XML - eXtensible Markup Language

- XML stands for eXtensible Markup Language
- XML is a markup language much like HTML
- XML was designed to store and transport data
- XML was designed to be self-descriptive
- XML is a W3C Recommendation

SYNTAX OF XML

- XML imposes two distinct levels of syntax:
  - There is a general low level syntax that is appreciable on all XML documents
  - The other syntactic level is specified by DTD (Document Type Definition) or XML schemas.
- The DTDs and XML schemas specify a set of tag and attribute that can appear in a particular document or collection of documents.
- They also specify the order of occurrence in the document.
- The XML documents consists of data elements which form the statements of XML document.
- The XML document might also consists of markup declaration, which act as instructions to the XML parser
- All XML documents begin with an XML declaration. This declaration identifies that the document isa XML document and also specifies version number of XML standard.
- It also specifies encoding standard.
  <?xml version = "1.0" encoding = "utf-8"?>
- Comments in XML is similar to HTML
- XML names are used to name elements and attributes.
- XML names are case-sensitive.
- There is no limitation on the length of the names.
- All XML document contains a single root element whose opening tag appears on first line of the code
- All other tags must be nested inside the root element
- As in case of XHTML, XML tags can also have attributes
- The values for the attributes must be in single or double quotation

Example:

1. **<?xml version = "1.0" encoding = "utf-8"?>**
   <student>
   <name>Santhosh B S</name>
   <usn>1DA18CS090</usn>
   </student>

2. Tags with attributes
   The above code can be also written as

   <student name = "Santhosh B S" usn = "1DA18CS090">
   </student>

XML DOCUMENT STRUCTURE

- An XML document often consists of 2 files:
  - One of the document – that specifies its tag set
  - The other specifies the structural syntactic role and one that contains a style sheet to describe how content of the document is to be printed
- The structural roles are given as either a DTD or an XML schema
- An XML document consists of logically related collection of information known as entities
- The *document entity* is the physical file that represent the document itself
- The document is normally divided into multiple entities.
- One of the advantage dividing document into multiple entities is managing the document becomes simple
- If the same data appears in more than one place, defining it as an entity allows number of references to a single copy of the data
- Many documents include information that cannot be represented as text. Ex: images
- Such information units are stored as binary data
- These binary data must be a separate unit to be able to include in XML document
- These entities are called as *Binary entities*
- When an XML processor encounters the name of a non-binary entity in a document, it replaces the name with value it references
- Binary entities can be handled only by browsers
- XML processor or parsers can only deal with text
- Entity names can be of any length. They must begin with a letter, dash or a colon
- A reference to an entity is its name with a prepended ampersand and an appended semicolon
- Example: if **stud_name** is the name of entity, **&stud_name;** is a reference to it
- One of the use of entities is to allow characters used as markup delimiters to appears as themselves
- The entity references are normally placed in CDATA section
- Syntax: **<! [CDATA[** *content* **]] >**
- For example, instead of

  The last word of the line is >>> here <<<.
  the following could be used:
  <![CDATA[The last word of the line is >>> here <<<]]>

DOCUMENT TYPE DEFINITIONS

- A DTD is a set of structural rules called declarations which specify a set of elements that can appear in the document. It also specifies how and where these elements appear
- DTD also specify entity definitions
- DTD is more useful when the same tag set definition is used by collection of documents
- A DTD can be embedded in XML document whose syntax rules it describes
- In this case, a DTD is called as *internal DTD* or a separate file can be created which can be linked to XML file. In this case the DTD is called as *External DTD*
- An external DTD can be used with more than one XML file
- Syntactically, a DTD is a sequence of declarations. Each declaration has the form of markup declaration
- Example: <!keyword...>
- Four possible keywords can be used in a declaration:
  - ELEMENT, used to define tags;
  - ATTLIST, used to define tag attributes;
  - ENTITY, used to define entities; and
  - NOTATION, used to define data type notations.

DECLARING ELEMENTS

- DTD follows rules of context-free grammar for element declaration
- A DTD describes the syntactic structure of a particular set of documents
- Each element declaration in a DTD specifies the structure of one category of elements
- An element is a node in such a tree either a leaf node or an internal node
- If element is leaf node, its syntactic description is its character pattern
- If the element is internal node, its syntactic description is a list of its child element
- The form of an element declaration for elements that contain elements is as follows:

  <!ELEMENT *element_name* (*list of names of child elements*)>

- For example, consider the following declaration:
  <!ELEMENT memo (from, to, date, re, body)>
- This element declaration would describe the document tree structure shown in Figure 7.1.

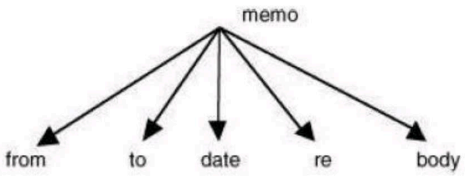


Figure 7.1 An example of the document tree structure for an element definition

- In many cases, it is necessary to specify the number of times that a child element may appear. This can be done in a DTD declaration by adding a modifier to the child element specification. These modifiers, described in Table 7.1, are borrowed from regular expressions.
- Any child element specification can be followed by one of the modifiers.

Modifier	Meaning
+	One or more occurrences
*	Zero or more occurrences
?	Zero or one occurrence

- Consider the following DTD declaration:

  <!ELEMENT person (parent+, age, spouse?, sibling*)>
- In this example, a person element is specified to have the following child elements: one or more parent elements, one age element, possibly a spouse element, and zero or more sibling elements.
- The leaf nodes of a DTD specify the data types of the content of their parent nodes, which are elements.
- In most cases, the content of an element is type PCDATA, for parsable character data. Parsable character data is a string of any printable characters except “less than” (<), “greater than” (>), and the ampersand (&).
- Two other content types can be specified: EMPTY and ANY.
- The EMPTY type specifies that the element has no content; it is used for elements similar to the XHTML img element.
- The ANY type is used when the element may contain literally any content.
- The form of a leaf element declaration is as follows:

  <!ELEMENT *element_name* (#PCDATA)>

DECLARING ATTRIBUTES

The attributes of an element are declared separately from the element declaration in a DTD. An attribute declaration must include the name of the element to which the attribute belongs, the attribute's name, its type, and a default option. The general form of an attribute declaration is as follows:

<!ATTLIST *element_name attribute_name attribute type default_option*>

If more than one attribute is declared for a given element, the declarations can be combined, as in the following element:

```
<!ATTLIST element_name
          attribute name_1  attribute type  default_option_1
          attribute_name_2  attribute_type  default_option_2
          ...
          attribute_name_n  attribute_type  default_option_n
>
```

The default option in an attribute declaration can specify either an actual value or a requirement for the value of the attribute in the XML document.

Table 7.2 Possible default options for attributes

Option	Meaning
A value	The quoted value, which is used if none is specified in an element
`#FIXED` value	The quoted value, which every element will have and which cannot be changed
`#REQUIRED`	No default value is given; every instance of the element must specify a value
`#IMPLIED`	No default value is given (the browser chooses the default value); the value may or may not be specified in an element

For example, suppose the DTD included the following attribute specifications:

```
<!ATTLIST airplane places CDATA "4">
<!ATTLIST airplane engine_type CDATA #REQUIRED>
<!ATTLIST airplane price CDATA #IMPLIED>
<!ATTLIST airplane manufacturer CDATA #FIXED "Cessna">
```

Then the following XML element would be valid for this DTD:
<airplane places = "10" engine_type = "jet"> </airplane>

DECLARING ENTITIES

- Entities can be defined so that they can be referenced anywhere in the content of an XML document, in which case they are called general entities. The predefined entities are all general entities.
- Entities can also be defined so that they can be referenced only in DTDs, in which case they are called parameter entities.
- The form of an entity declaration is

  <!ENTITY [%] *entity_name* "*entity_value*">
- When the optional percent sign (%) is present in an entity declaration, it specifies that the entity is a parameter entity rather than a general entity.
- Example: <!ENTITY sbs "Santhosh B Suresh">
- When an entity is longer than a few words, its text is defined outside the DTD. In such cases, the entity is called an external text entity. The form of the declaration of an external text entity is

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

A 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">
```

- Some XML parsers check documents that have DTDs in order to ensure that the documents conform to the structure specified in the DTDs. These parsers are called validating parsers.
- If an XML document specifies a DTD and is parsed by a validating XML parser, and the parser determines that the document conforms to the DTD, the document is called valid.
- Handwritten XML documents often are not well formed, which means that they do not follow XML's syntactic rules.
- Any errors they contain are detected by all XML parsers, which must report them.
- XML parsers are not allowed to either repair or ignore errors.
- Validating XML parsers detect and report all inconsistencies in documents relative to their DTDs.

INTERNAL AND EXTERNAL DTDs

Internal DTD Example:

```
<?xml version = "1.0" encoding = "utf-8"?>
      <!DOCTYPE planes [
          <!-- The DTD for planes -->
      ]>
<!-- The planes XML document -->
```

External DTD Example:

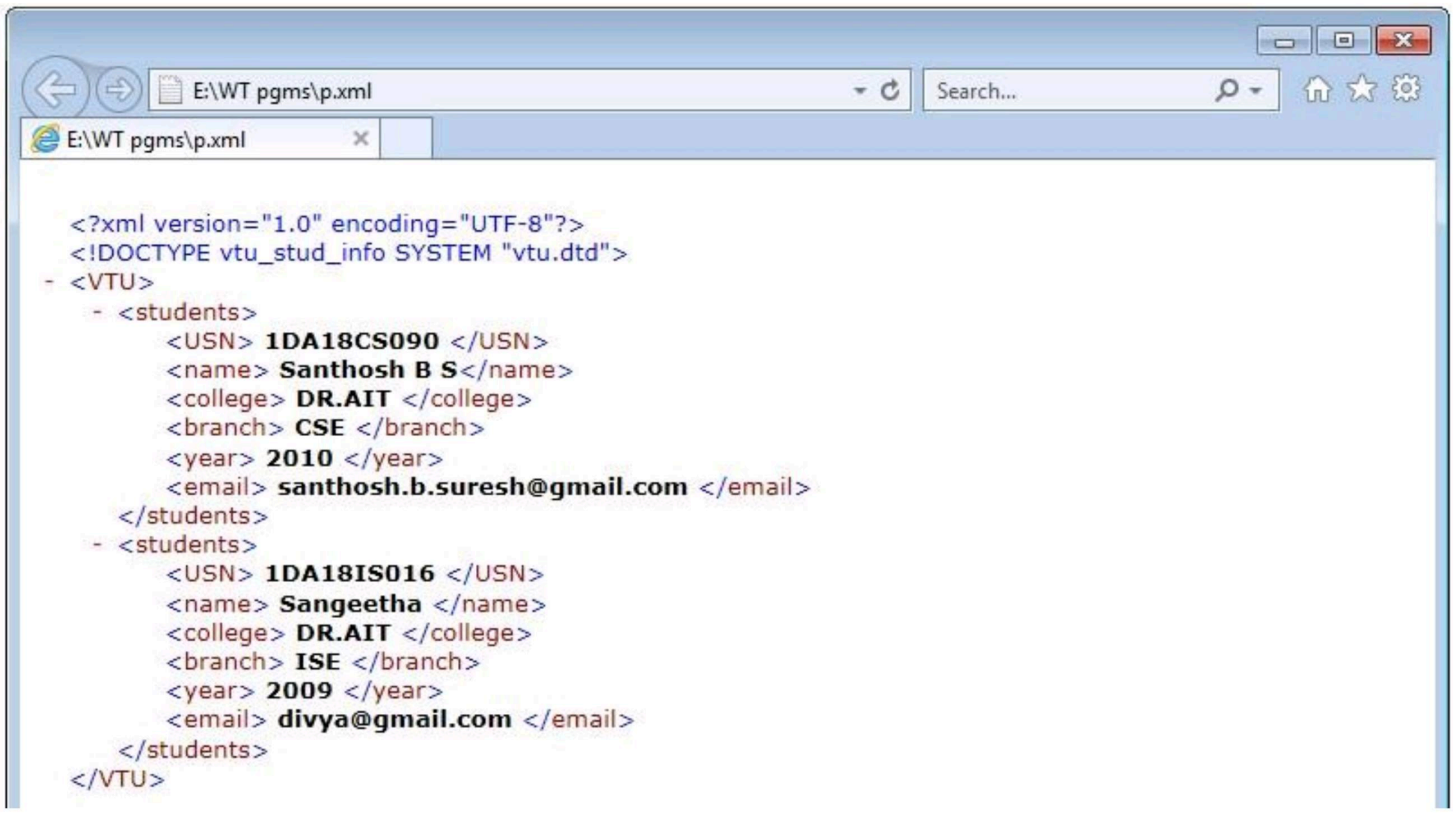
[assuming that the DTD is stored in the file named *planes.dtd*]

<!DOCTYPE planes_for_sale SYSTEM "planes.dtd">

//sampleDTD.xml

```
<?xml version = "1.0" encoding = "utf-8"?>
<!DOCTYPE vtu_stud_info SYSTEM "vtu.dtd">
<VTU>
<students>
        <USN> 1DA18CS090 </USN>
        <name> Santhosh B S</name>
        <college> DR.AIT </college>
        <branch> CSE </branch>
        <year> 2018</year>
        <email> santhosh.b.suresh@gmail.com </email>
</students>
```

```
<students>
        <USN> 1DA18IS016 </USN>
         <name> Sangeetha </name>
        <college> DR.AIT </college>
        <branch> ISE </branch>
        <year> 2018 </year>
        <email> sangeetha@gmail.com </email>
</students>
</VTU>
```



NAMESPACES

- One problem with using different markup vocabularies in the same document is that collisions between names that are defined in two or more of those tag sets could result.
- An example of this situation is having a <table> tag for a category of furniture and a <table> tag from XHTML for information tables.
- Clearly, software systems that process XML documents must be capable of unambiguously recognizing the element names in those documents.
- To deal with this problem, the W3C has developed a standard for XML namespaces (at http://www.w3.org/TR/REC-xml-names).
- An XML namespace is a collection of element and attribute names used in XML documents. The name of a namespace usually has the form of a uniform resource identifier (URI).
- A namespace for the elements and attributes of the hierarchy rooted at a particular element is declared as the value of the attribute xmlns.

- The form of a namespace declaration for an element is

<*element_name* xmlns[:*prefix*] = URI>

- The square brackets indicate that what is within them is optional. The prefix, if included, is the name that must be attached to the names in the declared namespace.
- If the prefix is not included, the namespace is the default for the document.

- A prefix is used for two reasons. First, most URIs are too long to be typed on every occurrence of every name from the namespace. Second, a URI includes characters that are invalid in XML.
- Note that the element for which a namespace is declared is usually the root of a document.
- For ex: all XHTML documents in this notes declare the xmlns namespace on the root element, html:

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

- This declaration defines the default namespace for XHTML documents, which is http://www.w3.org/1999/xhtml.
- The next example declares two namespaces. The first is declared to be the default namespace; 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> 754844 </population>
    <capital>
      <cap:name> Pierre </cap:name>
      <cap:population> 12429 </cap:population>
    </capital>
  </state>
  <!-- More states -->
</states>
```

XML SCHEMAS

XML schemas is similar to DTD i.e. schemas are used to define the structure of the document
DTDs had several disadvantages:

- The syntax of the DTD was un-related to XML, therefore they cannot be analysed with an XML processor
- It was very difficult for the programmers to deal with 2 different types of syntaxes
- DTDs does not support the datatype of content of the tag. All of them are specified as text

Hence, schemas were introduced

SCHEMA FUNDAMENTALS

- Schemas can be considered as a class in object oriented programming
- A XML document that conforms to the standard or to the structure of the schema is similar to an object
- The XML schemas have 2 primary purposes.
  - They are used to specify the structure of its instance of XML document, including which elements and attributes may appear in instance document. It also specifies where and how often the elements may appear
  - The schema specifies the datatype of every element and attributes of XML
- The XML schemas are ***namespace-centric***

DEFINING A SCHEMA

Schemas themselves are written with the use of a collection of tags, or a vocabulary, from a namespace that is, in effect, a schema of schemas. The name of this namespace is http://www.w3.org/2001/XMLSchema.

- Every schema has schema as its root element. This namespace specification appears as follows:

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

- The name of the namespace defined by a schema must be specified with the targetNamespace attribute of the schema element.

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

- If the elements and attributes that are not defined directly in the schema element are to be included in the target namespace, schema's elementFormDefault must be set to qualified, as follows:

elementFormDefault = "qualified"

- The default namespace, which is the source of the unprefixed names in the schema, is given with another xmlns specification, but this time without the prefix:

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

Example in 2 alternate methods of defining a schema

```
<xsd:schema
<!-- The namespace for the schema itself (prefix is xsd) -->
  xmlns:xsd = http://www.w3.org/2001/XMLSchema
<!-- The namespace where elements defined here will be placed -->
  targetNamespace = http://cs.uccs.edu/planeSchema
<!-- The default namespace for this document (no prefix) -->
  xmlns = http://cs.uccs.edu/planeSchema
<!-- We want to put non-top-level elements in the target namespace -->
  elementFormDefault = "qualified">
```

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

The above is an 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.

DEFINING A SCHEMA INSTANCE

- An instance document normally defines its default namespace to be the one defined in its schema. For example, if the root element is planes, we could have

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

- 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.

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

- Third, the instance document must specify the filename of the schema in which the default namespace is defined. This is accomplished with the schemaLocation attribute, which takes two values: the namespace of the schema and the filename of the schema.

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

- Combining everything, we get,

```
<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">
```

AN OVERVIEW OF DATA TYPES

There are two categories of user-defined schema data types: simple and complex.

- A simple data type is a data type whose content is restricted to strings. A simple type cannot have attributes or include nested elements.
- A complex type can have attributes and include other data types as child elements.

Data declarations in an XML schema can be either local or global.

- A local declaration is a declaration that appears inside an element that is a child of the schema element.
- A global declaration is a declaration that appears as a child of the schema element. Global elements are visible in the whole schema in which they are declared.

SIMPLE TYPES

- Elements are defined in an XML schema with the element tag.

<xsd:element name = “engine” type = “xsd:string” />

- An instance of the schema in which the engine element is defined could have the following element:

<engine> inline six cylinder fuel injected </engine>

- An element can be given a default value with the default attribute:

```
<xsd:element name = "engine"   type = "xsd:string"
             default = "fuel injected V-6"  />
```

- Constant values are given with the fixed attribute, as in the following example:

```
<xsd:element name = "plane"   type = "xsd:string"
             fixed = "single wing"  />
```

- A simple user-defined data type is described in a simpleType element with the use of facets.
- Facets must be specified in the content of a restriction element, which gives the base type name.
- The facets themselves are given in elements named for the facets: the value attribute specifies the value of the facet.

```
<xsd:simpleType name = "firstName">
  <xsd:restriction base = "xsd:string">
    <xsd:maxLength value = "10" />
  </xsd:restriction>
</xsd:simpleType>
```

COMPLEX TYPES

Complex types are defined with the complexType tag. The elements that are the content of an element-only element must be contained in an ordered group, an unordered group, a choice, or a named group. The sequence element is used to contain an ordered group of elements. Example:

```
<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>
```

A complex type whose elements are an unordered group is defined in an all element. Elements in all and sequence groups can include the minOccurs and maxOccurs attributes to specify the numbers of occurrences. Example:

<?xml version = “1.0” encoding = “utf-8”?>

```
<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>
```

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>
```

For example, the year element could be defined as follows:

```
<xsd:element name = "year">
  <xsd:simpleType>
    <xsd:restriction base = "xsd:decimal">
      <xsd:minInclusive value = "1900" />
      <xsd:maxInclusive value = "2007" />
    </xsd:restriction>
  </xsd:simpleType>
</xsd:element>
```

With the year element defined globally, the sports_car element can be defined with a reference to the year with the ref attribute:

```
<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 ref = "year" />
  </xsd:sequence>
</xsd:complexType>
```

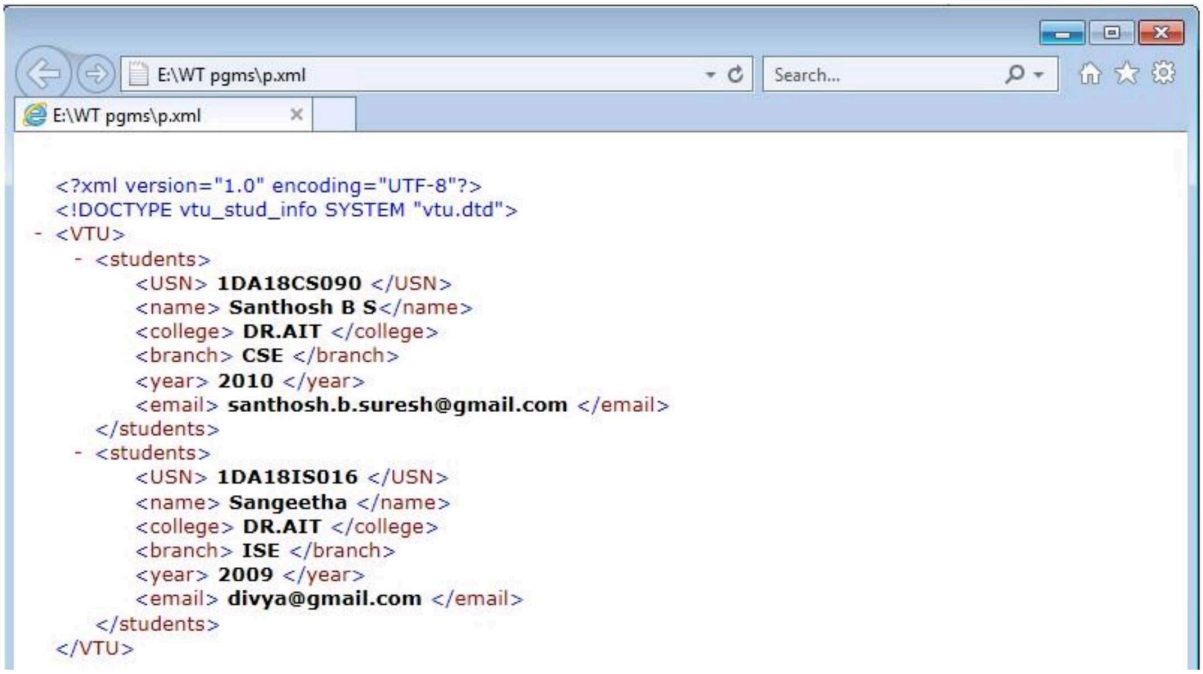
VALIDATING INSTANCES OF SCHEMAS

XSV is an abbreviation for ***XML Schema Validator***. If the schema and the instance document are available on the Web, xsv can be used online, like the XHTML validation tool at the W3C Web site. This tool can also be downloaded and run on any computer. The Web site for xsv is http://www.w3.org/XML/Schema#XSV.
The output of xsv is an XML document. When the tool is run from the command line, the output document appears on the screen with no formatting, so it is a bit difficult to read. The following is the output of xsv run on planes.xml:

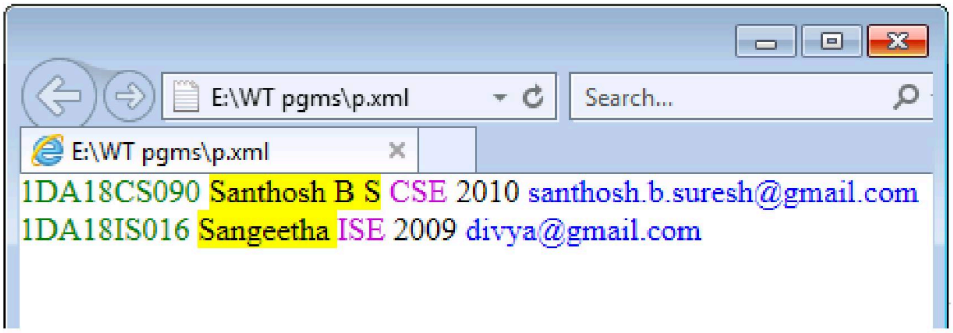
```
<?XML version='1.0' encoding = 'utf-8'?>
<xsv docElt='{http://cs.uccs.edu/planeSchema}planes'
     instanceAssessed='true'
     instanceErrors = '0'
     rootType='[Anonymous]'
     schemaErrors='0'
     schemaLocs='http://cs.uccs.edu/planeSchema -> planes.xsd'
     target='file:/c:/wbook2/xml/planes.xml'
     validation='strict'
     version='XSV 1.197/1.101 of 2001/07/07 12:10:19'
     xmlns='http://www.w3.org/2000/05/xsv' >

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

DISPLAYING RAW XML DOCUMENTS

If an XML document is displayed without a style sheet that defines presentation styles for the document's tags, the displayed document will not have formatted content.



DISPLAYING XML DOCUMENTS WITH CSS



//6a.xml

```
<?xml version = "1.0" encoding = "utf-8"?>
<?xml-stylesheet type = "text/css" href = "6a.css"?>

<VTU>
<students>
<USN> 1DA18CS090 </USN>
<name> Santhosh B S</name>
<college> DR.AIT </college>
<branch> CSE </branch>
<year> 2010 </year>
<email> santhosh.b.suresh@gmail.com </email>
</students>
<students>
<USN> 1DA18IS016 </USN>
<name> Sangeetha </name>
<college> DR.AIT </college>
<branch> ISE </branch>
<year> 2009 </year>
<email> divya@gmail.com </email>
</students>
</VTU>
```

//6a.css

```
students
{ clear: both; float : left;}
USN
{color: green; }
name
{background: yellow;}
college
{ display: none;}
branch
{color : #cd00dc; text-align: right;}
YOJ
{background : red; color : white;}
email
{ color: blue;}
```

XSLT STYLE SHEETS

- The eXtensible Stylesheet Language (XSL) is a family of recommendations for defining the presentation and transformations of XML documents.
- It consists of three related standards:
  - XSL Transformations (XSLT),
  - XML Path Language (XPath), and
  - XSL Formatting Objects (XSL-FO).
- XSLT style sheets are used to transform XML documents into different forms or formats, perhaps using different DTDs.
- One common use for XSLT is to transform XML documents into XHTML documents, primarily for display. In the transformation of an XML document, the content of elements can be moved, modified, sorted, and converted to attribute values, among other things.
- XSLT style sheets are XML documents, so they can be validated against DTDs.
- They can even be transformed with the use of other XSLT style sheets.
- The XSLT standard is given at http://www.w3.org/TR/xslt.
- 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.

- XPath is also used for XML document querying languages, such as XQL, and to build new XML document structures with XPointer. The XPath standard is given at http://www.w3.org/TR/xpath.

OVERVIEW OF XSLT

- XSLT is actually a simple functional-style programming language.
- Included in XSLT are functions, parameters, names to which values can be bound, selection constructs, and conditional expressions for multiple selection.
- XSLT processors take both an XML document and an XSLT document as input. T
- he 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.
- The transformation process used by an XSLT processor is shown in Figure 7.5.

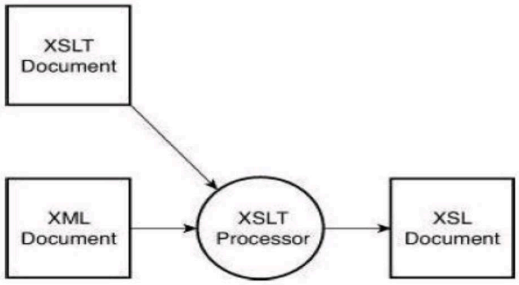


Figure 7.5 XSLT processing

- An XSLT document consists primarily of one or more templates.
- Each template describes a function that is executed whenever the XSLT processor finds a match to the template's pattern.
- One XSLT model of processing XML data is called the template-driven model, which works well when the data consists of multiple instances of highly regular data collections, as with files containing records.
- XSLT can also deal with irregular and recursive data, using template fragments in what is called the data-driven model.
- A single XSLT style sheet can include the mechanisms for both the template- and data-driven models.

XSL TRANSFORMATIONS FOR PRESENTATION

Consider a sample program:

//6b.xml

```
<?xml version="1.0" encoding="utf-8"?>
<?xml-stylesheet type="text/xsl" href="6b.xsl"?>
```

```
<?xml version="1.0" encoding="utf-8"?>
<?xml-stylesheet type="text/xsl" href="6b.xsl"?>

<vtu>
<student>
<name>Santhosh B S</name>
<usn>1DA10CS090</usn>
<collegeName>DR.AIT</collegeName>
<branch>CSE</branch>
<year>2010</year>
<email> santhosh.b.suresh@gmail.com </email>
</student>
<student>
<name>Akash Bangera</name>
<usn>1DA10CS003</usn>
<collegeName>DR.AIT</collegeName>
```

```
<branch>CSE</branch>
<year>2010</year>
<email>akash.bangera@gmail.com</email>
</student>
<student>
<name>Manoj Kumar</name>
<usn>1DA10CS050</usn>
<collegeName>DR.AIT</collegeName>
<branch>CSE</branch>
<year>2010</year>
<email>manoj.kumar@gmail.com</email>
</student>
</vtu>
```

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:

//6b.xsl

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
 <html>
 <body>
 <h2>VTU Student Information</h2>
  <table border="1">
   <tr bgcolor="#99cd32">
    <th>name</th>
    <th>usn</th>
    <th>collegeName</th>
    <th>branch</th>
    <th>year</th>
    <th>email</th>
     </tr>

   <xsl:for-each select="vtu/student">
   <xsl:choose>
    <xsl:when test="name = 'Santhosh B S'">
    <tr bgcolor="yellow">
    <td><xsl:value-of select="name"/></td>
    <td><xsl:value-of select="usn"/></td>
    <td><xsl:value-of select="collegeName"/></td>
    <td><xsl:value-of select="branch"/></td>
    <td><xsl:value-of select="year"/></td>
    <td><xsl:value-of select="email"/></td>
    </tr>
    </xsl:when>
    <xsl:otherwise>
     <tr >
    <td><xsl:value-of select="name"/></td>
    <td><xsl:value-of select="usn"/></td>
    <td><xsl:value-of select="collegeName"/></td>
```

```
        <td><xsl:value-of select="branch"/></td>
        <td><xsl:value-of select="year"/></td>
        <td><xsl:value-of select="email"/></td>
       </tr>
        </xsl:otherwise>
       </xsl:choose>
       </xsl:for-each>
     </table>
  <h2>selected student is highlighted</h2>
   </body>
   </html>
  </xsl:template>
  </xsl:stylesheet>
```

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.

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

In many XSLT documents, a template is included to match the root node of the XML document.

<xsl:template match="/">

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.

<xsl:value-of select="name"/>

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.

XML PROCESSORS

The XML processor takes the XML document and DTD and processes the information so that it may then be used by applications requesting the information. The processor is a software module that reads the XML document to find out the structure and content of the XML document. The structure and content can be derived by the processor because XML documents contain self-explanatory data.

THE PURPOSES OF XML PROCESSORS

- First, the processor must check the basic syntax of the document for well-formedness.
- Second, the processor must replace all references to entities in an XML document with their definitions.
- Third, attributes in DTDs and elements in XML schemas can specify that their values in an XML document have default values, which must be copied into the XML document during processing.
- Fourth, when a DTD or an XML schema is specified and the processor includes a validating parser, the structure of the XML document must be checked to ensure that it is legitimate.

THE SAX APPROACH

- The Simple API for XML (SAX) approach to processing is called event processing.
- The processor scans the XML document from beginning to end.
- Every time a syntactic structure of the document is recognized, the processor signals an event to the application by calling an event handler for the particular structure that was found.
- The syntactic structures of interest naturally include opening tags, attributes, text, and closing tags.
- The interfaces that describe the event handlers form the SAX API.

THE DOM APPROACH

- The Document Object Model (DOM) is an application programming interface (API) for HTML and XML documents.
- It defines the logical structure of documents and the way a document is accessed and manipulated
- Properties of DOM
    - Programmers can build documents, navigate their structure, and add, modify, or delete elements and content.
    - Provides a standard programming interface that can be used in a wide variety of environments and applications.
    - structural isomorphism.
- The DOM representation of an XML document has several advantages over the sequential listing provided by SAX parsers.
- First, it has an obvious advantage if any part of the document must be accessed more than once by the application.
- Second, if the application must perform any rearrangement of the elements of the document, that can most easily be done if the whole document is accessible at the same time.
- Third, accesses to random parts of the document are possible.
- Finally, because the parser sees the whole document before any processing takes place, this approach avoids any processing of a document that is later found to be invalid.

WEB SERVICES

A Web service is a method that resides and is executed on a Web server, but that can be called from any computer on the Web. The standard technologies to support Web services are WSDL, UDDI, SOAP, and XML. **WSDL** - It is used to describe the specific operations provided by the Web service, as well as the protocols for the messages the Web service can send and receive.

UDDI - also provides ways to query a Web services registry to determine what specific services are available.

SOAP - was originally an acronym for Standard Object Access Protocol, designed to describe data objects.

XML - provides a standard way for a group of users to define the structure of their data documents, using a subject-specific mark-up language.