UNIT 4 XML : EXTENSIBLE MARKUP LANGUAGE

Structure **Page Nos.**

4.0 INTRODUCTION

In the previous blocks, we have already learnt about Java Servlets, Enterprise Java Beans and Java Server pages. In this unit, we shall cover some basics aspects of XML, which stands for Extensible Markup language, and SGML. XML is a structured document containing structured information. Structured information contains both content (words, pictures, etc.) and some indication of the role that content plays (for example, content in a section heading has a different meaning from content in a footnote, which means something different than content in a figure caption or content in a database table, etc.). Almost all documents have some structure. SGML, stands for both a language and an ISO standard for describing information, embedded within a document. XML is a meta-language written in SGML that allows one to design a markup language, used for the *easy interchange of documents* on the World Wide Web. In this unit, we shall also learn to discuss the differences between XML, SGML, and DTD, which stands for Document Type Definition and the requirements of DTD for the XML document.

4.1 OBJECTIVES

After going through this unit, you should be able to:

- provide an overview of XML;
- distinguish between XML and HTML;
- define SGML and its use;
- discuss the difference between SGML and XML;
- understand the basic goals of the development of XML;
- define the basic structure of the XML document and its different components;
- define DTD and the method of preparing DTD for an XML document;
- understand what is XML parser and its varieties, and
- understand the different types of entities and their uses.

4.2 AN OVERVIEW OF XML

XML stands for Extensible Markup Language (often written as extensible Markup Language to justify the acronym). A markup language is a specification that adds new information to existing information while keeping two sets of information separate and XML is a set of rules for defining semantic tags that break a document into parts and identifies the different parts of the document. It is a meta-markup language that defines a syntax that is in turn used to define other domain-specific, semantic, structured markup languages. With XML we can store the information in a structured manner.

Comparison of HTML and XML

XML differs from HTML in many aspects. As, we already know, HTML is a markup language used for displaying information, while XML markup is used for describing data of virtually any type. In other words, we can say that HTML deals with how to present whereas XML deals with what to present. Actually, HTML is a markup language whereas XML is a markup language and a language for creating markup languages. HTML limits you to a fixed collection of tags and these tags are primarily used to describe how content will be displayed, such as, making text bold or italiced or headings etc., whereas with XML you can create new or any user defined tags. Hence, XML enables the creation of new markup languages to markup anything imaginable (such as Mathematical formulas, chemical formulas or reactions, music etc.)

Let us, understand the difference between HTML and XML with the help of an example. In HTML a song might be described using a definition title, definition data, an unordered list, and list items. But none of these elements actually have anything to do with music. The HTML might look something like this:

```
<HTML>
<body>
<dt>Indian Classical </dt>
<dd> by HariHaran , Ravi shankar  and Shubha Mudgal</dd>
<ul>
<li>Producer: Rajesh
<li>Publisher: T-Series Records
<li>Length: 6:20
<li>Written: 2002
<li>Artist: Village People
</ul>
<body>
</html>
```

In XML, the same data might be marked up like this:

```
<XML>
<SONG>
<TITLE>Indian Classical</TITLE>
<COMPOSER>Hariharan</COMPOSER>
<COMPOSER>Ravi Shankar</COMPOSER>
<COMPOSER>Shubha Mudgal</COMPOSER>
<PRODUCER>Rajesh</PRODUCER>
<PUBLISHER>T-Series</PUBLISHER>
```

```
<LENGTH>6:20</LENGTH>
<YEAR>2002</YEAR>
<ARTIST>Village People</ARTIST>
</SONG></XML>
```

Instead of generic tags like <dt> and <li>, this listing uses meaningful tags like <SONG>, <TITLE>, <COMPOSER>, and <YEAR>. This has a number of advantages, including the fact that its easier for a human to read the source code to determine what the author intended.

4.3 AN OVERVIEW OF SGML

Now, we shall learn how SGML acts as a base for all the markup languages. SGML is a system or meta-language for defining markup languages, organising and tagging elements of a document. SGML was developed and standardised by the International Organisation for Standards (ISO). SGML itself does not specify any particular formatting; rather, it specifies the rules for tagging elements. These tags can then be interpreted to format elements in different ways. HTML and XML are based on SGML.

Authors mark up their documents by representing structural, presentational, and semantic information alongside content. Each markup language defined in SGML is known as SGML application. An SGML application is generally characterised by:

1) An SGML declaration. The SGML declaration specifies which characters and delimiters may appear in the application. By means of an SGML declaration (XML also has one), the SGML application specifies which characters are to be interpreted as data and which characters are to be interpreted as markup. (They do not have to include the familiar < and > characters; in SGML they could just as easily be {and} instead).

2) A document type definition (DTD). The DTD defines the syntax of markup constructs. It has rules for SGML/XML document just like there is grammar for English. The DTD may include additional definitions such as character entity references. We shall study later in this unit how we can create DTD for an XML document.

3) A specification that describes the semantics to be ascribed to the markup. This specification also imposes syntax restrictions that cannot be expressed within the DTD.

4) Document instances containing data (content) and markup. Each instance contains a reference to the DTD to be used to interpret it.
Using the rules given in the SGML declaration and the results of the information analysis (which ultimately creates something that can easily be considered an information model), the SGML application developer identifies various types of documents—such as reports, brochures, technical manuals, and so on—and develops a DTD for each one. Using the chosen characters, the DTD identifies information objects (elements) and their properties (attributes). The DTD is the very core of an SGML application; how well it is made largely determines the success or failure of the whole activity. Using the information objects (elements), defined in the DTD, the actual information is then marked up using the tags identified for it in the application. If, the development of the DTD has been rushed, it might need continual improvement, modification, or correction. Each time the DTD is changed, the information that has been marked up with it might also need to be modified because it may be incorrect. Very quickly, the quantity of data that needs modification (now called legacy

data) can become a far more serious problem—one that is more costly and time-consuming than the problem that SGML was originally introduced to solve.

Why not SGML?

The SGML on the Web initiative existed a long time before XML was even considered. Somehow, though, it was never really successful. Basically, SGML is just too expensive and complicated for Web use on a large scale. It isn't that it can't be used its just that it won't be used. Using SGML requires too much of an investment in time, tools, and training.

Why do we need XML?

XML adds a list of features that make it far more suitable than either SGML or HTML for use on an increasingly complex and diverse Web:

- **Modularity:** Although HTML appears to have no DTD, there is an implied DTD hard-wired into Web browsers. SGML has a limitless number of DTDs, on the other hand, but there's only one for each type of document. XML enables us to leave out the DTD altogether or, using sophisticated resolution mechanisms, combine multiple fragments of either XML instances or separate DTDs into one compound instance.
- **Extensibility:** XML's powerful linking mechanisms allow you to link to the material without requiring the link target to be physically present in the object. This opens up exciting possibilities for linking together things like material to which you do not have write access, CD-ROMs, library catalogue, the results of database queries, or even non-document media such as sound fragments or parts of videos. Furthermore, it allows you to store the links separately from the objects they link. This makes long-term link maintenance a real possibility.
- **Distribution:** In addition to linking, XML introduces a far more sophisticated method of including link targets in the current instance. This opens the doors to a new world of composite documents—documents composed of fragments of other documents that are automatically (and transparently) assembled to form what is displayed at that particular moment. The content can be instantly tailored to the moment, to the media, and to the reader, and might have only a fleeting existence: a virtual information reality composed of virtual documents.
- **Internationality:** Both HTML and SGML rely heavily on ASCII, which makes using foreign characters very difficult. XML is based on Unicode and requires all XML software to support Unicode as well. Unicode enables XML to handle not just Western-accented characters, but also Asian languages.
- **Data orientation:** XML operates on data orientation rather than readability by humans. Although being humanly readable is one of XML's design goals, electronic commerce requires the data format to be readable by machines as well. XML makes this possible by defining a form of XML that can be more easily created by a machine, but it also adds tighter data control through the more recent XML schema.

4.4 DIFFERENCE BETWEEN SGML AND XML

Now, we shall study what are the basic differences between SGML and XML. SGML, the Standard Generalised Markup Language, is the international standard for defining descriptions of structure and content in electronic documents whereas XML is a simplified version of SGML and it was designed to maintain the most useful parts of SGML.

SGML requires that structured documents reference a Document Type Definition (DTD) to be "valid", XML allows for "well-formed" data and can be delivered with and without a DTD. XML was designed so that SGML can be delivered, as XML, over the Web.

What does XML mean to SGML product vendors? On the technology front, SGML products should be able to read valid XML documents as they sit, as long as they are in 7-bit ASCII. To read internationalised XML documents, (for example in Japanese) SGML software will need modification to handle the ISO standard 10646 character set, and probably also a few industry-but-not-ISO standard encoding such as JIS and Big5. To write XML, SGML products will have to be modified to use the special XML syntax for empty elements.

On the business front, much depends on whether the Web browsers learn XML. If they do, SGML product vendors should brace for a sudden, dramatic demand for products and services from all the technology innovators who are, at the moment, striving to get their own extensions into HTML, and will (correctly) see XML as the way to make this happen. If the browsers remain tied to the fixed set of HTML tags, then XML will simply be an easy on-ramp to SGML, important probably more so because, the spec is short and simple than because of its technical characteristics. This will probably still generate an increase in market size, but not at the insane-seeming rate that would result from the browers' adoption of XML.

Check Your Progress 1

1) State True or False

   a) In HTML, we can define our own tags. T ☐ F ☐

   b) XML deals with what to present on the web page. T ☐ F ☐

   c) With XML one can create new markup languages like math markup language. T ☐ F ☐

2) How does HTML differs from XML? Explain with the help of an example.

   ……...

   ……...

   ……...

3) What are the basic characterstics of SGML?

   ……...

   ……...

   ……...

4) What are the advantages of XML over HTML?

   ……...

   ……...

   ……...

5) Explain the basic differences between XML and SGML.

   ……...

   ……...

   ……...

4.5 XML DEVELOPMENT GOALS

XML was developed by an XML Working Group (originally known as the SGML Editorial Review Board) formed under the auspices of the World Wide Web Consortium (W3C) in 1996.

The design and development goals for XML are:

- XML shall be straightforwardly usable over the Internet. Users must be able to view XML documents as quickly and easily as HTML documents. In practice, this will only be possible when XML browsers are as robust and widely available as HTML browsers, but the principle remains intact.
- XML shall support a wide variety of applications. XML should be beneficial to a wide variety of diverse applications: authoring, browsing, content analysis, etc.
- XML shall be compatible with SGML. Most of the people involved in the XML effort come from organisations that have a large, in some cases staggering, amount of material in SGML. XML was designed pragmatically, to be compatible with existing standards while solving the relatively new problem of sending richly structured documents over the web.
- It shall be easy to write programs, which process XML documents.
- The number of optional features in XML is to be kept to the absolute minimum, ideally zero. Optional features inevitably raise compatibility problems when users want to share documents and sometimes lead to confusion and frustration.
- XML documents should be human-legible and reasonably clear. If you don't have an XML browser and you've received a hunk of XML from somewhere, you ought to be able to look at it in your favourite text editor and actually figure out what the content means.
- The XML design should be prepared quickly. XML was needed immediately and was developed as quickly as possible. It must be possible to create XML documents in other ways: directly in a text editor, with simple shell and Perl scripts, etc
- The design of XML shall be formal and concise.
- XML documents shall be easy to create.
- Terseness in XML markup is of minimal importance. Several SGML language features were designed to minimise the amount of typing required to manually key in SGML documents. These features are not supported in XML. From an abstract point of view, these documents are indistinguishable from their more fully specified forms, but supporting these features adds a considerable burden to the SGML parser (or the person writing it, anyway). In addition, most modern editors offer better facilities to define shortcuts when entering text.

4.6 THE STRUCTURE OF THE XML DOCUMENT

In this section, we shall create a simple XML document and learn about the structure of the XML document. Example 4.1 is about the simplest XML document I can imagine, so start with it. This document can be typed in any convenient text editor, such as Notepad, vi, or emacs.

Example 4.1: Hello XML

```
<?xml version=”1.0”?>
<FOO>
Hello XML!
</FOO>
```

Example-4.1 is not very complicated, but it is a good XML document. To be more precise, it is a well-formed XML document. (“Wellformed” is one of the terms, we shall study about it later)

Saving the XML file

After you have typed in *Example 4.1*, save it in a file called hello.xml or some other name. The three-letter extension .xml is fairly standard. However, do make sure that you save it in plain-text format, and not in the native format of a word processor such as WordPerfect or Microsoft Word.

Loading the XML File into a Web Browser

Now that you’ve created your first XML document, you’re going to want to look at it. The file can be opened directly in a browser that supports XML such as Internet Explorer 5.0. *Figure 1*a shows the result.

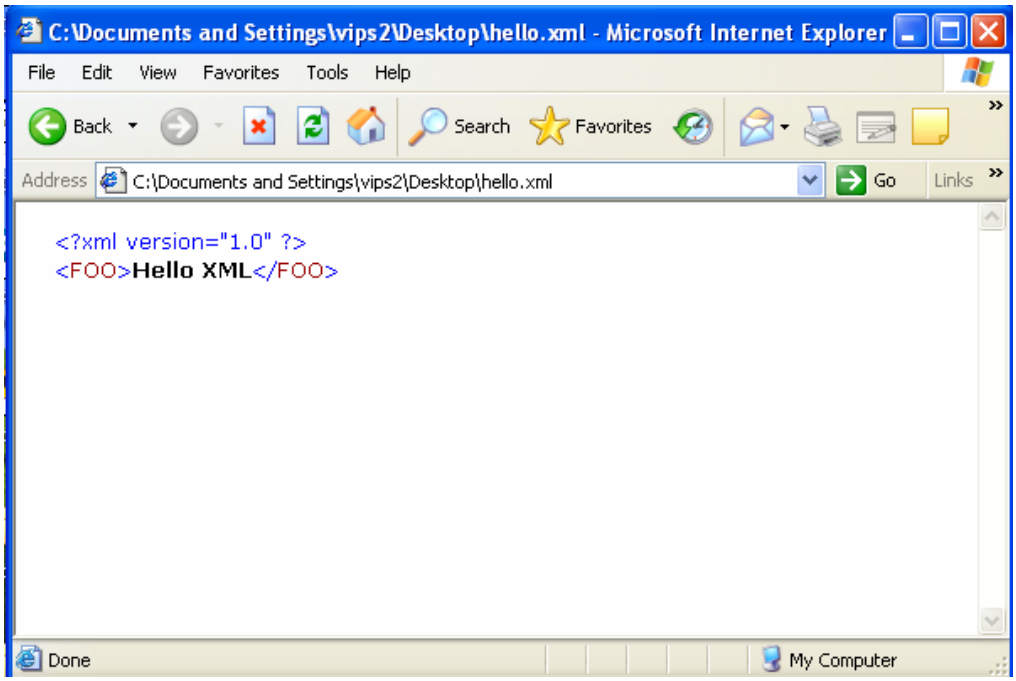

Figure 1: Output of XML Document

XML document may consists of the following parts:

a) The XML Prolog

XML file always starts with a Prolog as shown in the above example. An attribute is a name-value pair separated by an equals sign. Every XML document should begin with an XML declaration that specifies the version of XML in use. (Some XML documents omit this for reasons of backward compatibility, but you should include a version declaration unless you have a specific reason to leave it out). The minimal prolog contains a declaration that identifies the document as an XML document, like this:

<?xml version= “1.0”?>
The declaration may also contain additional information, like this:
<?xml version= “1.0” encoding= “ISO-8859-1” standalone= “yes”?>

The XML declaration may contain the following attributes:
Version Identifies the version of the XML markup language used in the data. This attribute is not optional.

Encoding

Identifies the character set used to encode the data. " ISO-8859-1" is "Latin-1" the Western European and English language character set. (The default is compressed Unicode: UTF-8.).

Standalone

This indicates whether or not this document references an external entity or an external data type specification (see below). If, there are no external references, then "yes" is appropriate.

The first line of the simple XML document in Listing 4.1 is the XML declaration:
<?xml version= "1.0"?>

<?xml version= "1.0" standalone= "no"?>

Identifying the version of XML ensures that future changes to the XML specification will not alter the semantics of this document. The standalone declaration simply makes explicit the fact that this document cannot "stand alone," and that it relies on an external DTD.

b) Elements and Attributes

Each Tag in a XML file can have Element and Attributes. Here's how a Typical Tag looks like,

<Email to= "admin@mydomain.com"
from= "user@mySite.com"
subject= "Introducing XML">
</Email>

In this Example, Email is known as an Element. This Element called E-mail has three attributes, to, from and subject.

The Following Rules need to be followed while declaring the XML Elements Names:

- Names can contain letters, numbers, and other characters
- Names must not start with a number or "_" (underscore)
- Names must not start with the letters xml (or XML or Xml).
- Names cannot contain spaces

Any name can be used, no words are reserved, but the idea is to make names descriptive. Names with an underscore separator are nice.
Examples: <author_name>, <published_date>.

Avoid "-" and "." in names. It could be a mess if your software tried to subtract name from first (author-name) or thought that "name" was a property of the object "author" (author.name).

Element names can be as long as you like, but don't exaggerate. Names should be short and simple, like this: <author_name> not like this: <name_of_the_author>.

XML documents often have a parallel database, where fieldnames are parallel to element names. A good rule is to use the naming rules of your databases.

The ":" should not be used in element names because it is reserved to be used for something called namespaces

In the above example-4.1, the next three lines after the prologue form a FOO element. Separately, <FOO> is a start tag; </FOO> is an end tag; and Hello XML! is the content of the FOO element. Divided another way, the start tag, end tag, and XML declaration are all markup. The text Hello XML! is character data.

c) Empty Tags:

In cases where you don't have to provide any sub tags, you can close the Tag, by providing a "/" to the Closing Tag. For example declaring
<Text></Text> is same a declaring <Text />

d) Comments in XML File:

Comments in XML file are declared the same way as Comments in HTML File.

<Text>Welcome To XML Tutorial </Text>

<!-- This is a comment -->

<Subject />

e) Processing Instructions

An XML file can also contain processing instructions that give commands or information to an application that is processing the XML data. Processing instructions have the following format:

<?target instructions?>

Where the target is the name of the application that is expected to do the processing, and instructions is a string of characters that embodies the information or commands for the application to process.

Well Formed Tags

One of the most important Features of a XML file is, it should be a Well Formed File. What it means that is all the Tags should have a closing tag. In a HTML file, for some tags like
 we don't have to specify a closing tag called </br>. Whereas in a XML file, it is compulsory to have a closing tag. So we have to declare
</br>. These are what are known as Well Formed Tags.

We shall take another *Example 4.2* ignou.xml to understand all the components of the XML document.

Example 4.2

```
<?xml version= "1.0"?>
<?xml version= "1.0" encoding= "ISO-8859-1" standalone= "yes"?>
<!-- ignou.xml-- >
<!-- storing the student details in XML document-- >
<!DOCTYPE ignou SYSTEM ignou.dtd">
<student>
    < course type = "undergraduate">
    <name>
    <fname> Saurabh </fname>
    <lname> Shukla </lname>
    <city> New Delhi </city>
    <state> Delhi </state>
    <zip> 110035 </zip>
    <status id = "P">
</course>
< course type = "Post Graduate">
    <name>
        <fname> Sahil</fname>
```

```
        <lname> Shukla </lname>
    <city> Kanpur </city>
    <state> U.P.</state>
    <zip> 110034 </zip>
    <status id = "D">
</course>
<content> IGNOU is world class open university which is offering undergraduate
and postgraduate course.
</content>
</student>
```

In the above XML document, student is the root element and it contains all other elements (e.g. course, content). Lines preceeding the root element is the prolog which we have discussed above.

<!DOCTYPE ignou SYSTEM ignou.dtd">

The above given line specifies the document type definition for this xml document. Document files define the rules for the XML document. This tag contains three items: the name of the root element to which the DTD is applied, the SYSTEM flag (which denotes an external DTD), and the DTD's name and location. We shall study more about DTD in the next section.

4.7 USING DTD WITH XML DOCUMENT

DTD stands for **document type definition**. A DTD defines the structure of the content of an XML document, thereby allowing you to store data in a consistent format. Document type definition lists the elements, attributes, entities, and notations that can be used in a document, as well as their possible relationships to one another. In brief, we may say that DTD specifies, a set of rules for the structure of a document.

If present, the document type declaration must be the first, in the document after optional processing instructions and comments. The document type declaration identifies the root element of the document and may contain additional declarations. All XML documents must have a single root element that contains all the contents of the document. Additional declarations may come from an external definition (a DTD) or be included directly in the document.

Creating DTD is just like creating a table in a database. In DTDs, you specify the structure of the data by declaring the element to denote the data. You can also specify whether providing a value for the element is mandatory or optional.

Declaring Elements in a DTD

After identifying the elements that can be used for storing structured data, they can be declared in a DTD. The XML document can be checked against the DTD. We can declare Element with the following syntax:

<!Element elementname(content-type or content-model)>

In the above syntax, elementname specifies the name of the element and content-type or content-model specifies whether the element contains textual data or other elements. For e.g. #PCDATA which specifies that element can store parsed character data(i.e. text). An element can be empty, unrestricted or container example. In the above mentioned example ignou.xml, the status is empty element whereas city is the unrestricted element.

Element Type	Description
Empty	Empty elements have no content and are marked up as <empty-elements>
Unrestricted	The opposite of an empty element is an unrestricted element, which can contain any element declared elsewhere in the DTD

In DTD various symbols are used to specify whether an element is mandatory or a optional or number of occurrences. The following is the list of symbols that can be used:

Symbol	Meaning	Example	Description
+	It indicates that there can be at least one or multiple occurrences of the element	Course+	There can multiple occurrences of course element.
*	It indicates that there can be either zero or any number of occurrences of the element	Content *	Any number of content element can be present.
?	It indicates that there can be either zero or exactly one occurrence.	Content ?	Content may not be present or present then only once
|	Or	City | state	City or state

We can declare attribute in DTD using the following syntax:

<!ATTLIST elementname att_name val_type [att_type] [“default”]>

att_name is the name of the attribute and val_type is the type of value which attribute can hold like CDATA defines that this attribute can contains a string.

We can also discuss whether attribute is mandatory or optional. We can do this with the help of att_type which can have the following values:

Attribute Type	Description
REQUIRED	If the attribute of an element is specified as # REQUIRED then, the value of that attribute must be specified if, it is not be specified then, the XML document will be invalid.
FIXED	If attribute of an element is specified as #FIXED then, the value of attribute cannot be changed in the XML document.
IMPLIED	If attribute of an element is specified as #IMPLIED then, attribute is optional i.e. this attribute need not be used every time the associated element is used.

Now, we are in the position to create the DTD named ignou.dtd, as shown in the *Example 4.2*

```
<?XML version= “1.0” rmd= “internal”?>
<!ELEMENT student( course +, content *)>
<!Element course (name, city, state, zip,status)>
<!ATTLIST course type CDATA #IMPLIED>
<!Element name (fname, lname)>
<!Element  fname (#PCDATA)>
```

```
<!Element  lname (#PCDATA)>
<!Element  city (#PCDATA)>
<!Element  state (#PCDATA)>
<!Element  zip (#PCDATA)>
<!Element  content (#PCDATA)>
```

This example, references an external DTD, ignou.dtd, and includes element and attribute declarations for the course element. In this case, course is being given the semantics of a simple link from the XLL specification.

In order to determine if a document is valid, the XML processor must read the entire document type declaration (both internal and external). But for some applications, validity may not be required, and it may be sufficient for the processor to read only the internal declaration. In the example given above, if validity is unimportant and the only reason to read the doctype declaration is to identify the semantics of course, then, reading the external definition is not necessary.

4.8 XML PARSER

An XML parser (or XML processor) is the software that determines the content and structure of an XML document by combining XML document and DTD (if any present). *Figure 2* shows a simple relationship between XML documents, DTDs, parsers and applications. XML parser is the software that reads XML files and makes the information from those files available to applications and other programming languages. The XML parser is responsible for testing whether a document is well-formed and, if, given a DTD or XML schema, whether will also check for validity (i.e., it determines if the document follows the rules of the DTD or schema). Although, there are many XML parsers we shall discuss only Micorosoft's parser used by the Internet explorer and W3C's parser that AMAYA uses.

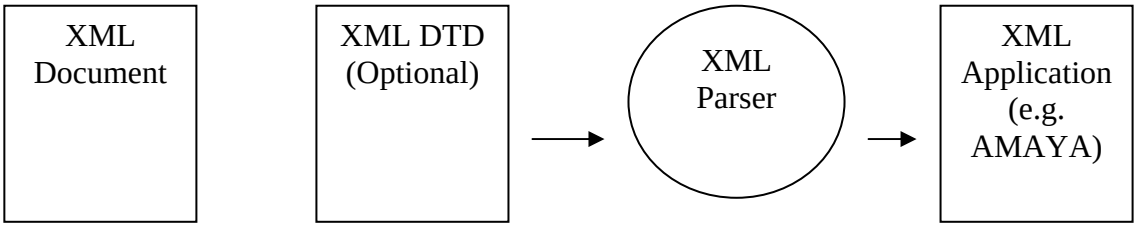

Figure 2: XML Documents and their Corresponding DTDs are Parsed and sent to Application.

XML Parser builds tree structures from XML documents. For example, an XML parser would build the tree structure shown in *Figure 2* for the previously mentioned example ignou.xml. If, this tree structure is created successfully without using a DTD, the XML document is considered **well-formed.** If, the tree stucture is created successfully and DTD is used, the XML document is considered valid. Hence, there can be two types of XML parsers : validating (i.e., enforces DTD rules) and non-validating (i.e., ignores DTD rules).

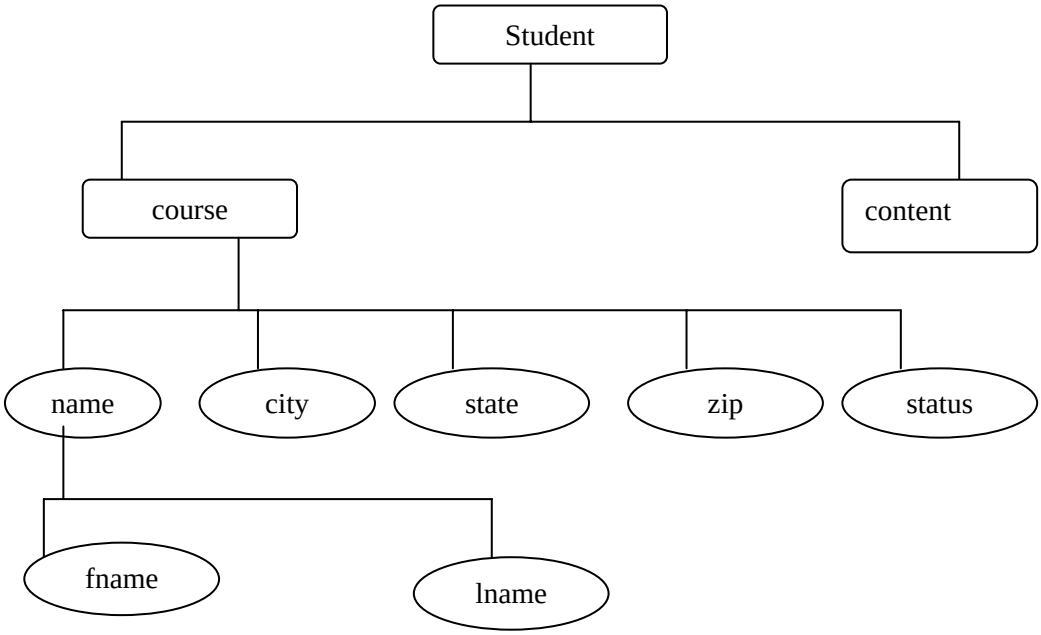

Figure 3: Tree Structure for ignou.xml

4.9 XML ENTITIES

Now, we shall learn about entity. Entities are normally external objects such as graphics files that are meant to be included in the document. Entity declarations [Section 4.3] allow you to associate a name with some other fragment of the document. That construct can be a chunk of regular text, a chunk of the document type declaration, or a reference to an external file containing either text or binary data.There are three varities of entities in XML.

Here are a few typical entity declarations:

<!ENTITY BCA “Bachelor of Computer Application.”>

<!ENTITY course SYSTEM “/standard/course1.xml”>

External Entity and Internal Entity : The first entity in the example given above is an internal entity because the replacement text is stored in the declaration. Using &BCA; anywhere in the document insert “Bachelor of computer Application.” at that location. Internal entities allow you to define shortcuts for frequently typed text or text that is expected to change, such as the revision status of a document. You must declare an internal entity before you can use it. It can save you a lot of unnecessary typing.

The declaration of an internal entity has this form:

<!ENTITY name “replacement text”>

Now, everytime the string &ampname; appears in your XML code, the XML processor will automatically replace it with the replacement text (which can be just as long as you like).

The XML specification predefines five internal entities:

- < produces the left angle bracket, <
- > produces the right angle bracket, >
- & produces the ampersand, &

- ' produces a single quote character (an apostrophe),
- " produces a double quote character,

External Entities: The second example id external entity. Using &course; will have - insert the contents of the file /standard/legalnotice.xml at that location in the document when it is processed. The XML processor will parse the content of that file as if its content were typed at the location of the entity reference. To reference external entities, you must have a DTD for your XML document.

External entities allow an XML document to refer to an external file. External entities contain either text or binary data. If they contain text, the content of the external file is inserted at the point of reference and parsed as part of the referring document. Binary data is not parsed and may only be referenced in an attribute. Binary data is used to reference figures and other non-XML content in the document.

Parameter Entities

Parameter entities can only occur in the document type declaration. A parameter entity is identified by placing "% " (percent-space) in front of its name in the declaration. The percent sign is also used in references to parameter entities, instead of the ampersand. Parameter entity references are immediately expanded in the document type declaration and their replacement text is part of the declaration, whereas normal entity references are not expanded.

Check Your Progress 2

1) Explain the structure of an XML document.

……...

……..

……..

2) Explain the different development goals of XML document.

……...

……...

……...

3) What is DTD? Why do we use it? Explain the different components of DTD.

……...

……...

……...

4) Differentiate between validating and non-validating parser.

……...

……...

……...……

5) Explain the need/use of entities in XML document. Describe all three types of entities with the help of an example.

……...

……

……...

6) Where would you declare entities?

……...

……..

……..

7) Why is an XML declaration needed?

……...

……

……...

8) Can entities in attribute values as well as in content be used?

……...

……

……...

9) Is the use of binary data permitted in a CDATA section?

……...

……

……...

10) How many elements should you have in a DTD?

……...……

…….……

……...11) Is it necessary to validate XML documents?

……...

……

……...

12) How can we check whether the DTD is correct?

……...

……

……...

13) How can you put unparsed (non-XML) text in an external entity?

……...

……

……...

4.10 SUMMARY

XML, which stands for Extensible Markup Language, is defined as a structured document containing structured information. Structured information contains both content (words, pictures, etc.) and some indication of what role that content plays. XML is a meta-language written in SGML that allows one to design a markup

language, used to allow for the easy interchange of documents on the World Wide Web. XML is a set of rules for defining semantic tags that break a document into parts and identifies the different parts of the document

XML differs from HTML in many aspects. As we know, HTML is a markup language and is used for displaying information, while XML markup is used for describing data of virtually any type. XML can be used to create other markup languages whereas HTML cannot be used for that purpose.

SGML acts as a base for all the markup languages. SGML is a system or meta-language for defining markup languages, organising and tagging elements of a document. SGML was developed and standardised by the International Organisation for Standards (ISO). SGML application is generally characterised by SGML declaration, Document Type Definition, specification and document instance.

XML is more suitable than SGML or HTML and is increasingly used for web application because of features like, modularity, extensibility distribution and data orientation. XML differs from SGML in some ways such as, SGML, is the international standard for defining descriptions of structure and content in electronic documents whereas XML is a simplified version of SGML and it was designed to maintain the most useful parts of SGML.

XML was developed by an XML Working Group with few design and development goals. Some of them like XML shall be straightforwardly usable over the Internet, XML shall be compatible with SGML, the number of optional features in XML is to be kept to the absolute minimum, ideally zero. XML documents should be human-legible and reasonably clear etc. These two qualities are of utmost importance and have contributed to the success of XML.

XML document may consist of many parts like XML prolog, elements, attributes, empty tags, comments and processing instructions. XML document should begin with an XML declaration, which is known as an XML prolog that specifies the version of XML in use. There can be two types of elements in XML container elements and empty elements. An XML file can also contain processing instructions that give commands or information to an application that is processing the XML data.

DTD stands for document type definition, it defines the structure of the content of an XML document, thereby allowing you to store data in a consistent format. Document type definition lists the elements, attributes, entities, and notations that can be used in a document, as well as their possible relationships with one another. The document type declaration identifies the root element of the document and may contain additional declarations. In DTDs, the structure of the data can be specified by declaring the element as denoting the data.

An XML parser (or XML processor) is the software that determines the content and structure of an XML document by combining XML document and DTD. The XML parser is responsible for testing whether a document is well-formed and, if given a DTD or XML schema, it also checks for the validity. XML Parser builds the tree structure from XML documents. If this tree structure is created successfully without using a DTD, the XML document is considered well-formed and if the tree structure is created successfully and DTD is used, the XML document is considered valid. Entities are normally external objects such as graphics files that are meant to be included in the document. Entity declarations allow you to associate a name with some other fragment of the document. There are three varities of entities in XML. Internal Entity, where the replacement text is stored in the declaration. Internal entities allow you to define shortcuts for frequently typed text. External entities allow

an XML document to refer to an external file. External entities contain either text or binary data. Parameter entities can only occur in the document type declaration. A parameter entity is identified by placing "% " (percent-space) in front of its name in the declaration

4.11 SOLUTIONS/ ANSWERS

Check Your Progress 1

1) True/ False

   a) False

   b) True

   c) True

2) XML differs from HTML in many aspects. As we know, HTML is a markup language that is used for displaying information, while XML markup is used for describing data of virtually any type. HTML deals with How to present whereas XML deals with what to present. XML can be used for creating other markup languages whereas HTML cannot be used for creating other markup languages.

   In HTML a song might be described using a definition title, definition data, an unordered list, and list items. But none of these elements actually have anything to do with music. The HTML example:

```
<HTML>
<body>
<dt>Indian Classical </dt>
<dd> by HariHaran , Ravi shankar  and Shubha Mudgal</dd>
<ul>
<li>Producer: Rajesh
<li>Publisher: T-Series Records
<li>Length: 6:20
<li>Written: 2002
<li>Artist: Village People
</ul>
<body>
</html>
```

In XML the same data might be marked up like this:

```
<XML>
<SONG>
<TITLE>Indian Classical</TITLE>
<COMPOSER>Hariharan</COMPOSER>
<COMPOSER>Ravi Shankar</COMPOSER>
<COMPOSER>Shubha Mudgal</COMPOSER>
<PRODUCER>Rajesh</PRODUCER>
<PUBLISHER>T-Series</PUBLISHER>
<LENGTH>6:20</LENGTH>
<YEAR>2002</YEAR>
<ARTIST>Village People</ARTIST>
</SONG></XML>
```

Instead of generic tags like <dt> and <li>, this listing uses meaningful tags like <SONG>, <TITLE>, <COMPOSER>, and <YEAR>.

3) SGML stands for Standard Generalised Markup Language, SGML is a system or meta-language for defining markup languages, organising and tagging elements of a document. Each markup language defined in SGML is known an SGML application. An SGML application is generally characterised by the following:

   a) **An SGML Declaration:** The SGML declaration specifies which characters and delimiters may appear in the application. By means of an SGML declaration (XML also has one), the SGML application specifies which characters are to be interpreted as data and which characters are to be interpreted as markup.

   b) **A Document Type Definition (DTD):** The DTD defines the syntax of markup constructs. It has rules for SGML/XML document just as there are rules of grammar for English. The DTD may include additional definitions such as character entity references.

   c) **A Specification:** This describes the semantics to be ascribed to the markup. This specification also imposes syntax restrictions that cannot be expressed within the DTD.

   d) **Document Instances:** Contain data (content) and markup. Each instance contains a reference to the DTD to be used to interpret it.

4) The advantages of XML over HTML and SGML are the following:

   - ***Modularity :*** Although HTML appears to have no DTD, there is an implied DTD hard-wired into Web browsers. XML enables us to leave out the DTD altogether or, using sophisticated resolution mechanisms, combine multiple fragments of either XML instances or separate DTDs into one compound instance.

   - ***Extensibility:*** XML's powerful linking mechanisms allow you to link to material without requiring the link target to be physically present in the object. This helps in linking together things like material to which you do not have write access, CD-ROMs, library catalogue, the results of database queries etc.

   - ***Distribution :***XML introduces a far more sophisticated method of including link targets in the current instance. This opens the doors to a new world of composite documents—documents composed of fragments of other documents that are automatically (and transparently) assembled to form what is displayed at that particular moment. The content can be instantly tailored to the moment, to the media, and to the reader.

   - ***Internationality:*** Both HTML and SGML rely heavily on ASCII, which makes using foreign characters very difficult. XML is based on Unicode and requires all XML software to support Unicode as well. Unicode enables XML to handle not just Western-accented characters, but also Asian languages.

   - ***Data orientation:*** XML operates on data orientation rather than readability by humans. Although being humanly readable is one of XML's design goals, electronic commerce requires the data format to be readable by machines as well.

5) XML differs from SGML in many respects. SGML, is the international standard for defining descriptions of structure and content in electronic documents whereas XML is a simplified version of SGML and it was designed to maintain the most useful parts of SGML. SGML requires that structured documents reference a Document Type Definition (DTD) to be “valid”, XML allows for “well-formed” data and can be delivered with and without a DTD.

Check Your Progress 2

1) XML document may consists of the following parts:

a) The XML Prolog

XML file always starts with a Prolog. Every XML document should begin with an XML declaration that specifies the version of XML in use. The minimal prolog contains a declaration that identifies the document as an XML document, like this:

<?xml version="1.0"?>
The declaration may also contain additional information, like this:
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>

b) Elements and Attributes:

Each Tag in a XML file can have Element and Attributes. Here’s what a Typical Tag looks like,

<Email to="admin@mydomain.com"
from="user@mySite.com"
subject="Introducing XML">
</Email>

In this Example, Email is called an Element. This Element called E-mail has three attributes: to, from and subject.

c) Empty Tags:

In cases where you don't have to provide any sub tags, you can close the Tag, by providing a "/" to the Closing Tag. For example declaring

<Text></Text> is same a declaring <Text />

d) Comments in XML File:

Comments in XML file are declared the same way as Comments in HTML File.
<Text>Welcome To XML Tutorial </Text>
<!-- This is a comment -->
<Subject />

e) Processing Instructions

An XML file can also contain processing instructions that give commands or information to an application that is processing the XML data. Processing instructions have the following format:

<?target instructions?>

Where the target is the name of the application that is expected to do the processing, and instructions is a string of characters that embodies the information or commands for the application to process.

2) XML was developed by an XML Working Group formed under the auspices of the World Wide Web Consortium (W3C) in 1996.

The design and development goals of XML are:

- XML shall be straightforwardly usable over the Internet. Users must be able to view XML documents as quickly and easily as HTML documents.
- XML shall support a wide variety of applications. XML should be beneficial to a wide variety of diverse applications: authoring, browsing, content analysis, etc.
- XML shall be compatible with SGML. Most of the people involved in the XML effort come from organisations that have a large, in some cases staggering, amount of material in SGML.
- It shall be easy to write programs, which process XML documents.
- The number of optional features in XML is to be kept to the absolute minimum, ideally zero.
- XML documents should be human-legible and reasonably clear.
- The XML design should be prepared quickly. XML was needed immediately and was developed as quickly as possible.
- The design of XML shall be formal and concise.
- XML documents shall be easy to create.
- Terseness in XML markup is of minimal importance.

3) DTD stands for document type definition. A DTD defines the structure of the content of an XML document, thereby allowing you to store data in a consistent format. Document type definition lists the elements, attributes, entities, and notations that can be used in a document, as well as their possible relationships with one another. The document type declaration identifies the root element of the document and may contain additional declarations. To determine whether document is valid or not, the XML processor must read the entire document type declaration.

In DTDs, the structure of the data can be specified by declaring element to denote the data. DTD can be declared with the following syntax:

<!Element elementname(content-type or content-model)>

In the above syntax, elementname specifies the name of the element & content-type or content-model specifies whether the element contains textual data or other elements. . For e.g. #PCDATA which specifies that element can store parsed character data (i.e. text). DTD can describe two types of elements

Empty : Empty elements have no content and are marked up as <empty-elements>

Unrestricted: The opposite of an empty element is an unrestricted element, which can contain any element declared elsewhere in the DTD.
DTD may contain various symbols. These varied symbols are used to specify an element and may be mandatory, or optional, or number. For example '+' is used indicate that there can be at least one or multiple occurrences of the element, '*' indicates that there can be either zero or any number of occurrences of the element, '?' indicates that there can be either zero or exactly one occurrence etc.
Attribute can be declared in DTD by the following syntax:
<!ATTLIST elementname att_name val_type [att_type] ["default"]>
att_name is the name of the attribute and val_type is the type of value which an attribute can hold such as CDATA. CDATA defines that this attribute can contains a string.
An Attribute type may be defined as REQUIRED, FIXED or IMPLIED.

4) XML parser, which enforces the DTD rules on the XML document, is known as the validating parser, whereas the XML document which ignores DTD rules, is known as the non-validating parser.

5) Entity declarations are used in XML document to associate a name with some other fragment of the document. It is also used to create shortcuts for the frequently typed text. That construct/text can be a chunk of regular text, a chunk of the document type declaration, or a reference to an external file containing either text or binary data. There are three varieties of entities in XML.

Internal Entity: In internal entity the replacement text is stored in the declaration. Internal entities allow you to define shortcuts for frequently typed text or text that is expected to change, such as the revision status of a document. The declaration of an internal entity has this form:

<!ENTITY name "replacement text">
for e.g,
<!ENTITY BCA "Bachelor of Computer Application.">
Using &BCA; anywhere in the document insert "Bachelor of computer Application." at that location.

External Entities: External entities allow an XML document to refer to an external file. External entities contain either text or binary data. If they contain text, the content of the external file is inserted at the point of reference and parsed as part of the referring document. Binary data is not parsed and may only be referenced in an attribute. Binary data is used to reference figures and other non-XML content in the document. For e.g.,

<!ENTITY course SYSTEM "/standard/course1.xml">
Using &course; will have insert the contents of the file /standard/legalnotice.xml at that location in the document when it is processed.

Parameter Entities :
Parameter entities can only occur in the document type declaration. A parameter entity is identified by placing "% " (percent-space) in front of its name in the declaration. The percent sign is also used with reference to parameter entities, instead of the ampersand. Parameter entity references are immediately expanded in the document type declaration and their replacement text is part of the declaration, whereas normal entity references are not expanded.

6) You can declare entities inside either the internal subset or the external subset of the DTD. If , you have an external DTD, you will have to create a complete DTD. If, you need only the entities then, you can get away with an internal DTD subset.

Entity references in XML documents that have external DTD subsets are only replaced when the document is validated.

7) As such we do not need an XML declaration. XML has also been approved as a MIME type, which means that if you add the correct MIME header (xml/text or xml/application), a Web server can explicitly identify the data that follows as being an XML document, regardless of what the document itself says. MIME, (Multipurpose Internet Mail Extensions), is an Internet standard for the transmission of data of any type via electronic mail. It defines the way messages are formatted and constructed, can indicate the type and nature of the contents of a message, and preserve international character set information. MIME types are used by Web servers to identify the data contained in a response to a retrieval request.

The XML declaration is not mandatory for some practical reasons; SGML and HTML code can often be converted easily into perfect XML code (if it isn't already). If the XML declaration was compulsory, this wouldn't be possible.

8) You can use entity references in attribute values, but an entity cannot be the attribute value. There are strict rules on where entities can be used and when they are recognised. Sometimes they are only recognised when the XML document is validated.

9) Technically, there's nothing which stops you from using binary data in a CDATA section, even though it's really a character data section, particularly as, the XML processor doesn't consider the contents of a CDATA section to be part of the document's character data. However, this may cause increase in file size and all the transportation problems that may be implied. Ultimately, it would could create problems for the portability of the XML documents when there is a far more suitable feature of XML you can use for this purpose. Entities, allow you to declare a format and a helper application for processing a binary file (possibly displaying it) and associating it with an XML document by reference.

10) It all depends on your application. HTML has about 77 elements, and some of the industrial DTDs have several hundred. It isn't always the number of elements that determine the complexity of the DTD. By using container elements in the DTD (which add to the element count), authoring software is able to limit possible choices and automatically enter required elements. Working with one of the major industrial DTDs can actually be far easier than creating HTML. Because HTML offers you more free choice, you have to have a much better idea of the purpose of all the elements rather than just the ones you need.

11) No, but unless you are certain that your documents are valid you cannot predict what will happen at the receiving end. Although the XML specification lays down rules of conduct for XML processors and specifies what they must do when certain invalid content is parsed, the requirements are often quite loose. Validation certainly won't do any harm—though it might create some extra work.

12) Simply validate an XML document that uses the DTD. There aren't as many tools specifically intended for checking DTDs as there are for validating documents. However, when an XML document is validated against the DTD, the DTD is checked and errors in the DTD are reported.

13) There are several ways. You could declare a TEXT notation, but this would not allow you to physically include the text in the XML document. (It would go to the helper application you designated in the notation declaration.) The best way would probably be to declare the file containing the text as an external text entity and put the text in that file in a CDATA section.

4.12 FURTHER READINGS/REFERENCES

- Brett McLaughlin & Justin Edelson, *Java and XML*, Third Edition, Oreilly
- Simon North, *Sams Teach yourself XML in 21 days,* Macmillan Computer Publishing
- Eric Westermann, *Learn XML in a weekend XML Bible*, Premier press
- Natanya Pitts-Moultis and Cheryl Kirk, *XML Black Book,* The Coriolis Group

References websites:

- www.w3schools.com
- www.xml.com
- www.java.sun.com/xml/tutorial_intro.html
- www.zvon.org
- www.javacommerce.com