes and its corresponding simple element definitions

```
<lastname language="EN">Smith</lastname>
```

- Simple element attribute definition

```
<xs:attribute name="language" type="xs:string"/>
```

- **Default and Fixed Values for Attributes**
  - Attributes may have a default value OR a fixed value specified.

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

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

- **Optional and Required Attributes**
  - Attributes are optional by default. To specify that the attribute is required, use the "use" attribute:

```
<xs:attribute name="language" type="xs:string" use="required"/>
```

XML Facets

- When an XML element or attribute has a data type defined, it puts **restrictions** on the element's or attribute's content.
  - If an XML element is of type "**xs:date**" and contains a string like "HelloWorld", the element will not validate.
- With XML Schemas, you can also add your own restrictions to your XML elements and attributes.
- These restrictions are called **facets**

XSD Restrictions/Facets

- Restriction on value

```
<xs:element name="age">
    <xs:simpleType>
      <xs:restriction base="xs:integer">
        <xs:minInclusive value="0"/>
        <xs:maxInclusive value="120"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:element>
```

- Restriction on set of values

```
<xs:element name="car">
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:enumeration value="Audi"/>
        <xs:enumeration value="Golf"/>
        <xs:enumeration value="BMW"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:element>
```

XSD Restrictions/Facets

```
<xs:restriction base="xs:string">
   <xs:pattern value="[a-z]"/>
</xs:restriction>
```

```
<xs:restriction base="xs:string">
        <xs:pattern value="[a-zA-Z][a-zA-Z][a-zA-Z]"/>
</xs:restriction>
```

```
<xs:restriction base="xs:string">
   <xs:pattern value="[xyz]"/>
   </xs:restriction>
```

```
<xs:restriction base="xs:integer">
   <xs:pattern value="[0-9][0-9][0-9][0-9][0-9]"/>
   </xs:restriction>
   </xs:simpleType>
```

```
<xs:restriction base="xs:string">
   <xs:pattern value="([a-z])*"/>
   </xs:restriction>
```

```
<xs:restriction base="xs:string">
        <xs:pattern value="male|female"/>
</xs:restriction>
```

```
<xs:restriction base="xs:string">
   <xs:pattern value="([a-z][A-Z])+"/>
   </xs:restriction>
```

```
<xs:restriction base="xs:string">
        <xs:pattern value="[a-zA-Z0-9]{8}"/>
      </xs:restriction>
```

XSD Complex Elements

- A complex element contains other elements and/or attributes.
- **Four kinds of complex elements**
  - empty elements with attribute
  - elements that contain only other elements
  - elements that contain only text with one attribute
  - elements that contain both other elements and text
- Examples of Complex Elements
  - empty elements

```
<product pid="1345"/>
```

  - elements that contain only other elements

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

XSD Complex Elements (2)

- elements that contain only text with one attribute

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

- elements that contain both other elements and text

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

XSD Complex Elements (3)

- **Defining complex element**
  - For the following XML

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

  - The "employee" element can be declared directly by naming the element, like this:

```
<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>
```

XSD Empty Elements

- An empty complex element cannot have contents, only attributes.

```
<product prodid="1345" />
```

- XSD would be

```
<xs:element name="product">
    <xs:complexType>
      <xs:attribute name="prodid" type="xs:positiveInteger"/>
    </xs:complexType>
  </xs:element>
```

XSD Text-Only Elements

- A complex text-only element can contain text and attributes.

```
<shoesize country="france">35</shoesize>
```

  - The following example declares a complexType, "shoesize". The content is defined as an integer value, and the "shoesize" element also contains an attribute named "country":

```
<xs:element name="shoesize">
    <xs:complexType>
      <xs:simpleContent>
        <xs:extension base="xs:integer">
          <xs:attribute name="country" type="xs:string" />
        </xs:extension>
      </xs:simpleContent>
    </xs:complexType>
  </xs:element>
```

XSD Mixed Content

- A mixed complex type element can contain attributes, elements, and text

```
<letter>
    Dear Mr.<name>John Smith</name>.
    Your order <orderid>1032</orderid>
    will be shipped on <shipdate>2001-07-13</shipdate>.
  </letter>
```

```
<xs:element name="letter">
    <xs:complexType mixed="true">
      <xs:sequence>
        <xs:element name="name" type="xs:string"/>
        <xs:element name="orderid" type="xs:positiveInteger"/>
        <xs:element name="shipdate" type="xs:date"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
```

- <xs:complexType mixed="true"> which enables character data to appear between the child-elements of "letter"

XSD String Data Types

- String data types are used for values that contains character strings.
- string data type can contain characters, line feeds, carriage returns, and tab characters.

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

- **NormalizedString DataType**
  - normalizedString data type also contains characters, but the XML processor will remove line feeds, carriage returns, and tab characters.

```
<xs:element name="customer" type="xs:normalizedString"/>
```

XSD Indicators

- We can control HOW elements are to be used in documents with indicators.
- Seven Indicators
  - **Order indicators:**
    - All, Choice, Sequence
  - **Occurrence indicators:**
    - maxOccurs & minOccurs
  - **Group indicators:**
    - Group & attributeGroup

XSD Indicators

- **Order Indicators :All Indicator**
  - The <all> indicator specifies that the child elements can appear in **any order**, and that each child element must occur only once

```
<xs:element name="person">
    <xs:complexType>
      <xs:all>
        <xs:element name="firstname" type="xs:string"/>
        <xs:element name="lastname" type="xs:string"/>
      </xs:all>
    </xs:complexType>
  </xs:element>
```

XSD Indicators

- **Order Indicators : Choice Indicator**
- The <choice> indicator specifies that either one child element or another can occur:

```
<xs:element name="person">
    <xs:complexType>
      <xs:choice>
        <xs:element name="employee" type="employee"/>
        <xs:element name="member" type="member"/>
      </xs:choice>
    </xs:complexType>
  </xs:element>
```

XSD Indicators

- **Order Indicators : Sequence Indicator**
- The <sequence> indicator specifies that the child elements must appear in a specific order:

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

XSD Indicators

- **Occurrence Indicators: maxOccurs Indicator**
- The <maxOccurs> indicator specifies the maximum number of times an element can occur:

```
<xs:element name="person">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="full_name" type="xs:string"/>
        <xs:element name="child_name" type="xs:string" maxOccurs="10"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
```

XSD Indicators

- **Occurrence Indicators: minOccurs Indicator**
- The <minOccurs> indicator specifies the minimum number of times an element can occur:

```
<xs:element name="person">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="full_name" type="xs:string"/>
        <xs:element name="child_name" type="xs:string"
        maxOccurs="10" minOccurs="0"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
```

maxOccurs="unbounded“ for unlimited occurrences

XSD any elements

- The <any> element enables us to extend the XML document with elements not specified by the schema!

```
<xs:element name="person">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="firstname" type="xs:string"/>
        <xs:element name="lastname" type="xs:string"/>
        <xs:any minOccurs="0"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
```

DTDVs XSD

The important differences are given below:

No.	DTD	XSD
1)	DTD stands for **Document Type Definition**.	XSD stands for XML Schema Definition.
2)	DTDs are derived from **SGML** syntax.	XSDs are written in XML.
3)	DTD **doesn't support datatypes**.	XSD **supports datatypes** for elements and attributes.
4)	DTD **doesn't support namespace**.	XSD **supports namespace**.
5)	DTD **doesn't define order** for child elements.	XSD **defines order** for child elements.
6)	DTD is **not extensible**.	XSD is **extensible**.
7)	DTD is **not simple to learn.**.	XSD is **simple to learn** because you don't need to learn new language..
8)	DTD provides **less control** on XML structure.	XSD provides **more control** on XML structure.

XML- Javascript

- **The XMLHttpRequest Object**
  - The XMLHttpRequest object is used to exchange data with a server behind the scenes.
- XMLHttpRequest can
  - Update a web page without reloading the page
  - Request data from a server after the page has loaded
  - Receive data from a server after the page has loaded
  - Send data to a server in the background
- Before more on XMLHttpRequest object need to know about XML DOM

XML DOM

- The XML DOM defines a standard for **accessing** and **manipulating** XML documents.
- The DOM presents an XML document as a **tree-structure**
- *"TheW3C Document Object Model (DOM) is a platform and language-neutral interface that allows programs and scripts to dynamically access and update the content, structure, and style of a document."*
- The DOM is separated into 3 different parts / levels:
  - Core DOM - standard model for any structured document
  - XML DOM - standard model for XML documents
  - HTML DOM - standard model for HTML documents

HTML DOM

- The HTML DOM defines the **objects and properties** of all HTML elements, and the **methods** (interface) to access them
- There are many Properties and Methods

Demo codes are given under HTMLDOM directory

XML DOM(2)

- The XML DOM is
  - A standard object model for XML
  - A standard programming interface for XML
  - Platform and language independent
- **The XML DOM is a standard for how to get, change, add, or delete XML elements.**

XML DOM(3)

- **XML DOM Nodes**
  - Everything in XML document is a node

- **DOM Nodes:**
  - The entire document is a document node
  - Every XML element is an element node
  - The text in the XML elements are text nodes
  - Every attribute is an attribute node
  - Comments are comment nodes

DOM Example

```
<?xml version="1.0" encoding="UTF-8"?>
<bookstore>
  <book category="
    <title lang="                    an</title>
    <aut                             thor>
    <year
    <price>30.00</price>
  </book>
  <book category="children">
    <title lang="en">Harry Potter</title>
    <author                          >
    <year>
    2price>29.99</price>
  </book>
  <book category="web" cover="paperback">
    <title lang="en">Learning XML</title>
    <author>                         r>
    <year>2
    <price>39.95</price>
  </book>
</bookstore>
```

Root node

First node

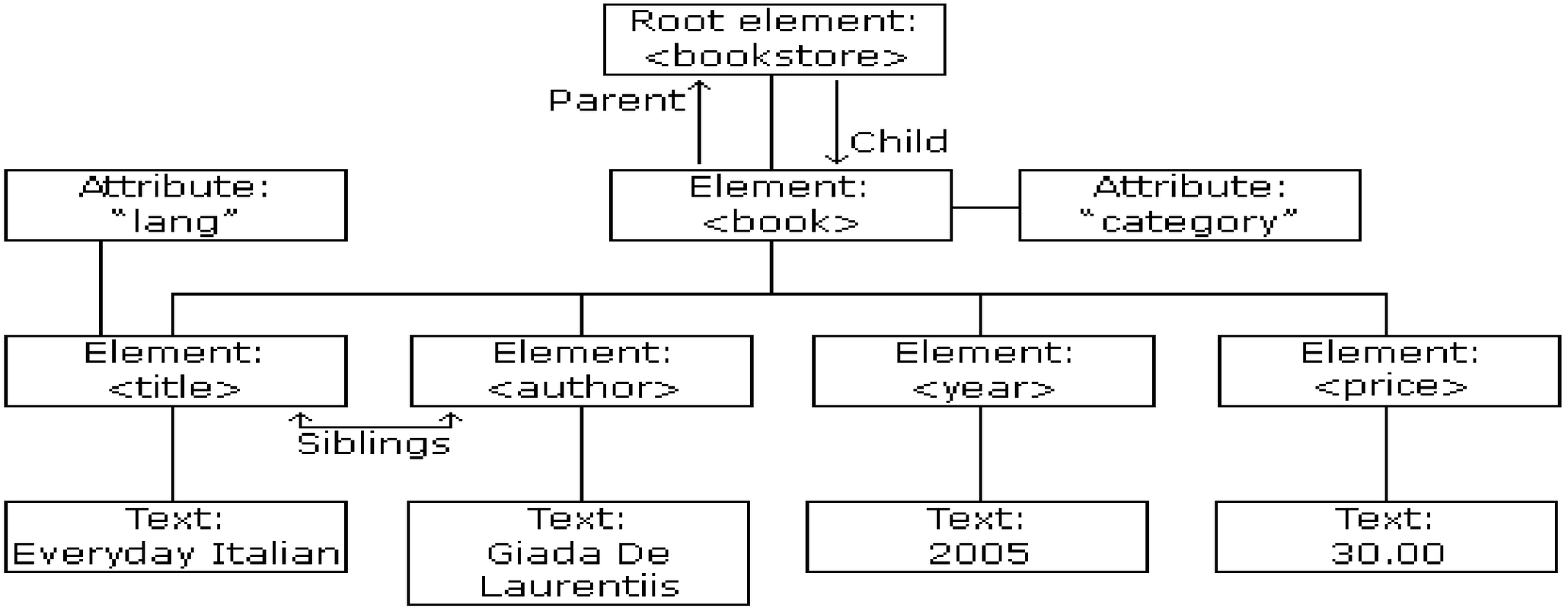
Second node

Third node

XML DOM(4)

- The XML DOM views an XML document as a node-tree.
  - In a node tree, the top node is called the root
  - Every node, except the root, has exactly one parent node
  - A parent node can have any number of children
  - A leaf is a node with no children
  - Siblings are nodes with the same parent

XML DOM Tree Example



XML DOM Parser

- All major browsers have a built-in XML parser to read and manipulate XML
- **The XML parser converts XML into an XML DOM object that can be accessed with JavaScript**.
- **XML Parser**
  - The XML DOM contains methods to traverse XML trees, access, insert, and delete nodes.
  - XML Document should be loaded into XML DOM Object

XML Parser(2)

- An XML parser reads XML, and converts it into an XML DOM object that can be accessed with JavaScript.

```
xmlhttp=new xmlhttprequest();
xmlhttp.Open("get","books.xml");
xmlhttp.Send();
xmldoc=xmlhttp.Responsexml;
```

open(*method, url, async*) GET is simpler and faster than POST, and can be used in most cases.

  - Create an XMLHttpRequest object
  - Use the open() and send() (used only for get method) methods of the XMLHttpRequest object to send a request to a server
  - Get the response data as XML data

XML DOM Load Functions

```
function loadXMLDoc(filename)
{
if (window.XMLHttpRequest)
  {
  xhttp=new XMLHttpRequest();
  }
else // code for IE5 and IE6
  {
  xhttp=new ActiveXObject("Microsoft.XMLHTTP");
  }
xhttp.open("GET",filename, false);
xhttp.send();
return xhttp.responseXML;
}
```

The function above can be stored in the <head> section of an HTML page, and called from a script in the page.

XML DOM Load Functions(2)

- **An External JavaScript for loadXMLDoc()**

  If we want to use the same code in all xml pages,we can load as external java script

```
<html>
<head>
<script src="loadxmldoc.js">
</script>
</head>
<body>
<script>
var xmlDoc=loadXMLDoc("books.xml");
code goes here.....
</script>
</body>
</html>
```

XML DOM - Properties and Methods

- **XML DOM Properties**
  - x.nodeName - the name of x
  - x.nodeValue - the value of x
  - x.parentNode - the parent node of x
  - x.childNodes - the child nodes of x
  - x.attributes - the attributes nodes of x
- **XML DOM Methods**
  - x.getElementsByTagName(*name*) - get all elements with a specified tag name
  - x.appendChild(*node*) - insert a child node to x
  - x.removeChild(*node*) - remove a child node from x
- **With the DOM, you can access every node in an XML document.**

XML DOM - Properties and Methods(2)

- The nodeName, nodeValue, and nodeType properties contain information about nodes.

- **Accessing Nodes**
  - A list of nodes is returned by the getElementsByTagName() method
  - By looping through (traversing) the nodes tree
  - By navigating the node tree, using the node relationships.

XML DOMTraverse NodeTree

- Looping the XML document is called **Traversing the node tree**
- Traversing can extract value of each element.

Manipulate Nodes

- **XML DOM Get NodeValues**
  - The **nodeValue** property is used to get the text value of a node.
  - The **getAttribute()** method returns the **value** of an attribute.
- **Get an ElementValue**
  - The getElementsByTagName() method returns a node list containing all elements.

```
xmlDoc=loadXMLDoc("books.xml");
x=xmlDoc.getElementsByTagName("title")[0];
```

Manipulate Nodes(2)

- **Change the Value of an Attribute**
  - setAttribute() method
  - The setAttribute() method changes the value of an existing attribute, or creates a new attribute.
- **XML DOM Remove Nodes**
  - The removeChild() method removes a specified node.
  - The removeAttribute() method removes a specified attribute.
- **Remove an Attribute Node by Name**
  - The removeAttribute(*name*) method is used to remove an attribute node by its name.

Manipulate Nodes(3)

- **Create a New Element Node**
  - The createElement() method creates a new element node:

```
xmlDoc=loadXMLDoc("books.xml");
newel=xmlDoc.createElement("edition");
x=xmlDoc.getElementsByTagName("book")[0];
x.appendChild(newel);
```

- **Create a New Attribute Node**

```
xmlDoc=loadXMLDoc("books.xml");

newatt=xmlDoc.createAttribute("edition");
newatt.nodeValue="first";

x=xmlDoc.getElementsByTagName("title");
x[0].setAttributeNode(newatt);
```

Manipulate Nodes(4)

- **Create an Attribute Using setAttribute()**

```
xmlDoc=loadXMLDoc("books.xml");
x=xmlDoc.getElementsByTagName('book');
x[0].setAttribute("edition","first");
```

  - `createAttribute` : If the attribute already exists, it is replaced by the new one.

- **Create a Text Node**
  - The createTextNode() method creates a **new text node**

```
xmlDoc=loadXMLDoc("books.xml");

newel=xmlDoc.createElement("edition");
newtext=xmlDoc.createTextNode("first");
newel.appendChild(newtext);

x=xmlDoc.getElementsByTagName("book")[0];
x.appendChild(newel);
```

Manipulate Nodes(5)

- **Create a CDATA Section Node**
  - The createCDATASection() method creates a new CDATA section node
- **Add a Node**
  - The appendChild() method adds a child node to an existing node.
  - The new node is added (appended) after any existing child nodes.
  - Use insertBefore() if the position of the node is important.

```
xmlDoc=loadXMLDoc("books.xml");

newel=xmlDoc.createElement("edition");

x=xmlDoc.getElementsByTagName("book")[0];
x.appendChild(newel);
```

Manipulate Nodes(6)

- **Add a New Attribute**
  - The setAttribute() method creates a new attribute if the attribute does not exist
- **Add Text to a Text Node**
  - The insertData() method inserts data into an **existing text node**.
  - The insertData() method has two parameters
    - offset - Where to begin inserting characters (starts at zero)
    - string - The string to insert

Manipulate Nodes(7)

- **AddText to aText Node**
  - The following code fragment will add "Easy" to the text node of the first <title> element of the loaded XML

```
xmlDoc=loadXMLDoc("books.xml");
x=xmlDoc.getElementsByTagName("title")[0].childNodes[0];
x.insertData(0,"Easy ");
```

- **Copy a Node**
  - The cloneNode() method creates a copy of a specified node.

XML Technology-XPath

- XPath is used to navigate through elements and attributes in an XML document.
- In XPath, there are following nodes:
  - element, attribute, text, namespace, path instructions, comment, and document nodes.

```
<?xml version="1.0" encoding="UTF-8"?>
<bookstore>
  <book>
    <title lang="en">Harry Potter</title>
    <author>J K. Rowling</author>
    <year>2005</year>
    <price>29.99</price>
  </book>
</bookstore>
```

```
<bookstore> (root element node)

<author>J K. Rowling</author> (element node)

lang="en" (attribute node)
```

XML Technology-Xpath(2)

- What are following
  - Parent
  - Children
  - Siblings
  - Ancestors
  - Descendants

```
<bookstore>

<book>
  <title>Harry Potter</title>
  <author>J K. Rowling</author>
  <year>2005</year>
  <price>29.99</price>
</book>

</bookstore>
```

XML Technology-Xpath(3)

- **XPath Syntax**
  - XPath uses **path expressions** to select nodes or node-sets in an XML document

Expression	Description
nodename	Selects all nodes with the name "*nodename*"
/	Selects from the root node
//	Selects nodes in the document from the current node that match the selection, no matter where they are
.	Selects the current node
..	Selects the parent of the current node
@	Selects attributes

XML Technology-Xpath(4)

```
<bookstore>
<book>
  <title>Harry Potter</title>
  <author>J K. Rowling</author>
  <year>2005</year>
  <price>29.99</price>
</book>
</bookstore>
```

Path Expression	Result
book	Selects all nodes with the name "book"
/book	Selects the root element bookstore **Note:** If the path starts with a slash (/) it always represents an absolute path to an element!
bookstore/book	Selects all book elements that are children of bookstore
//book	Selects all book elements no matter where they are in the document
bookstore//book	Selects all book elements that are descendant of the bookstore element, no matter where they are under the **bookstore** element
//@lang	Selects all attributes that are named lang

XML Technology-Xpath(5)

Predicates: used to find a specific node or a node that contains a specific value

Path Expression	Result
/bookstore/book[1]	Selects the first book element that is the child of the bookstore element.
/bookstore/book[last()]	Selects the last book element that is the child of the bookstore element
/bookstore/book[last()-1]	Selects the last but one book element that is the child of the bookstore element
/bookstore/book[position()<3]	Selects the first two book elements that are children of the bookstore element
//title[@language]	Selects all the title elements that have an attribute named language
//title[@lang='en']	Selects all the title elements that have a "lang" attribute with a value of "en"
/bookstore/book[price>35.00]	Selects all the book elements of the bookstore element that have a price element with a value greater than 35.00
/bookstore/book[price>35.00]/title	Selects all the title elements of the book elements of the bookstore element that have a price element with a value greater than 35.00

XML Technology-Xpath(6)

- **Selecting Nodes**

```
xmlDoc.selectNodes(xpath);
```

- **Select all the titles**

```
/bookstore/book/title
/bookstore/book[1]/title <!–Selects title of first book-->
```

- **Select all the prices**

```
/bookstore/book/price
/bookstore/book[price>35]/price
```

XML Technology-XSLT

- XSL stands for **EX**tensible **S**tylesheet **L**anguage
- XSLT stands for XSL Transformations
- XSLT is used to transform XML documents into other formats, like HTML.
- CSS = Style Sheets for HTML *like wise*
- XSLT = recommended Style Sheets for XML
- XSLT Uses XPath

XSLT Basics

- Prerequisite for XSLT is HTML and XML
- **How it works**
  - In the transformation process, XSLT uses **XPath** to define parts of the source document that should **match one or more predefined templates**.
  - When a match is found, XSLT will transform the matching part of the source document into the result document.

How to convert?

- Correct style sheet declaration

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

  - The xmlns:xsl="http://www.w3.org/1999/XSL/Transform" points to the officialW3C XSLT namespace.
  - If you use this namespace,you must also include the attribute version="1.0".

XSLT - Example

- **Example :** Consider the catalog.xml file as shown

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

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

XSLT <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 templates.
- The **match** attribute is used to associate a template with an XML element.
- The value of the match attribute is an XPath expression (/ expression). That means we are selecting whole document first.

XSLT <xsl:template> Element Example

- Example of xsl for building template

```
<?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>
    <tr>
      <td>.</td>
      <td>.</td>
    </tr>
  </table>
  </body>
  </html>
</xsl:template>
</xsl:stylesheet>
```

The **<xsl:template>** element defines a template.

The **match="/"** attribute associates the template with the root of the XML source document.

The last two lines define the end of the template and the end of the style sheet.

XSLT <xsl:value-of> Element

- The **<xsl:value-of>** element is used to extract the value of a selected node.
- Example of xsl selecting the values is shown

My CD Collection

Title	Artist
Empire Burlesque	Bob Dylan

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

</xsl:stylesheet>
```

XSLT <xsl:for-each> Element

- The <xsl:for-each> element allows you to do looping in XSLT.
- Output is as follows

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

XSLT : Filtering the Output

- Example Filtering the output

```
<xsl:for-each select="catalog/cd[artist='Gary Moore']">
    <tr>
      <td><xsl:value-of select="title"/></td>
      <td><xsl:value-of select="artist"/></td>
    </tr>
</xsl:for-each>
```

My CD Collection

Title	Artist
Still got the blues	Gary Moore

- Legal filter operators are:
  - = (equal)
  - != (not equal)
  - <less than
  - >greater than

XSLT : Sort

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

```
<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>
```

XSLT <xsl:if> Element

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

```
<xsl:if test="expression">
  ...some output if the expression is true...
</xsl:if>
```

- **Example**

```
<xsl:for-each select="catalog/cd">
      <xsl:if test="price &gt; 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 <xsl:choose> Element

- The <xsl:choose> element is used in conjunction with <xsl:when> and <xsl:otherwise> to express multiple conditional tests.
- **Syntax**

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

  - To insert a multiple conditional test against the XML file, add the <xsl:choose>, <xsl:when>, and <xsl:otherwise> elements to the XSL file:

THANKYOU

XSLT <xsl:apply-templates> Element

- The <xsl:apply-templates> element applies a template to the **current element or to the current element's child nodes**.
- If we add a **select attribute** to the <xsl:apply-templates> element it will process only the child element that matches the value of the attribute.
  - We can use the select attribute to specify the order in which the child nodes are processed.