ILAHIA COLLEGE OF ENGINEERING

WEB TECHNOLOGIES

Module V

Introduction to Data Interchange Formats

XML: The Syntax of XML, XML Document Structure, Namespaces, XML Schemas, Displaying Raw XML Documents, Displaying XML Documents with CSS, XSLT Style Sheets, XML Applications.

JSON(Basics Only): Overview, Syntax, Data types, Objects, Schema, Comparison with XML.

1. INTRODUCTION

SGML is a meta-markup language is a language for defining markup languages. Developed in the early 1980s; In 1986 SGML was approved by ISO standard. HTML was developed using SGML in the early 1990s . . The first XML standard, 1.0, was published in February 1998. The second, 1.1, was published in 2004.We use XML 1.0 because this newer version is not yet widely supported.

Motivations for the development of XML is because of the problems associated with HTML**:**

Problems with HTML:

1. HTML is defined to describe the general form and layout of information in web documents without considering its meaning of the information.

To describe a particular kind of information, it would be necessary to have tags indicating the meaning of the element's content. That would allow the processing of specific categories of information in a document. For example, if the price of a used car is stored as the content of an element named price, an application could find all cars in the document that cost less than $20,000.

2. HTML defines fixed set of tags and attributes. Given tags must fit every kind of document.

3. There are no restrictions on arrangement or order of tag appearance in HTML document. For example, an opening tag can appear in the content of an element, but its corresponding closing tag can appear after the end of the element in which it is nested.

Eg : Now is the time

One solution addresses the deficiencies of HTML is to allow for group of users with common needs to define their own tags and attributes and then use the SGML standard to define a new markup language to meet those needs. Each application area would have its own markup language.

Problem with using SGML:

1. It's too large and complex to use and it is very difficult to build a parser for it. SGML includes a large number of capabilities that are only rarely used.

2. A program capable of parsing SGML documents would be very large and costly to develop.

3. SGML requires that a formal definition be provided with each new markup language. So having area-specific markup language is a good idea, basing them on SGML is not.

A better solution: Define a simplified version of SGML and allow users to define their own markup languages based on it. XML was designed to be that simplified version of SGML.

XML is not a replacement for HTML . Infact two have different goals . HTML is a markup language used to describe the layout of any kind of information and provide some guidance as to how it should be displayed. XML is a meta-markup language that provides framework for defining specialized markup languages .

HTML Syntax

<html>

<head><title>name</title></head>……

XML Syntax

<name>

<first> nandini </first>

<last> sidnal </last>

</name>

XML is far more than a solution to the deficiencies of HTML:

Advantages of XML or Why XML is called universal data interchange language ?

It provides a simple and universal way of storing any textual data. Data stored in XML documents can be electronically distributed and processed by any number of different applications. These applications are relatively easy to write because of the standard way in which the data is stored. Therefore, **XML is a universal data interchange language**.

XM L BASICS

- XML is not a markup language. It is meta markup language that specifies rules for creating markup languages.When designing markup language using XML, the designer must define a collection of tags that are useful in the intended area.
- XML tag and its content, together with closing tag is called ***element*** .
- XML has no hidden specifications .Therefore XML documents are plain text which is easily readable by both people and application program.
- XML based markup language is called ***tag set.***
- Document that uses XML based markup language is called ***XML document*** .
- There are many XML oriented text editors that assist with the creation and maintenance of XML documents. Eg:Altova XMLspy, XMLFox etc.
- An ***XML application*** is a program that processes information stored in an xml document.
- An ***XML processor*** is a program that parses XML documents and provides the parts to an application .

- Application programs that process the data in XML documents must analyze the documents before they gain access to the data. This analysis is performed by an XML processor, which has several tasks, one of which is to parse XML documents, a process that isolates the constituent parts (such as tags, attributes, and data strings) and provides them to an application.

- All contemporary browsers support XML. Both IE7 and FX2 support basic XML .

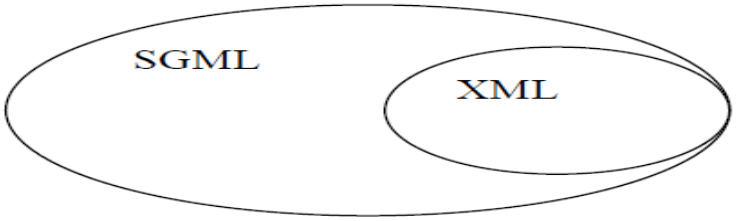
XML Vs HTML

HTML	XML
1. HTML is HyperText Markup Language.	1. XML is eXtensible Markup Language.
2. It is used for displaying information and to format the document.	2. It is designed to describe data and to focus on what data is?
3. HTML is not extensible. The user can't modify the structure or format by adding your tags.	3. It is Extensible. It allows the author to define a particular structure.
4. HTML tags are predefined. Eg: <html>,<head>	4. Tags are not predefined. Eg: <person>,<name>
5. Closing tags are mostly optional.	5. Closing tags are compulsory.
6. HTML is not case sensitive.	6. XML is highly case sensitive.
7. HTML has no Document Type Definition (DTD).	7. XML uses DTD to describe data elements used in the document.
8. Document display is direct and easy using any web browser with HTML	8. XML need XSL interaction for web browser display of document.
9. Cascading Style Sheet (CSS) a style sheet standard for HTML can be embed within HTML code.	9. In XML presentation and content are kept separate i.e. XSL page is acting independently.
10. About displaying the information.	10. About describing the information.
11. HTML was designed to display data and to focus on how data looks.	11. XML was designed to describe data and to focus on what data is.
12. HTML documents needn't be in a well formed structure.	12. XML documents must be in a well-formed structure.
13. HTML lacks syntactic checking. So we cannot validate HTML code.	13. We can validate XML code.
14. Sample code: <html> <head> <title> This is my homepage </title> <body bgcolor = "FFFFFF"> WELCOME </body> </head> </html>	14. Sample code: <?xml version="1.0"?> <person> <name > Aswathy </name> < age> 21 </age> <place> Trivandrum </place> </person>

XML as a Subset of SGML

- SGML is a very powerful, very general and a standard markup language. But with that power comes the increased complexity.
- XML is a subset of SGML intended to make SGML" light " enough for use on web.

- As XML is a proper subset of SGML, all XML documents are valid SGML documents .But not all SGML documents are valid XML document.
- XML can be considered as SGML-Lite: 20% of SGML's complexity, 80% of its capacity.
- XML is a lightweight cut-down version of SGML i.e. XML uses only the most commonly-used SGML features.



SGML Features

- The term SGML stands for Standard Generalized Markup Language
- It is a system for defining the markup language.
- SGML is a meta language .It facilitates the creation of other languages.
- SGML is extensible .
- It is widely used to manage large document that are subject to frequent revisions and need to be print in different format.

2.THE SYNTAX OF XML/ XML SYNTAX RULES

The syntax of XML can be thought of at **two distinct levels**.

1. There is the general **low-level syntax of XML that imposes its rules on all XML documents.**

2. The other syntactic level is specified by either **document type definitions or XML schemas**. These two kinds of specifications impose structural syntactic rules on documents written with specific XML tag sets.

- DTDs and XML schema specify the set of tags and attributes that can appear in particular document or collection of documents, and also the orders and various arrangements in which they can appear.
- DTD‘s and XML schema can be used to define a XML markup language.
- All XML documents begin with XML declaration.

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

. It identifies the document as being XML and provides the version no. of the XML standard being used. It will also include encoding standard.

XML document can include several different kinds of statements.

1) Data elements

2) Markup declarations - instructions to XML parser

3) Processing instructions – instructions for an applications program that will process the data described in the document.

Basic XML Rules

- **All XML document must begin with XML declaration. It identifies the document as being XML and provides the version no. of the XML standard being used. It will also include encoding standard.**
- **Comments in XML are same as HTML**
  <!-- This is a comment -->
- **XML names must begin with a letter or underscore and can include digits, hyphens, and periods.**
- **XML names are case sensitive. , the tag <Letter> is different from the tag <letter>.**
- **There is no length limitation for names.**
- **Space is Preserved in XML**
- HTML truncates multiple white-space characters to one single white-space**: With XML, the white-space in a document is not truncated.**
- **In XML closing tags are necessary.**
- **In XML, all elements must be properly nested within each other:**
- **XML Documents Must Have a Root Element .It contain one element that is the parent of all other elements. This element is called the root element.**

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

- XML tags can have attributes, which are specified with name/value assignments. XML Attribute Values must be enclosed with single or double quotation marks. **XML document that strictly adheres to these syntax rule is considered as well formed.**

Example :

```
<?xml version = "1.0" encoding =" utf-8"? >
<ad>
<year>1960</year>
<make>Cessna</make>
<model>Centurian</model>
<color>Yellow with white trim</color>
<location>
     <city>Gulfport</city>
     <state>Mississippi</state>
</location>
</ad>
```

None of this tag in the document is defined in HTML, all are designed for the specific content of the document.

- When designing an XML document, the designer is often faced with the choice between adding a new attribute to an element or defining a nested element.

- In some cases there is no choices.

- In other cases, it may not matter whether an attribute or a nested element is used.

- Nested tags are used

```
<!-- A tag with one attribute -->
<patient name = "Maggie Dee Magpie">
...
</patient>
-------------------------------------------------------------
<!-- A tag with one nested tag -->
<patient>
<name> Maggie Dee Magpie </name>
...
</patient>
-------------------------------------------------------------
<!-- A tag with one nested tag, which contains  three nested tags -->
<patient>
<name>
<first> Maggie </first>
<middle> Dee </middle>
<last> Magpie </last>
</name>
...
 </patient>
-------------------------------------------------------------
```

Here third one is a better choice because it provides easy access to all of the parts of data.

WELL FORMED AND VALID XML DOCUMENTS

- **Well-formed documents applies basic xml rules on all its Documents**
- **Valid documents are well-formed and also specified by DTDs or xml schemas**
- ***DTDs or xml schemas specify the set of tags and attributes that can appear in a particular document/documents and also the order in which they appear.***
- **Xml with correct syntax is "well formed" xml.**
- **Xml validated against a dtd/schema is "valid" xml.**

- A **"valid" xml** document is a "well formed" xml document, which also conforms to/ suitable with the rules of a document type definition (dtd)/schema:

Xml dtd

- Defines the legal building blocks of an xml document
- Can be inline in xml or as an external reference .

Xml schema

- An xml based alternative to dtd, more powerful and support namespace and data types.

3.XML DOCUMENT STRUCTURE

An XML document often uses two auxiliary files:

• One to specify the structural syntactic rules (DTD / XML schema)

• One to provide a style specification to describe how the content of the document to be printed (CSS /XSLT Style Sheets)

XML documents are made up by the following building blocks :

Elements, Tags, Attributes, Entities, PCDATA, and CDATA

1. **ELEMENTS**

Elements are the basic building blocks of XML files.

Elements are the main building blocks of both XML and HTML documents.

Elements can contain text, other elements, or be empty.

2 .TAGS

Tags are used to markup elements. A starting tag like <element_name> mark up the beginning of an element, and an endingtag like </element_name> mark up the end of an element.

Examples:

```
A message element: <message>some message in between</message>
```

Tag names are case sensitive. XML supports two types of elements, closed and empty (open) elements. Closed elements consist of both opening(start) and closing(ending) tags. The following example presents a closed element. In the closing tag, a forward slash precedes the element name.

```
<Month>January</Month>
```

Elements can be nested, and all elements must be nested within a single root element. Elements must be nested correctly, with child elements enclosed within their parent opening and closing element tags, as follows:

```
<Year>
<Month>January</Month>
<Month>February</Month>
```

</Year>

Empty (open) elements contain no content. An empty or open element can be used to mark sections of the document for the processor. .Empty element can have attributes.<hello happy="TRUE"/> is valid. An empty element has the following syntax; the element name is followed by a slash. Eg: <Year/>. Empty element can also have matching start and end tags as given below.<hello></hello>.

Tag	TagMeaning
<greeting>	Starts a greeting element
</introduction>	Ends an introduction element
<Joe Black>	Bad start tag .No space allowed
<42>	Element name cannot begin with number
</ Product>	No space allowed b/w slash and element name

3.ATTRIBUTES

Attributes provide extra information about elements. Attributes are placed inside the start tag of an element. Attributes come in name/value pairs. <fruit type="apple">: type attribute have value "apple. The attribute *type* provide additional information about fruit .The name of the element is "fruit". The name of the attribute is "type".

Attribute Assignment	Meaning
1) <fruit type="apple">	type attribute have value "apple
2) <fruit type='apple'>	Single quotes can also be used
3) <table border=2>	Invalid.Attributes must be quoted
4) <animal leg="4" blood="cold">	The leg attribute has the value 4,blood attribute has the value . White space within start tag are ignored by the parser.

4.PCDATA

PCDATA means parsed character data. Think of character data as the text found between the start tag and the end tag of an XML element. PCDATA is text that will be parsed by a parser. Tags inside the text will be treated as markup and entities will be expanded.

5.CDATA

CDATA also means character data. This section is used *to shield a body of text from the attentions of the XML processor.* CDATA is text that will NOT be parsed by a parser. Tags inside the text will NOT be treated as markup and entities will not be expanded that is used to insert an extra space in an HTML document.Entities are expanded when a document is parsed by an XML parser.

syntax.

<![CDATA[content]]>

Example

<Document>

<![CDATA[if a<b and b<c then a<c]] >

</Document>

6.ENTITIES

An XML document consist of one or more entities that are logically related collection of information, . Entities range from a single special character to a book chapter. An XML document has one document entity. The document entity can be the entire document , but in many cases it includes references to the names of entities that are stored elsewhere. For eg: the document entity for a technical article might contain beginning and ending material but have references to article body sections, which are entities stored in separate files..All other entities except the document entity must have a name.

Reasons to break a document into multiple entities:

1. Good to define a Large documents as a smaller no. of parts to make it more easier to manage .

2. If the same data appears in more than one place in the document, defining it as an entity allows any no. of references to a single copy of data.

3. Many documents include information that cannot be represented as text, such as images. Such information units are usually stored as binary data. Binary entities can only be referenced in the document entities because XML documents cannot include binary data.

Rules of Entity names:

• No length limitation

• Must begin with a letter, a dash, or a colon

• Can include letters, digits, periods, dashes, underscores, or colons.

- A reference to an entity (Entity Reference) has the form, name with prepended ampersand and appended semicolon: &entity_name; Eg. &apple_image;

➢ When the processor parses the document it will replace the entity reference with actual characters.

- One of the common uses of entities is to allow characters that are normally used as markup delimiters to appear as themselves in a document. Because of this XML includes the entities that are predefined for HTML.

Five Built-in Entity **Entity Reference**	 **Interpretation**
&lt;	<
&gt;	>
&amp;	&
&apos;	‘
&quot;	”

- Regardless of the entity types, all entities are referenced in the same way: ***&name;*** This code will include the simple entity

```
<!entity iso "International Organization for Standardization">
```

within a sentence :

```
The &iso; sets the standard for character encoding.
```

when interepted by an XML parser the result is the

```
The International Organization for Standardization sets the standard for character encoding.
```

- If several predefined entities must appear near each other in a document, it is better to avoid using entity references. Character data section can be used. The content of a character data section is not parsed by the XML parser, so it can include any tags.
- The form of a character data section is as follows: **<![CDATA[content]]>** // no tags can be used since it is not parsed For example, instead of Start >>>> HERE <<<< use <![CDATA[Start >>>> HERE <<<<]]>
- The opening keyword of a character data section is not just CDATA, it is in effect [CDATA[. There can be any spaces between [and C or between A and [.
- Because the content of Character data section is not parsed by the XML,any entity references that are included are not expanded.For Eg: the content of the line
- `<![CDATA[The form of a tag is <tag name>]]>`
  is as follows .

The form of a tag is <tag name>

7.PROCESSING INSTRUCTION (PI).

PI's are defined as markup that provides information to be used by s/w application. PI's begins with "<?" and ends with "?>" pair.XML itself make use of processing instruction in what is known as XML declaration. The simplest form of PI which should head up the entire XML document is<?xml version="1.0" encoding="utf-8"?>

DOCUMENT TYPE DECLARATION

1. A Document Type Declaration is a statement embedded in an XML document whose purpose is to acknowledge the existence and location of Document Type Definition(DTD).
2. Document Type Declaration is a statement that points to the Document Type Definition(DTD) .
3. Document Type Definition is a set of rules that defines the structure of an XML document where as a Document Type Declaration is a statement that tells the parser which DTD to use for checking and validation.
4. All Document Type Declaration starts with a string "<!DOCTYPE "
5. **The Document Type Declaration can be external or internal**
6. **If external the DTD must be specified either as SYSTEM or PUBLIC.**
7. If **PUBLIC** the DTD can be used by anyone by referring the URL.
8. If **SYSTEM** that means it resides on local harddisk and may not be available for use by other application. For example suppose there is an XML document called myfile.xml that we want to parse and validate against a DTD called my-rules.dtd .

- The DTD for a document can be
  - **internal** (embedded in XML document) or
  - **external** (separate file)- can be used with more than one document.
- DTD declarations have the form: <!keyword … >
- There are four possible declaration keywords:
- ELEMENT, ATTLIST, ENTITY, and NOTATION
- ELEMENT- used to define tags
- ATTLIST – used to define tag attributes
- ENTITY- used to define entities
- NOTATION – used to define datatype notations

Declaring attributes

An attribute declaration must include the name of the element to which the attribute belongs, the attribute name and its type.

Syntax

- **<!ATTLIST *element-name attribute_name attribute type [default-value*]>**

If More than one attribute

- **<!ATTLIST element_name attribute_name1 attribute_type default_value_1**
  attribute_name2 attribute_type default_value_2>

Attribute type :There are ten different types, but we will consider only CDATA .

- Possible Default value for attributes:
- **Value** - value ,which is used if none is specified in an element (as default)
- **#Fixed value** - value ,which every element have and can't be changed (as constant)
- **# Required** - no default value is given ,every instance must specify a value
- **#Implied** - no default value is given ,the value may or may not be specified (optional)
- Example :

```
<!ATTLIST car doors CDATA "4">
<!ATTLIST car engine_type CDATA #REQUIRED>
<!ATTLIST car make CDATA #FIXED "Ford">
<!ATTLIST car price CDATA #IMPLIED>
The following xml element is valid for this DTD
<car doors = "2" engine_type = "V8">
...
</car>
```

Attribute that include fixed in the DTD may or may not be specified in particular element instances.

+ - one or more occurences
* - zero or more occurences
? - zero or one occurrence

- Eg DTD Declaration

<!ELEMENT person (parent+, age, spouse?, sibling*)>

Means parent element can occur one or more times, spouse element can occur zero or one time. Sibling element can occur zero or more times.

INTERNAL DTD

- If the DTD's are internal then the syntax is
- **<!DOCTYPE root-element [<!internal type definition>]>**

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE mail [
      <!ELEMENT mail(to,from,heading,body)>
```

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

            <!ELEMENT from (#PCDATA)>

            <!ELEMENT heading (#PCDATA)>

            <!ELEMENT body (#PCDATA)>]>

      <mail>

      <to>Rani</to>

      <from>Ravi</from>

      <heading>Remainder</heading>

      <body>About our parents Wedding Anniversary</body>

      </mail>
```

EXTERNAL DTD

- The DTD section can be placed in an external file .
- If the Document Type Declaration is external then the DTD must be specified either as SYSTEM or PUBLIC in the Document Type Declaration.
- If the DTD's are external then the syntax is

<!DOCTYPE root-element SYSTEM "DTD_file_URL**">**

or

<!DOCTYPE root-element PUBLIC "public_id" "DTD_file_URL**" >**

- If **SYSTEM** the DTD resides on the local hard disk and may not be available for use by other applications. If **PUBLIC** the DTD can be used by anyone by referring the URL . The public_id is unique for standard DTD files on the web.
- **//mail.xml**

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

  <!DOCTYPE mail SYSTEM "mail.dtd">

  <mail>

  <to>Rani</to>

  <from>Ravi</from>

  <heading>Remainder

  </heading>
```

```
<body>About our parents Wedding Anniversary</body>

</mail>
```

//mail.dtd

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

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

<!ELEMENT to (#PCDATA)>

<!ELEMENT from #PCDATA)>

<!ELEMENT heading (#PCDATA)>

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

4. XML NAMESPACES

- Xml namespaces provide a method to avoid element 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.
- An example of this situation is having a <table> tag for a category of furniture and a <table>tag from HTML for information tables.
- 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.

An XML namespace is a collection of element and attribute names used in XML documents. The name of the namespace usually has the form of a URL. But the XML processors never reference the site whose URL is used as the name of a namespace. *The purpose of the URL is to give the namespace a unique name.* The namespaces can be declared as the value of **xmlns** in the elements where they are used or in the xml root element.

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

- The namespace declaration has the following syntax:
  <element_name xmlns[:*prefix]* ="*URL*>"

• The square bracket indicates that what is within them is optional. prefix[optional] specify name to be attached to names in the declared namespace .

Two reasons for prefix :

1. Shorthand for URL is too long to be typed on every occurrence of every name from the namespace.
2. URL may includes characters that are illegal in XML

- **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>africancoffeetable</f:name>
  <f:width>80</f:width>
  <f:length>120</f:length>
</f:table>
</root>
```

OR

```
<root>
<h:table  xmlns:="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.abc.com/furniture">
 <f:name>africancoffeetable</f:name>
 <f:width>80</f:width>
 <f:length>120</f:length>
</f:table>
```

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="*namespaceURL*"

The next example declares two namespaces. The first is declared to be the default namespace; the second defines the prefix, cap: So we can avoid the use of prefix in the first section.

```
<states>
  xmlns = "http://www.states-info.org/states"
  xmlns:cap = "http://www.states-info.org/state-capitals"
  <state>
    <name> South Dakota </name>
    <population> 754844 </population>
    <capital>
      <cap:name> Pierre </cap:name>
      <cap:population> 12429 </cap:population>
    </capital>
  </state>
  <!-- More states -->
</states>
```

5. XML SCHEMAS

An XML schema is an XML document so it can be parsed with an XML parser.

5.1Schema Fundamentals:

• Schema are related idea of class and an object in an OOP language. A Schema is similar to a class definition ;an XML document that conforms to a specific schema are considered instances of that schema or object of the schema's class.

• Schemas have three primary purposes

- Specify elements and attributes of an XML language
- Specify the structure of its instance XML documents including where and how often the elements may appear.
- Specify the data type of every element in its instance XML documents .

Disadvantages of DTD compared to XML schema

- **Cannot be parsed by an XML parser**
- **All declarations in DTD are global (impossible to define two elements with same name.)**
- **DTD cannot control what kind of information a given element or attribute can contain.**

5.2 Defining a schema:

Schemas are written from a namespace(schema of schemas): The name of this namespace is

`http://www.w3.org/2001/XMLSchema`

`element`, **`schema`**, **`sequence`** and **`string`** are some names from this namespace Every XML schema has a single root, **`schema`**.

• The **`schema`** element must specify the namespace for the schema of schemas from which the schema's elements and its attributes will be drawn. It often specifies a prefix that will be used for the names in the schema. This name space specification appears as

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

Every XML schema itself defines a tag set like DTD, which must be named with the targetNamespace attribute of schema element. The target namespace is specified by assigining a name space to the target namespace attribute as the following:

`targetNamespace = http://cs.uccs.edu/planeSchema`

Every top-level element places its name in the target namespace If we want to include nested elements(ie if the names of the elements and attributes that are not defined directly in the schema element), we must set the `elementFormDefault` attribute to `qualified`.

`elementFormDefault = "qualified"`

The default namespace which is source of the unprefixed names in the schema is given

with another **`xmlns`** specification without the prefix.

`xmlns = "http://cs.uccs.edu/planeSchema "`

A complete example of opening tag for a schema element:

```
<xsd:schema
<!-- Namespace for the schema itself -->
xmlns:xsd = http://www.w3.org/2001/XMLSchema <!-- Namespace where elements defined here will be placed -->
targetNamespace = http://cs.uccs.edu/planeSchema
<!-- Default namespace for this document -->
xmlns = http://cs.uccs.edu/planeSchema.
<!-- Specify non-top-level elements to be in the target namespace-->
elementFormDefault = "qualified" >
```

One alternative to the preceding opening tag would be to make the XMLSchema names the default so that they do not need to be prefixed in the schema. Then the names in the target namespace would need to be prefixed. The following schema tag illustrates this approach:

```
<schema
  xmlns = "http://www.w3.org/2001/XMLSchema"
  targetNamespace = "http://cs.uccs.edu/planeSchema"
  xmlns:plane = "http://cs.uccs.edu/planeSchema"
  elementFormDefault = "qualified">
```

5.3 Defining a schema instance:

An instance of schema must specify the namespaces it uses. These specifications are given as attribute assignments in the tag for its root element

1. Define the default namespace

- if the root element is planes

<planes xmlns = http://cs.uccs.edu/planeSchema

...>

2. The second attribute specification in the root element of an instance document is for the `schemaLocation` attribute. This attribute is used to name the standard namespace for instances which includes the name XMLSchema-instance .(`XMLSchema-instance`)

xmlns:xsi ="http://www.w3.org/2001/XMLSchema-instance"

3. Specify location where the default namespace is defined, using the **schemaLocation** attribute, which is assigned two values namespace and filename.

xsi:schemaLocation ="http://cs.uccs.edu/planeSchema planes.xsd" >

Altogether, the opening root tag of an XML instance of the planes.xsd schema, where the root element name in the instance is planes, could appear as follows:

```
<planes
   xmlns = "http://cs.uccs.edu/planeSchema"
   xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation = "http://cs.uccs.edu/planeSchema
                         planes.xsd">
```

5.4. Schema Data types

Two categories of user-defined schema data types :

1.Simple data type is a data type whose content is restricted to strings only, no attributes and no nested elements(child elements) .It is possible to add restrictions(facets) to a datatype inorder to limit its content .

2. Complex datatype can have attributes and nested elements.

• XML Schema defines 44 data types .19 of which are primitive and 25 of which are derived.

• ***Primitive datatype***: string, Boolean, float, time …

• ***Predefined Derived datatype***: Examples byte, decimal, positiveInteger,.

• ***User-defined derived data types*** – These datatypes are defined by specifying restrictions on an existing type,which is then called a ***basetype***. Such user defined types are derived types. Constraints on derived types are given in terms of facets of the base type .Ex: interget data type has 8 possible facets : totalDigits, maxInclusive,,maxExclusive etc.

- Both simple and complex types can be either ***named or anonymous*** .

DTDs define global elements (context of reference is irrelevant). But context of reference is essential in XML schema

Data declarations in an XML schema can be

1. Local ,which appears inside an element that is a child of schema
2. Global, which appears as a child of schema

5.5. Defining a simple type:

• Elements are defined in an XML schema with the `element` tag, which is from the XMLSchema namespace. Recall that the prefix `xsd` is normally used for names from this namespace. Use the `element` tag and set the `name` and `type` attributes

<xsd:element name = "bird" type = "xsd:string" />

The instance could be :

<bird> Yellow-bellied sap sucker </bird>

• An element can be given default value using `default` attribute .Default value is automatically assigned to the element when no other value is specified.

<xsd:element name = "bird" type = "xsd:string" default="Eagle" />

• An element can have constant value, using `fixed` attribute .A fixed value is automatically assigned to the element , and we cannot specify another value.

<xsd:element name = "bird" type = "xsd:string" fixed="Eagle" />

Declaring User-Defined Types:

• User-Define type is described in a `simpleType` element, using facets . facets must be specified in the content of `restriction` element. **Restrictons on XML elements are called facets**. **Restrictions are**

used to define acceptable values for XML elements or attributes. Facets values are specified with the `value` attribute.

For example, the following declares a user-defined type , firstName

```
<xsd:simpleType name = "firstName" >
<xsd:restriction base = "xsd:string" >
<xsd:maxLength value = "20" />
</xsd:restriction>
</xsd:simpleType>
```

The `length` facet is used to restrict the string to an exact number of characters. The `minLength` facet is used to specify a minimum length.

```
<xsd:simpleType name = "phoneNumber" >
<xsd:restriction base = "xsd:decimal" >
<xsd:precision value = "10" />
</xsd:restriction>
</xsd:simpleType>
```

Declaring Complex Types:

• There are several categories of complex types, but we discuss just one, `element-only elements` which can have elements in their content, but no text. Element-only elements are defined with the `complexType` tag.

• The `sequence` tag is used to contain an ordered group of elements.

• Use the `all` tag if the order is not important .

• Nested elements can include attributes that give the allowed number of occurrences (minOccurs, maxOccurs, unbounded)

• For ex:

```
<xsd:complexType name = "sports_car" >
<xsd:sequence>
<xsd:element name = "make" type = "xsd:string" />
<xsd:element name = "model" type = "xsd:string" />
<xsd:element name = "engine" type = "xsd:string" />
<xsd:element name = "year" type = "xsd:decimal" />
</xsd:sequence>
</xsd:complexType>
```

Complete example of schema

Example 1:

Schema definition document

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.w3schools.com"
xmlns="http://www.w3schools.com"
elementFormDefault="qualified">

<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>
```

Schema Instance Document

```
<?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>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

Example 2

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

<!-- planes.xsd
     A simple schema for planes.xml
```

```
    -->
<xsd:schema
   xmlns:xsd = "http://www.w3.org/2001/XMLSchema"
   targetNamespace = "http://cs.uccs.edu/planeSchema"
   xmlns = "http://cs.uccs.edu/planeSchema"
   elementFormDefault = "qualified">

   <xsd:element name = "planes">
     <xsd:complexType>
       <xsd:all>
         <xsd:element name = "make"
                      type = "xsd:string"
                      minOccurs = "1"
                      maxOccurs = "unbounded" />
       </xsd:all>
     </xsd:complexType>
   </xsd:element>
</xsd:schema>
```

Notice that we use the `all` element to contain the single element of the complex type `planes`, although `sequence` could have been used instead.

An XML instance that conforms to the planes.xsd schema is as follows:

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

<!-- planes1.xml
     A simple XML document for illustrating a schema
     The schema is in planes.xsd
     -->
<planes
  xmlns = "http://cs.uccs.edu/planeSchema"
  xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation = "http://cs.uccs.edu/planeSchema
                        planes.xsd">
    <make> Cessna </make>
    <make> Piper </make>
    <make> Beechcraft </make>
</planes>
```

5.7. Validating Instances of Schemas:

An XML schema provides a definition of a category of XML documents. However, developing a schema is of limited value unless there is some mechanical way to determine whether a given XML instance document confirms to the schema. Several XML schema validation tools are available eg. xsv(XML schema validator) This can be used to validate online. Output of xsv is an XML document. When run from command line output appears without being formated.

If schema is not in the correct format, the validator will report that it could not find the specified schema.

6.DISPLAYING RAW XML DOCUMENTS

An XML enabled browser or any other system that can deal with XML documents cannot possibly know how to format the tags defined in the xml documents. Without a style sheet that defines presentation styles for the document tags the XML documents cannot be displayed in a formatted manner. Most contemporary browsers like FX3 have default style sheets that are used when style sheets are not defined for xml documents.

//mail.xml

```
<?xml version="1.0"?>
<mail>
        <to>Rani</to>
        <from>Ravi</from>
        <heading>Remainder</heading>
        <body>
                <title>Dear Sister</title>
                <wish>Good Morning to you</wish>
                <msg>Please dont forget about our parents Wedding Anniversary today.Be there in time</msg>
        </body>
</mail>
```

Raw xml document mail.xml

```
− <mail>
    <to>Rani</to>
    <from>Ravi</from>
    <heading>Remainder</heading>
  − <body>
      <title>Dear Sister</title>
      <wish>Good Morning to you</wish>
    − <msg>
        Please dont forget about our parents Wedding Anniversary today.Be there in time
      </msg>
    </body>
  </mail>
```

Some of the elements in the display are preceded by dashes. These elements can be elided (temporarily suppressed) by placing the mouse cursor over the dash and clicking the left mouse button. For example, if the mouse cursor is placed over the dash to the left of the first<ad> tag and the left mouse button is clicked, the result is as shown below.

7. DISPLAYING XML DOCUMENTS WITH CSS

Style sheet information can be provided to the browser for an xml document in two ways. First, a CSS file that has style information for the elements in the XML document can be developed. Second the XSLT style sheet technology can be used. Using CSS is effective, XSLT provides far more power over the appearance of the documents display.

A CSS style sheet for an XML document is just a list of its tags and associated styles. The connection of an XML document and its style sheet is made through an **xmlstylesheet** processing instruction .**Display** is used to specify whether an element is to be displayed inline or in a separate block.

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

For example: planes.css

```
ad { display: block; margin-top: 15px; color: blue;}
year, make, model { color: red; font-size: 26pt;}
color {display: block; margin-left: 20px; font-size: 12pt;}
description {display: block;color: green; margin-left: 20px; font-size:
10pt;}
seller { display: block; margin-left: 15px; font-size: 14pt;}
location {display: block; color: cyan;margin-left: 40px; }
city {font-size: 12pt;}
state {font-size: 12pt;}
```

planes.dtd document

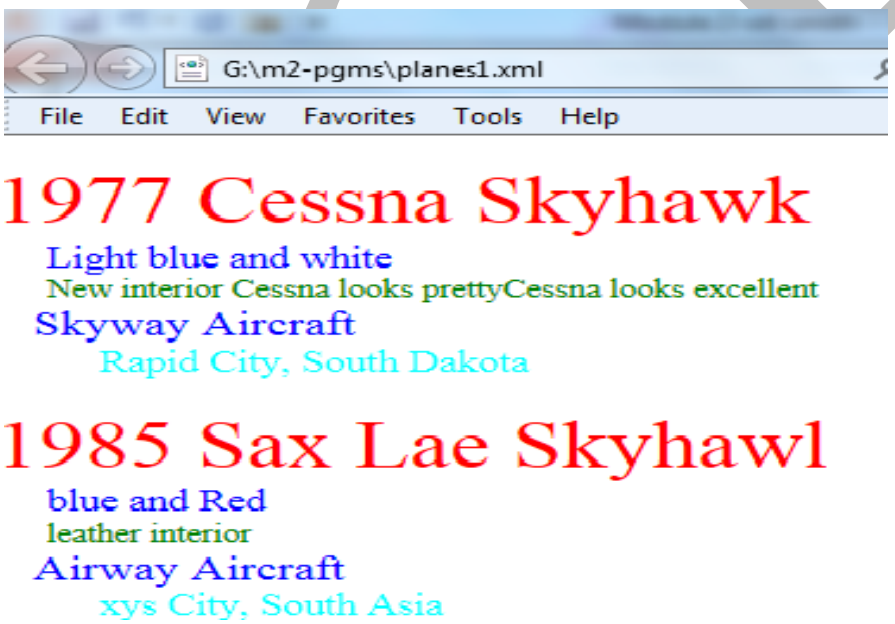
```
<!ELEMENT planes_for_sale (ad+)>
<!ELEMENT ad (year, make, model, color, description, price?, seller, location)>
<!ELEMENT year (#PCDATA)>
<!ELEMENT make (#PCDATA)>
<!ELEMENT model (#PCDATA)>
<!ELEMENT color (#PCDATA)>
<!ELEMENT description (#PCDATA)>
<!ELEMENT price (#PCDATA)>
<!ELEMENT seller (#PCDATA)>
<!ELEMENT location (city, state)>
<!ELEMENT city (#PCDATA)>
<!ELEMENT state (#PCDATA)>
<!ATTLIST seller phone CDATA #REQUIRED>
<!ATTLIST seller email CDATA #IMPLIED>
<!ENTITY c "Cessna">
<!ENTITY p "Piper">
<!ENTITY b "Beechcraft">
```

Planes.xml

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

```
<!DOCTYPE planes_for_sale SYSTEM "planes.dtd">
<planes_for_sale>
<ad>
<year> 1977 </year>
<make> &c; </make>
<model> Skyhawk </model>
<color> Light blue and white </color>
<description> New paint, nearly new interior,
685 hours SMOH, full IFR King avionics </description>
<price> 23,495 </price>
<seller phone = "555-222-3333"> Skyway Aircraft </seller>
<location>
<city> Rapid City, </city>
<state> South Dakota </state>
</location>
</ad>
</planes_for_sale>
```

With planes.css the display of planes.xml as following:



STORING XML DATA IN HTML DOCUMENT

- Data island:

An XML data island is Extensible Markup Language (**XML) embedded in an HTML document**.

- In an external data island, data (xml) and presentation of data (HTML) are being seperated.
- In internal data island data and presentation of data is in the same document.

- The HTML element **xml** is used to embed XML in HTML documents.
- Data islands are created by assigning an **ID attribute** to the xml tag.
- The source src attribute specifies the data file location.

<xml id="customers" src="customers.xml"/>

- IE's XML parser creates a data island from an XML document by storing the data as a Data Source Object (DSO).).
- Interactions between data islands and the Web page are controlled by the DSO.

internal data island

```
<html>
<body>
<xml id="products">
<Products>
<Item>
<Name>Coffee Cup Warmer</Name>
<Number>0001</Number>
<Price>15.00</Price>
</Item>
<Item>
<Name>42 Cup Coffee Brewing System</Name>
<Number>0015</Number>
<Price>473.00</Price>
</Item></Products>
</xml></body>
</html>
```

DISPLAYING XML DATA IN HTML BROWSER AS HTML TABLES

- Data islands has the ability to bind HTML tables.
- The element attributes datasrc, datafld allow browser to display bound data.
- **The datasrc attribute specifies the data island source with a URI.**
- **The datafld attribute specifies elments in the XML data source**.
- The datasrc attribute links an HTML element to a DSO.
- The attribute property specifing the unique identifier of a DSO must be prefixed by a number sign (#).
- The datafld attributes reference specific elements in the XML data source. The following syntax is required:

<tag datasrc="#id" datafld="field_name">

This is an example of external style sheets.ie data and presentation of data are in different file.

//catalogs.xml

```
<?xml version="1.0"?>
  <PCS>
    <PC>
     <NAME>Zenith</NAME>
     <CAPACITY>100</CAPACITY>
     <PRICE>20000</PRICE>
    </PC>
  <PC>
     <NAME>DELL</NAME>
     <CAPACITY>200</CAPACITY>
     <PRICE>40000</PRICE>
   </PC>
   <PC>
     <NAME>Acer</NAME>
     <CAPACITY>300</CAPACITY>
     <PRICE>35000</PRICE>
   </PC>
 </PCS>
```

//catalog1.html

```
<html>
<body>
<xml id="PCS" src="catalogs.xml">
</xml>
<table datasrc="#PCS" border="1" align="center">
<thead><th>Name</th><th>Capacity</th><th>Price</th></thead>
<tr><td><div datafld="NAME"></td>
<td><div datafld="CAPACITY"></td>
<td><div datafld="PRICE"></td>
</tr>
</table>
</body>
</html>
```

Name	Capacity	Price
Zenith	100	20000
DELL	200	40000
Acer	300	35000

Example 2

Product.xml

```
<?xml version="1.0" ?>
<Products>
<Item>
<Name>Coffee Cup Warmer</Name>
<Number>0001</Number>
<Price>15.00</Price>
</Item>
<Item>
<Name>42 Cup Coffee Brewing System</Name>
<Number>0015</Number>
<Price>473.00</Price>
</Item></Products>
```

- In the following html document(saved as filename.html), external data island is used(product.xml)

```
<HTML>
<HEAD>
<TITLE>Show XML In Your HTML</TITLE>
</HEAD>
<BODY>
<XML ID="MyProducts" SRC="Product.xml"></XML>
<TABLE DATASRC="#MyProducts" BORDER="1">
<THEAD>
<TH>Item Name</TH>
<TH>Item Number</TH>
<TH>Price</TH>
</THEAD>
<TR>
<TD><Span DATAFLD="Name"/></TD>
<TD><SPAN DATAFLD="Number"/></TD>
```

```
<TD><SPAN DATAFLD="Price"/></TD>
</TR>
</TABLE>
</BODY>
</HTML>
```

Item Name	Item Number	Price
Coffee Cup Warmer	0001	15.00
42 Cup Coffee Brewing System	0015	473.00

Example

Write an application to create an Address Book in XML and display the Address book in the browser using HTML.

1) Create a valid XML document for Address Book with following elements.The address book stores 'N' persons Name(fname,lname),Address(hname,city,state,country) and all the phone numbers(mob,off,res).

2) The sub elements are given in the bracket. Store the xml data in an HTML page and display the data in HTML browser as HTML Tables. As we are storing the XML data in HTML page , there is only one HTMLfilecalledaddressbook.html

3) Use XML tag for storing the content in an HTML page.

NAME	ADDRESS	EMAIL	PHONE
Ammu Kurian	Vadakkan Pala Kerala India	ammu@yahoo.com	Res:647346 Mob:97878473 Off:38463486
Anjujohn	Anju Nivas Kavadiyar,Tvm Kerala India	ammu@hotmail.com	Res:4447346 Mob:4548473 Off:3763486

8.EXTENSIBLE STYLE SHEET LANGUAGE (XSL)

- A family of specifications for transforming XML documents. It consist of three standards:
- **XSLT**: specifies how to transform documents(xml to html)
- **Xpath**: identify parts of xml documents, such as specific elements that are in specific positions in the document or element that have particular attribute values.
- **XSL-FO**: is used to generate high quality printable documents in formats such as PDF etc.
- Together they provide powerful means of formatting xml documents.
- XSLT describes how to transform XML documents into different form or formats such as HTML ,Plain text etc.

Overview of XSLT

XSLT processors take both an XML document and an XSLT document as input. The XSLT document is the program to be executed; the XML document is the input data to the program. Parts of the XML document are selected, possibly modified, and merged with parts of the XSLT document to form a new document, which is sometimes called an ***XSL document***. Note that the XSL document is also an XML document, which could be again the input to an XSLT processor. **The output document can be stored for future use by applications, or it may be immediately displayed by an application, often a browser**. Neither the XSLT document nor the input XML document is changed by the XSLT processor.

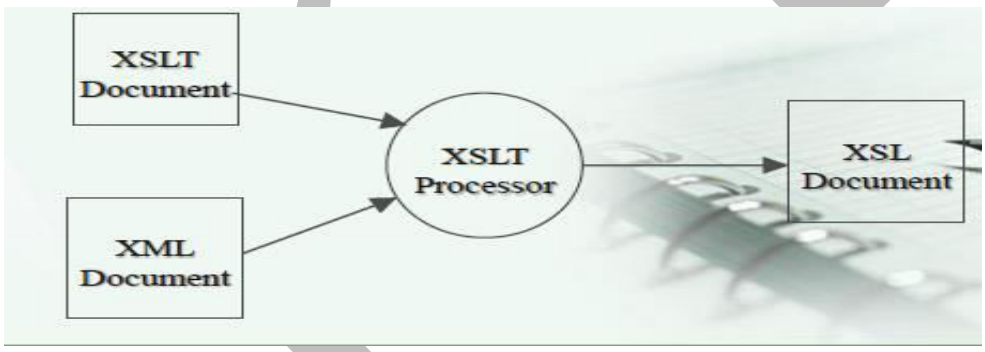


Figure:XSLT processing

8.2 XSL Transformations for Presentation

XSLT includes more than 50 formatting object(element) types and more than 230 attributes, so it is a large and complex tag set.

In this section we assume that the XSLT processor process an XML document with its associated XSLT style-sheet document and produces as its output and XSL document that is an HTML document to be displayed.

An XML document that is to be used as data to an XSLT style sheet must include a processing instruction to inform the XSLT processor that the style sheet is to be used. The form of this instruction is as follows:

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

The following is a simple example of **XML document** that illustrate XSLT formatting:

```
<?xml version = "1.0" encoding = "utf-8"?>
<!-- xslplane.xml -->
<?xml-stylesheet type = "text/xsl"  href = "xslplane.xsl" ?>
<plane>
  <year> 1977 </year>
  <make> Cessna </make>
  <model> Skyhawk </model>
  <color> Light blue and white </color>
</plane>
```

Notice that this document specifies xslplane.xsl as its XSLT style sheet. **An XSLT style sheet is an XML document whose root element** is the special-purpose element **`stylesheet.`** The **`stylesheet`** tag defines namespaces as its attributes and encloses the collection of elements that defines its transformations. It also identifies the document as an XSLT document. The namespace for all XSLT elements is specified with a W3C URI. If the style sheet includes XHTML elements, the style sheet tag also specifies the XHTML namespace. In the stylesheet tag

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

Notice that this tag specifies that the prefix for XSLT elements is xsl and the default namespace is that for XHTML/HTML.

A style-sheet document must include at least one **`template`** element. The template opening tag includes a **`match`** attribute to specify an XPath expression that **selects a node in the XML document**. **The content of a template element specifies what is to be placed in the output document.** In many XSLT documents, a template is included to match the root node of the XML document. This can be done in two ways. One way is to use the XPath expression **"/",** as in

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

The alternative to use '/' is to use the **actual root of the document**. In the example xslplane.xml the document root is plane.

If the output of the XSLT processor is an XHTML/HTML document, the template that matches the root node is used to create the HTML header of the output document. The header code appears as the content of the template element. An example of a complete template element is

```
<xsl:template match = "plane">
<html><head><title> Example </title></head><body>
...
</body></html>
</xsl:template>
```

Style sheets nearly always have templates for specific nodes of the XML document, which are descendants of the root node, as in the following example:

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

Template elements are of **two distinct kinds*: those that literally contain content and those that specify content to be copied from the associated XML document***.

XSLT elements that represent HTML elements often are used to specify content. XSLT elements have the appearance of their associated HTML elements, like the following HTML element:

```
<span style = "font-size: 14pt"> Happy Holidays! </span>
```

In many cases, the **content of an element of the XML document is to be copied to the output document. This is done with the `value-of` element, which uses a select attribute to specify the element of the XML document whose contents are to be copied.** For example, the element

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

specifies that the content of the AUTHOR element of the XML document is to be copied to the output document. Because **the `value-of` element cannot have content**, it is terminated with a slash and a right angle bracket. The `select` attribute can specify any node of the XML document. This is an advantage of XSLT formatting over CSS, in which the order of data as stored is the only possible order of display.

CONVERTING XML INTO HTML WITH XSL

Example1:

```
<?xml version = "1.0" encoding = "utf-8"?>
<!-- xslplane2.xsl
     An XSLT Stylesheet for xslplane.xml using implicit templates
     -->
<xsl:stylesheet version = "1.0"
                xmlns:xsl = "http://www.w3.org/1999/XSL/Transform"
                xmlns = "http://www.w3.org/1999/xhtml">

<!-- The template for the whole document (the plane element) -->

  <xsl:template match = "plane" >
    <html><head><title> Style sheet for xslplane.xml </title>
    </head><body>
    <h2> Airplane Description </h2>
    <span style = "font-style: italic; color: blue;"> Year:
    </span>
    <xsl:value-of select = "year" /> <br />
    <span style = "font-style: italic; color: blue;"> Make:
    </span>
    <xsl:value-of select = "make" /> <br />
    <span style = "font-style: italic; color: blue;"> Model:
    </span>
    <xsl:value-of select = "model" /> <br />
    <span style = "font-style: italic; color: blue;"> Color:
    </span>
    <xsl:value-of select = "color" /> <br />
    </body></html>
  </xsl:template>
</xsl:stylesheet>
```

An output document from the XSLT processor

The XSLT document, xslplane1.xsl, is more general and complex than necessary for the simple use for which it was written. There is actually no need to include templates for all of the child nodes of plane, because the select clause of the value-of element finds them. The following XSLT document, xslplane2.xsl, produces the same output as xslplane1.xsl.

Example 2:

```
<?xml version = "1.0" encoding = "utf-8"?>
<!-- xslplane2.xsl
     An XSLT Stylesheet for xslplane.xml using implicit templates
     -->
<xsl:stylesheet version = "1.0"
                xmlns:xsl = "http://www.w3.org/1999/XSL/Transform"
                xmlns = "http://www.w3.org/1999/xhtml">

<!-- The template for the whole document (the plane element) -->

  <xsl:template match = "plane" >
    <html><head><title> Style sheet for xslplane.xml </title>
    </head><body>
    <h2> Airplane Description </h2>
    <span style = "font-style: italic; color: blue;"> Year:
    </span>
    <xsl:value-of select = "year" /> <br />
    <span style = "font-style: italic; color: blue;"> Make:
    </span>
    <xsl:value-of select = "make" /> <br />
    <span style = "font-style: italic; color: blue;"> Model:
    </span>
    <xsl:value-of select = "model" /> <br />
    <span style = "font-style: italic; color: blue;"> Color:
    </span>
    <xsl:value-of select = "color" /> <br />
    </body></html>
  </xsl:template>
</xsl:stylesheet>
```

Airplane Description

Year: 1977
Make: Cessna
Model: Skyhawk
Color: Light blue and white

XML Applications

XML enables the users/organizations to create their own elements attributes and entities to describe the structure of specific type of data they use.

This flexibility leads to use XML in many applications like:

- Channel definition format [CDF]
- Open software description [OSD]
- Resource description framework [RDF]
- Mathematical markup language[MathML]
- Ontology markup language[OML]
- Chemical markup language[CML]

Common Data Format (CDF) – for describing and storing scalar and multidimensional data

Scalable Vector Graphics (SVG) – to describe vector images

Mathematics Markup Language (MathML) – to integrate mathematical notation into a Web document

Chemical Markup Language (CML) - to support chemistry

GPS eXchange Format (GPX) – to describe GPS data

Medical Markup Language (MML) – to represent medical information

Office Open XML (OOXML) – for Microsoft Office

OML – allows a webpage designer to annotate their web pages soυ they can processed with intelligent agent software.

CML – CML was created to provide a way to describe molecularυ information & chemical equations within a single language.

XML APPLICATIONS

- **PUSH Technology**
- **Online Banking**
- **Software Distribution**
- **Web Automation**
- **Database Integration**
- **Scientific Publishing**

University Questions 2018

1. Explain XML document structure?
2. What is the requirement of XML schema? Explain.
3. Explain simple and complex data types in XML with suitable example?
4. How XSLT stylesheets can be used to control page layout in XML?
5. Differentiate between JSON and XML?
6. How can we declare the attributes of an element in DTD?

Difference between XML and JSON?

JSON	XML

1. It is JavaScript Object Notation 2. It is based on JavaScript language. 3. It is a way of representing objects. 4. JSON supports array. 5. It doesn't use end tag. 6. It is less secured. 7. It doesn't supports comments.	1. It is Extensible markup language 2. It is derived from SGML. 3. It is a markup language and uses tag structure to represent data items. 4. XML doesn't support array. 5. It has start and end tags. 6. It is more secured than JSON. 7. It supports comments.