

Introduction to XML

Agenda:

- XML Schemas – provides a more elaborate way to describe the structure of XML documents than DTDs
- Displaying XML Documents – Two different approaches to format XML documents, CSS and XSLT Style Sheets are discussed with examples. Actually XSLT style sheets are used to transform XML document. The target of transformations we describe is XHTML document, which can include CSS style specification for display.
- XML Processors
- Web Services

How computers store and access data?

Two kinds of data files

Binary files - stream of bits, application that created the binary file will interpret it.

Text files – stream of bits grouped together in standardized ways to form numbers which are mapped to characters. Compatible with all text editors but cannot add meta data.

Is it possible to combine Universality of text files + efficiency and rich information storage capabilities of binary files?

Standard Generalized Markup Language – is a text based lang that can be used to mark up data --that is add meta data in a self describing way.

eXtensible Markup Language (XML)

Introduction

- SGML is a meta-markup language is a language for defining markup language it can describe a wide variety of document types.
- Developed in the early 1980s; In 1986 SGML was approved by ISO std.
- HTML was developed using SGML in the early 1990s - specifically for Web documents.
- Two problems with HTML:
 1. HTML is defined to describe the general form and layout of information without considering its meaning.
 2. Fixed set of tags and attributes. Given tags must fit every kind of document. No way to find particular information
 3. There are no restrictions on arrangement or order of tag appearance in 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 to the first problems 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.
- Use SGML to define a new markup language to meet those needs
- 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 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
- XML is a meta-markup language that provides framework for defining specialized markup languages
- Instead of creating text file like

Nandini sidnal Or

```
<html>
<head><title>name</title></head>.....
```

Create an XML like

```
<name>
<first> nandini </first>
<last> sidnal </last>
</name>
```

XML is much larger than text file but makes easier to write software that accesses the information by giving structure to data.

- XML is a very simple and universal way of storing and transferring any textual kind

- XML does not predefine any tags
 - XML tag and its content, together with closing tag ◊ element
- XML has no hidden specifications
- XML based markup language as ◊ tag set
- Document that uses XML based markup language ◊ XML document
- An XML processor is a program that parses XML documents and provides the parts to an application
- Both IE7 and FX2 support basic XML

XML is a meta language for describing mark-up languages. It provides a facility to define tags and the structural relationship between them

What is XML?

- XML stands for EXtensible Markup Language
- XML is a markup language much like HTML
- XML was designed to carry data, not to display data
- XML tags are not predefined. You must define your own tags
- XML is designed to be self-descriptive
- XML is a W3C Recommendation

XML is not a replacement for HTML.

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.

XML document can include several different kinds of statements.

- Data elements
- Markup declarations - instructions to XML parser

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

The most common of these are the data elements of the document. XML document may also include markup declarations, which are instructions to the XML parser, and processing instructions, which are instructions for an application program that will process the data described in the document.

- 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. It is a first line of the XHTML document.
- Comments are same as HTML. `<!-- This is a comment -->`
- XML names are used to name elements and attributes.
 - 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.

- White-space is Preserved in XML

HTML truncates multiple white-space characters to one single white-space:

HTML: Hello my name is Tove

Output: Hello my name is Tove.

With XML, the white-space in a document is not truncated.

- Every XML document defines a single root element, whose opening tag must appear on the first line of XML code.
- All other elements must be nested inside the root element. The root element of every XHTML document is html.

- All XML Elements Must Have a Closing Tag

`<element_name/>` --- no content

- In HTML, you will often see elements that don't have a closing tag:

`<p>`This is a paragraph
`<p>`This is another paragraph.

- In XML, it is illegal to omit the closing tag. All elements **must** have a closing tag:

`<p>`This is a paragraph`</p>`
`<p>`This is another paragraph`</p>`

- Opening and closing tags must be written with the same case:

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

- XML Elements Must be Properly Nested
 - In HTML, you might see improperly nested elements:


```
<b><i>This text is bold and italic</b></i>
```
 - In XML, all elements **must** be properly nested within each other:


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

In the example above, "Properly nested" simply means that since the `<i>` element is opened inside the `` element, it must be closed inside the `` element.
- XML Documents Must Have a Root Element
- XML documents must 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.

An XML document that follows all of these rules is well formed

Example :

```
<?xml version = "1.0" encoding = "utf-8"?>
<ad>
  <year>1960</year>
  <make>Cessna</make>
  <model>Centurian</moel>
  <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 XHTML-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,
 - when tags might need to grow in structural complexity in the future
 - if the data is subdata of the parent element's content
 - if the data has substructure of its own
- Attribute is used ,
 - For identifying numbers/names of element
 - If data in question is one value from a given possibilities
 - The attribute is used if there is no substructure

<!-- 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.

XML Document Structure:

XML document often uses two auxiliary files:

1. It specifies tag set and syntactic structural rules.
2. It contain the style sheet to describe how the content of the document to be printed.

XML documents (and HTML documents) are made up by the following building blocks:

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

Elements

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

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

Tags

Tags are used to markup elements.

A starting tag like `<element_name>` mark up the beginning of an element, and an ending tag like `</element_name>` mark up the end of an element.

Examples:

A body element: `<body>body text in between</body>`.

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

Attributes

Attributes provide extra information about elements.

Attributes are placed inside the start tag of an element. Attributes come in name/value pairs. The following "img" element has an additional information about a source file:

`` The name of the element is "img". The name of the attribute is "src". The value of the attribute is "computer.gif". Since the element itself is empty it is closed by a `/`.

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.

CDATA

CDATA also means character data.

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.

Most of you know the HTML entity reference: " " that is used to insert an extra space in an HTML document. Entities are expanded when a document is parsed by an XML parser.

- An XML document often uses two auxiliary files:
 - One to specify the structural syntactic rules (DTD / XML schema)
 - One to provide a style specification (CSS /XSLT Style Sheets)

Entities

An XML document has a single root element, but often consists of one or more entities. An XML document consists 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
- * All other entities are referenced in the document entity

Reasons to break a document into multiple entities.

1. Good to define a Large documents as a smaller no. of parts 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

(XML is all text!)

When the XML parser encounters the name of a non-binary entity in a doc, it replaces the name with the value it references. Bin entities are not parsed by XML parser but by application that deal with the document. Eg. Browsers.

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 has the form name with prepended ampersand and appended semicolon: &entity_name; Eg. &apple_image;
 - One common use of entities is for special characters that may be used for markup delimiters to appear themselves in the document.

These are predefined (as in XHTML):

Character	Entity	Meaning
<	<	Less than
>	>	Greater than
&	&	Ampersand
“	"	Double quote
‘	'	Single quote

- 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 [.

- Content of Character data section is not parsed by parser

For example 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>

Document Type Definitions:

A DTD is a set of structural rules called declarations. These rules specify a set of elements, along with how and where they can appear in a document

- Purpose: provide a standard form for a collection of XML documents and define a markup language for them.
- DTD provide entity definition.
- With DTD, application development would be simpler.
- Not all XML documents have or need a DTD
- External style sheets are used to impose a uniform style over a collection of documents.

- When are DTDs used?
When same tag set definition are used by collection of documents , collection of users and documents must have a consistent and uniform structure.
- A document can be tested against a DTD to determine weather it confirms to the rules the DTD describes.
- Application programs that processes the data in the collection of XML documents can be written to assume the particular document form.
- Without such structural restrictions, developing such applications would be difficult. If not impossible.
- 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 with incorrect/inappropriate declaration can have wide-spread consequences.
- DTD declarations have the form:

<!keyword ... >

There are four possible declaration keywords:

ELEMENT, ATTLIST, ENTITY, and NOTATION

1. Declaring Elements:

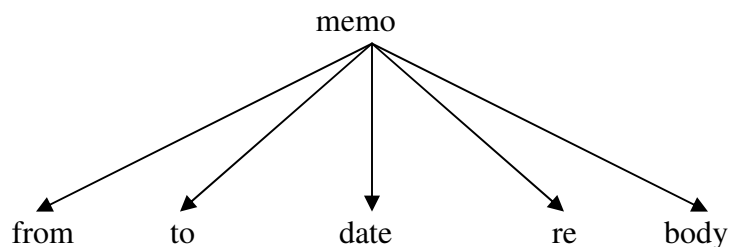
- Element declarations are similar to BNF(CFG)(used to define syntactic structure of Programming language) here DTD describes syntactic structure of particular set of doc so its rules are similar to BNF.
- An element declaration specifies the name of an element and its structure
- If the element is a leaf node of the document tree, its structure is in terms of characters
- If it is an internal node, its structure is a list of children elements (either leaf or internal nodes)
- General form:

<!ELEMENT element_name(list of child names)>

e.g.,

<!ELEMENT memo (from, to, date, re, body)>

This element structure can describe the document tree structure shown below.



- Child elements can have modifiers,
 - + -> One or more occurrences
 - * -> Zero or more occurrences
 - ? -> Zero or one occurrences

Ex: consider below DTD declaration

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

- One or more parent elements
- One age element
- Possible a spouse element.
- Zero or more sibling element.

- Leaf nodes specify data types of content of their parent nodes which are elements
 1. PCDATA (parsable character data)
 2. EMPTY (no content)
 3. ANY (can have any content)

Example of a leaf declaration:

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

2. Declaring Attributes:

- Attributes are declared separately from the element declarations
- General form:

```
<!ATTLIST element_name attribute_name attribute_type [default_value]>
```

More than one attribute

```
<!ATTLIST element_name attribute_name1 attribute_type default_value_1
attribute_name 2 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
 - #Fixed value - value ,which every element have and can't be changed
 - # 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

Example :

```
<!ATTLIST car doors CDATA "4">
<!ATTLIST car engine_type CDATA #REQUIRED>
<!ATTLIST car make CDATA #FIXED "Ford">
<!ATTLIST car price CDATA #IMPLIED>
```

```
<car doors = "2" engine_type = "V8">
...
</car>
```

Declaring Entities :

Two kinds:

- A general entity can be referenced anywhere in the content of an XML document
Ex: Predefined entities are all general entities.
- A parameter entity can be referenced only in DTD.
- General Form of entity declaration.

```
<!ENTITY [%] entity_name "entity_value">
```

% when present it specifies declaration parameter entity

Example :

```
<!ENTITY jfk "John Fitzgerald Kennedy">
```

✓ A reference above declared entity: &jfk;

- If the entity value is longer than a line, define it in a separate file (an external text entity)
✓ General Form of external entity declaration

```
<!ENTITY entity_name SYSTEM "file_location">
```

SYSTEM specifies that the definition of the entity is in a different file.

- Example for parameter entity
 <!ENTITY %pat "(USN, Name)">
 <!ELEMENT student %pat; >

4. Sample DTD:

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

```
<!-- planes.dtd - a document type definition for the planes.xml document, which specifies  

a list of used airplanes for sale -->
```

```
<!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">
```

5. Internal and External DTD's:

- Internal DTDs

```
<!DOCTYPE planes [
```

```
    <!-- The DTD for planes -->
```

```
]>
```

- External DTDs

```
<!DOCTYPE XML_doc_root_name SYSTEM
    "DTD_file_name">
```

For examples,

```
<!DOCTYPE planes_for_sale SYSTEM
    "planes.dtd">
```

```
<?xml version = "1.0" encoding = "utf-8"?>
<!-- planes.xml - A document that lists ads for used airplanes -->
<!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>
```

7.5 Namespaces:

- XML provides benefits over bit formats and can now create well formed XML doc. But when applications become complex we need to combine elements from various doc types into one XML doc. This causes Pb?
- Two documents will have elements with same name but with different meanings and semantics.
- Namespaces – a means by which you can differentiate elements and attributes of different XML document types from each other when combining them together into other documents , or even when processing multiple documents simultaneously.
- Why do we need Namespaces?
- XML allows users to create their tags, **naming collisions** can occur For ex: element title
 - <title>Resume of John</title>
 - <title>Sir</title><Fname>John</Fname>
- Namespaces provide a means for user to prevent **naming collisions** Element title
 - <xhtml:title>Resume of John</xhtml:title> –
 - <person:title>Sir</person:title><Fname>John</Fname>
 - xhtml and person --- two namespaces
- Namespace is collection of elements and attribute names used in XML documents. It has a form of a URI
- A Namespace for elements and attributes of the hierarchy rooted at particular element is declared as the value of the attribute xmlns. Namespace declaration for an element is

<element_name xmlns[:prefix] = URI>

 - 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 URI // URI is too long to be typed on every occurrence of every name from the namespace.
 2. URI may includes characters that are illegal in XML
- Usually the element for which a namespace is declared is usually the root of a document.

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

This declares the default namespace, from which names appear without prefixes.

As an example of a prefixed namespace declaration, consider

```
<birds xmlns:bd = http://www.audubon.org/names/species>
```

Within the birds element, including all of its children elements, the names from the namespace must be prefixed with bd, as in the following.

```
<bd:lark>
```

If an element has more than one namespace declaration, then

```
<birds xmlns:bd = "http://www.audubon.org/names/species">
```

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

Here we have added the standard XHTML namespace to the birds element.

One namespace declaration in an element can be used to declare a default namespace. This can be done by not including the prefix in the declaration. The names from the default by omitting the prefix.

Consider the example in which two namespaces are declared.

The first is declared to be the default namespaces.

The second defines the prefix, cap.

```
<states>
  xmlns = http://www.states-info.org/states
  xmlns:cap = http://www.states-info.org/state-capitals
  <state>
    <name> South Dakota </name>
    <population> 75689 </population>
    <capital>
      <cap:name> Pierre </cap:name>
      <cap:population>12429</cap:population>
    </capital>
  </state>
</states>
```

Each state element has name and population elements for both namespaces.

7.6 XML Schemas:

- A schema is any type of model document that defines the structure of something, such as databases structure or documents. Here something is XML doc. Actually DTDs are a type of schema.
- An XML schema is an XML document so it can be parsed with an XML parser.
- The term XML scheme is used to refer to specify W3C XML schema technology.
- W3C XML Schemas like DTD allow you to describe the structure for an XML doc.

- DTDs have several disadvantages
 - Syntax is different from XML - cannot be parsed with an XML parser
 - It is confusing to deal with two different syntactic forms
 - DTDs do not allow restriction on the form of data that can be content of element

ex: `<quantity>5</quantity>` and `<quantity>5</quantity>` are valid

DTD can only specifies that could be anything.

Eg time

No datatype for integers all are treated as texts.

- XML Schemas is one of the alternatives to DTD
 - It is XML document, so it can be parsed with XML parser
 - It also provides far more control over data types than do DTDs
 - User can define new types with constraints on existing data types

1. Schema Fundamentals:

- Schema are related idea of class and an object in an OOP language
 - ✓ Schema \diamond class definition
 - ✓ XML document confirming to schema structure \diamond Object
- Schemas have two primary purposes
 - ✓ Specify the structure of its instance XML documents
 - ✓ specify the data type of every element & attribute of its instance XML documents

2. Defining a schema:

Schemas are written from a namespace(schema of schemas):

<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 namespace 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 assigning 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, we must set the elementFormDefault attribute to qualified. elementFormDefault = qualified.

- The default namespace which is source of the unprefix names in the schema is given with another xmlns specification

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

- A complete example of 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" >

Defining a schema instance:

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

1. Define the default namespace

```
<planes
  xmlns = http://cs.uccs.edu/planesScema
  ...>
```

2. It is root element of an instance document is for the schemaLocation attribute. Specify the standard namespace for instances (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" >
```

4. Schema Data types:

Two categories of data types

1. Simple (strings only, no attributes and no nested elements)

2. Complex (can have attributes and nested elements)

- XML Schema defines 44 data types
- Primitive: string, Boolean, float, ...
- Derived: byte, decimal, positiveInteger, ...
- User-defined (derived) data types – specify constraints on an existing type (then called as base type)
- Constraints are given in terms of facets of the base type

Ex: interget data type has *8 facets :totalDigits, maxInclusive....

- 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. Defining a simple type:

- 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

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

- An element can have constant value, using fixed attribute

```
<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
- 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>
```

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

6. Declaring Complex Types:

- There are several categories of complex types, but we discuss just one, element-only elements
- Element-only elements are defined with the complex Type element
- Use the sequence tag for nested elements that must be in a particular order
- 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:string" />
</xsd:sequence>
</xsd:complexType>

```

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 formatted.

Output of xsv when run on planes.xml

```

<?XML version = '1.0' encoding = 'utf-8?>
<xsv docElt = '{ http://cs.uccs.edu/planesSchema } planes'
instanceAccessed = 'true'
instanceErrors = '0'
schemaErrors = '0'
schemaLocs = 'http://cs.uccs.edu/planesSchema->planes.xsd'
Target = 'file: /c:/wbook2/xml/planes.xml'
Validation = 'strict'
Version = 'XSV 1.197/1.101 of 2001/07/07 12:01:19'
Xmlns='http:// www.w3.org/2000/05.xsv'>

```

```

<importAttempt URI = 'file:/c:/wbook2/xml/planes.xsd'
namespace = 'http://cs.uccs.edu/planesSchema'
outcome = 'success' />
</xsv>

```

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

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 doc.
- Without a style sheet that defines presentation styles for the doc tags the XML doc can not be displayed in a formatted manner.
- Some browsers like FX2 have default style sheets that are used when style sheets are not defined.
- Eg of planes.xml document.

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 xml-stylesheet processing instruction
- Display– 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

```
<!-- planes.css - a style sheet for the planes.xml document -->
ad { display: block; margin-top: 15px; color: blue;}
year, make, model { color: red; font-size: 16pt;}
color {display: block; margin-left: 20px; font-size: 12pt;}
description {display: block; margin-left: 20px; font-size: 12pt;}
seller { display: block; margin-left: 15px; font-size: 14pt;}
location {display: block; margin-left: 40px; }
city {font-size: 12pt;}
state {font-size: 12pt;}
```

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

<!-- planes.xml - A document that lists ads for used airplanes -->
<planes_for_sale>
  <ad>
    <year> 1977 </year>
    <make> Cessana </make>
    <model> Skyhawk </model>
    <color> Light blue and white </color>
    <description> New interior
  </description>
  <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:

1977 Cessana Skyhawk

Light blue and white

New interior

Skyway Aircraft

Rapid City, South Dakota

XSLT Style Sheets:

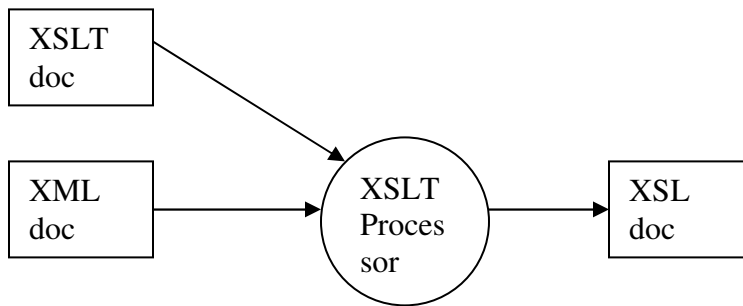
eXtensible Stylesheet Language (XSL) is recommendations for defining XML document transformations and presentations

- Split into three parts:
 - XSLT – transforms XML documents into forms & formats
 - XPath – XML Path Lan. language for expressions , identifies parts of XML documents
 - XSL-FO – Formatting objects

- XSLT style sheets are used to transform XML documents into different forms or formats. ie. to transform XML doc into XHTML doc primarily for display. During transformation the content can be modified, sorted, converted to attribute values among other things.
- XPath is a language for expressions, which are often used to identify parts of XML documents, such as specific elements that are in specific positions in the document or elements that have particular attribute values.
- XSLT requires such expressions to specify transformations.
- XPath is also used for XML document querying languages, such as XQL for building new XML document structures using Xpointer.

Overview of XSLT:

- XSLT is a functional-style programming language.
- XSLT includes functions, parameters, names to which values can be bound , selection constructs and conditional expressions for multiple selection.
- The syntactic structure of XSLT is XML, so each stmt is specified with an element.
- XSLT document is a program to be executed and XML doc is input data to that program.
- Parts of XML document are selected possibly modified and merged with other parts of XSLT document to form new doc called XSL document which could be again input to an XSLT processor. o/p can be stored for future use by applications like browsers.
- XSLT documents consists of templates which use XPath to describe element/attribute patterns in the input XML document
- Each template has XSLT code which is executed when a match to the template is found in the XML document. So each template describes a function, which is executed whenever the XSLT processor finds a match to the templates pattern.
- XSLT processor sequentially examines the i/p XML documents searching for parts that match one of the templates in XSLT doc.
- XML doc consists of nodes where nodes are elements, attributes, comments, text and processing instructions.
- If the template matches an element, the element is not processed until the closing tag is found.
- When a template matches an element, the child elements of that element may or may not be processed.
- XSLT has two models of processing XML data
 1. *Template –Driven model* ◇ data consists of multiple instances of regular data collections files of records.
 2. *Data – Driven model* ◇ irregular and recursive data



XSL Transformations for presentation:

- Will discuss simplest of formatting specifications for the smallest units of information. XSLT includes more than 50 formatting object types and more than 230 properties.
- XSLT for presentations (XML document \diamond XHTML document)

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

-Specifies the XSLT processor that style sheet to be used for XML document

Eg.

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

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

```
<plane>
```

```
  <year> 1977 </year>
```

```
  <make> Cessana </make>
```

```
  <model> Skyhawk </model>
```

```
  <color> Light blue and white </color>
```

```
</plane>
```

This specifies xslplane.xml as its XSLT style sheet.

- XSLT style sheet is XML document
 - Stylesheet root element of XSLT style sheet
 - It defines namespaces as attributes and collections of elements that defines its transformations

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

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

Prefix for XSLT elements is xsl and default namespace is that for XHTML

- A style sheet doc must include at least one template element.

- The template opening tag includes a match attribute to specify an XPath expression to select a node in XML doc.
- The content of template element specifies what is to be placed in the output document.
- If a template element is thought of as a subprogram, the opening tag states where the subprogram is to be applied and the content of the element specifies the body of the subprogram.
- Template is included to match root node of XML document using Xpath expression as in the following:

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

- Template is included to match specific node of XML document

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

- If XPath expression begins with / the address is absolute otherwise it is relative to the current node of the xml doc.
- The template for root node is implicitly applied and others in xslt doc must be applied explicitly to the xml doc.
- The apply-templates applies appropriate templates to descendant nodes of current node with select attribute applies templates to specific descendant nodes otherwise it applies to all descends.
- Template elements include two distinct kinds of elements:
 - Those that literally contain content and
 - Those that specify content to be copied from the associated XML doc.

Eg. ``

Happy Easter!

``

- All XSLT elements that represent XHTML elements are copied by XSLT processor to the output document being generated. All XHTML elements that appear in an XSLT document must conform to the syntactic restrictions that apply to XML.
- Content of an element of XML document is to be copied to o/p doc is specified with value-of element
- The value-of element with select attribute specify the element of XML document whose content are to be copied.

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

- This implies that content of the AUTHOUR element of the XML doc is to be copied to output document.

- Select attribute can specify any node of XML doc ---this is adv of XSLT over CSS.

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

```

<!-- xslplane1.xsl  An XSLT stylesheet for xslplane.xml using child templates -->
<xsl:stylesheet version = "1.0"
    xmlns:xsl = "http://www.w3.org/1999/XSL/Transform"
    xmlns = "http://www.w3.org/1999/xhtml">
  < - - template for whole document. - - >
  <xsl:template match = "plane">
    <html><head><title> Style sheet for xslplane.xml </title>
    </head><body>
    <h2> Airplane Description </h2>
    <!-- Apply the matching templates to the elements in plane - - >
    <xsl:apply-templates />
    </body></html>
  </xsl:template>
< - - the templates to be applied to the elements  in the plane element - - >
<xsl:template match = "year">
  <span style = "font-style: italic; color: blue;"> Year;
  </span>
  <xsl:value-of select = "." /> <br />
</xsl:template>

```

XSLT Style Sheets

```

<xsl:template match = "make">
  <span style = "font-style: italic; color: blue;"> Make:
  </span>
  <xsl:value-of select = "." /> <br />
</xsl:template>
<xsl:template match = "model">
  <span style = "font-style: italic; color: blue;"> Model:
  </span>
  <xsl:value-of select = "." /> <br />
</xsl:template>
<xsl:template match = "color">
  <span style = "font-style: italic; color: blue;"> Color:

```

```

</span>
<xsl:value-of select = "." /> <br />
</xsl:template>
</xsl:stylesheet>

```

More general and complex. No need to include templates for all of the child nodes of plane, because the select clause of the value-of element finds them.

Airplane Description

Year: 1977

Make : Cessna

Model : Skyhawk

Color : Light blue and white

- If xml document includes a collection of data elements with the same structure then XSLT template used for one plane can be used repeatedly with the for-each element, which uses a select attribute to specify an element in XML data.
- The value of the select attribute is a pattern, which is a path expression that specifies an element. Any child elements of the specified element are included.

```

<?xml version = "1.0" encoding = "utf-8"?>
<!-- xslplanes.xml -->
<?xml-stylesheet type = "text/xsl" href = "xslplanes.xsl" ?>
<planes>
  <plane>
    <year> 1977 </year>
    <make> Cessna </make>
    <model> Skyhawk </model>
    <color> Light blue and white </color>
  </plane>
  <plane>
    <year> 1975 </year>

```

```

<make> Piper </make>
<model> Apache </model>
<color> White </color>
</plane>

```

XSLT Style Sheets:

```

<?xml version = "1.0" encoding = "utf-8"?>
<!-- xslplanes.xsl -->
<xsl:stylesheet version = "1.0"
    xmlns:xsl = "http://www.w3.org/1999/XSL/Transform"
    xmlns = "http://www.w3.org/1999/xhtml" >
<xsl:template match = "planes">
    <h2> Airplane Descriptions </h2>
<xsl:for-each select = "plane">
    <span style = "font-style: italic"> Year: </span>
<xsl:value-of select = "year" /> <br />
    <span style = "font-style: italic"> Make: </span>
<xsl:value-of select = "make" /> <br />
    <span style = "font-style: italic"> Model: </span>
<xsl:value-of select = "model" /> <br />
    <span style = "font-style: italic"> Color: </span>
<xsl:value-of select = "color" /> <br /> <br />
</xsl:for-each>
</xsl:template>
</xsl:stylesheet>

```

Airplane Description

Year: 1977

Make : Cessna

Model : Skyhawk

Color : Light blue and white

Year: 1975

Make : Piper

Model : Apache

Color : White

XSLT provides a simple way to sort the elements of the XML document before sending them or their content to the output documents. This is done with sort element, which can take several attributes.

- Select attribute specifies the node that is to be used for key of sort.
- Data-type attribute is used to specify whether the key is to be sorted as text or numeric.
- By default sort is ascending but order attribute can set it to descending.

```
<xsl:sort select = "year" data-type = "number" />
```

XSLT Style Sheets

XSLT provides a simple way to sort the elements of the XML document before sending them or their content to the output documents. This is done with sort element, which can take several attributes.

- Select attribute specifies the node that is to be used for key of sort.
- Data-type attribute is used to specify whether the key is to be sorted as text or numeric.
- By default sort is ascending but order attribute can set it to descending.

```
<xsl:sort select = "year" data-type = "number" />
```

XML Processors:

➤ Purposes

1. Check the syntax of a document for well-formedness
2. Replace all references to entities by their definitions
3. Copy default values (from DTDs or schemas) into the document
4. If a DTD or schema is specified and the processor includes a validating parser, the structure of the document is validated

➤ Two ways to check well-formedness:

1. A browser with an XML parser
2. A stand-alone XML parser
3. Xerces-J

✓ Downloadable from <http://xml.apache.org/xerces-j/index.html>

- ✓ It is Java ,works with JVM1.1 or later
- ✓ Class - sax.SAXCount
- Should expose the data of XML document in a standard API
- There are two different approaches to designing XML processors 1.SAX and 2.DOM approach
- The **SAX** (Simple API for XML) Approach:
 - Widely accepted and supported
 - Based on the concept of event processing:
 - Every time a syntactic structure (e.g., a tag, an attribute, etc.) is recognized, the processor raises an event
- The application defines event handlers to respond to the syntactic structures
 - The **DOM** Approach
 - Builds a DOM tree (hierarchical syntactic structure) of the document
 - When the tree is complete, it can accessed in a number of ways including various tree traversals and random accesses.

Advantages of DOM approach:

1. Good if any part of the document must be accessed more than once
2. If any rearrangement of the document must be done, it is facilitated by having a representation of the whole document in memory
3. Random access to any part of the document is possible
4. Because the whole document is parsed before any processing takes place, processing of an invalid document is avoided

Disadvantages of the DOM approach:

1. Large documents require a large memory
 2. The DOM approach is slower
- Note: Most DOM processors use a SAX front end

Web Services:

- The ultimate goal of Web services:

Allow different software in different places, written in different languages and resident on different platforms, to connect and interoperate
- The Web began as provider of markup documents, served through the HTTP methods, GET and POST
- A Web service is closely related to an information service

- The server provides services, through server- resident software
- The same Web server can provide both documents and services

- The original Web services were provided via Remote Procedure Call (RPC), through two technologies, DCOM and CORBA.

DCOM and CORBA use different protocols, which defeats the goal of universal component interoperability
- There are three roles required to provide and use Web services:
 1. Service providers
 2. Service requestors
 3. A service registry
- **Service providers**
 - Must develop & deploy software that provide service
 - Service Description --> Web Serviced Definition Language (WSDL)
 - *Used to describe available services, as well as of message protocols for their use their use
 - * Such descriptions reside on the Web server
- **Service requestors**
 - Uses WSDL to query a query a web registry
- **Service registry**
 - Created using Universal Description, Discovery, and Integration Service (UDDI)
 - * UDDI also provides
 - Create service registry
 - Provides method to query a Web service registry
- **Standard Object Access Protocol (SOAP)**
 - An XML-based specification that defines the forms of messages and RPCs
 - The root element of SOAP is envelope
 - Envelope contains SOAP messages – description of web services
 - Supports the exchange of information among distributed systems