Chapter Two:

2. XML and Related Technologies

2.1. Introduction to XML

2.1.1.Types of computer files

XML is a technology concerned with the description and structuring of *data*, so before we can really look into the concepts behind XML, we need to understand how computers store and access data. Computers understand two kinds of data files: **binary** files and **text** files.

Binary Files

A *binary file* is just a stream of *bits* (1s and 0s). They can only be read and produced by certain computer programs, which have been specifically written to understand them.

For instance, when a document is created with Microsoft Word, the program creates a binary file with an extension of "doc,'' in its own proprietary format. The programmers who wrote Word decided to insert certain binary codes into the document to denote bold text, codes to denote page breaks, and other codes for all of the information that needs to go into a "doc'' file. When you open a document in Word, it interprets those codes and displays the properly formatted text or prints it to the printer.

The codes inserted into the document are *meta data*, or information about information. Examples could be "this word should be in bold," "that paragraph should be centered," and so on. This meta data is really what differentiates one file type from another; the different types of files use different kinds of meta data. For example, a word processing document has different meta data than a spreadsheet document, because they are describing different things. Not so obviously, documents from different word processing applications, such as Microsoft Word and WordPerfect, also have different meta data, because the applications were written differently.

Binary file formats are advantageous because it is easy for computers to understand these binary codes—meaning that they can be processed much faster than non-binary

formats—and they are very efficient for storing this meta data. There is also a disadvantage, as you've seen, in that binary files are proprietary.

Text Files

Like binary files, text files are also streams of bits. However, in a text file these bits are grouped together in standardized ways, so that they always form numbers. These numbers are then further mapped to characters. For example, a text file might contain the following bits: **1100001**. This group of bits would be translated as the number **97**, which could then be further translated into the letter **a**.

Because of these standards, text files can be read by many applications, and can even be read by humans, using a simple text editor. If I create a text document, anyone in the world can read it (as long as they understand English, of course) in any text editor they wish. Some issues still exist, such as the fact that different operating systems treat line-ending characters differently, but it is much easier to share information when it's contained in a text file than when the information is in a binary format. Some of the applications, for example, which can open/read text files are Microsoft word, Notepad, internet explorer, Netscape Navigator, visual studio, WordPad, PowerPoint etc.

The disadvantage of text files is that adding other information—our meta data, in other words—is more difficult and bulky. For example, most word processors enable you to save documents in text form, but if you do, you can't mark a section of text as bold or insert a binary picture file. You will simply get the words with none of the formatting.

2.1.2. A brief History of Markup

You can see that there are advantages to binary file formats (easy to understand by a computer, compact, the ability to add meta data), as well as advantages to text files (universally interchangeable). What if there were a format that combined the universality of text files with the efficiency and rich information storage capabilities of binary files?
In fact, for as long as computers have been around, programmers have been trying to find ways to exchange information between different computer programs. An early attempt to combine a universally interchangeable data format with rich information storage capabilities was *Standard Generalized Markup Language (SGML)*. **SGML** is a text-based language that can be used to mark up data—that is, add meta data—in a way that is *self-describing*.

SGML was designed to be a standard way of marking up data for any purpose, and took off mostly in large document management systems. When it comes to huge amounts of complex data, a lot of considerations must be taken into account, so SGML is a very complicated language.

A very well-known language based on the SGML work is the *Hypertext Markup Language (HTML)*. HTML uses many of SGML's concepts to provide a universal markup language for the display of information, and the linking of different pieces of information. The idea was that any HTML document would be presentable in any application that was capable of understanding HTML (termed a *web browser*).
Even many word processors, such as WordPerfect and Word, allow you to save documents as HTML. Any HTML editor, including a simple text editor, can create an HTML file, and that HTML file can then be viewed in any web browser on the Internet!

2.1.3.What is XML?

The SGML is complicated language. it's not well suited for data interchange over the web. In addition, although HTML has been incredibly successful, it's limited in scope: because it is only intended for displaying documents in a browser. The tags it makes available do not provide any information about the content they encompass. This is because html uses to provide only instructions about how to display that content. HTML document that displays information.Html doesn't describe specialized information. In fact, HTML wouldn't even know that the document was about something at all. Extensible Markup Language (XML) was created to address these issues.XML is a text-based markup language derived from Standard Generalized Markup Language (SGML).XML is not really a "language" at all, but a standard for creating languages that meet the XML criteria. In other words, XML describes a syntax that creates languages. For example, suppose someone has data about a name, and we want to be able to share that information with others as well as use that information in a computer program. Instead of just creating a text file of :- "John Doe" On an HTML file like this

```
<html>
<head><title>Name</title></head>
<body>
<p>John Doe</p>
</body>
</html>
```

We might create an XML file like the following:

```
<name>
<first>John</first>
<last>Doe</last>
</name>
```

From this simple example, the markup languages such as SGML and XML are called"self-describing." The data on the above is information about a <name>, and there is data called <first> and more data called <last>. It is possible to give the tags any names as we like, but if for XML use, we should use the right and give things meaningful names.

Therefore, XML:-

- XML is a data structuring technology, used to design document formats for a wide range of uses.

- XML is a language for creating other markup languages. So, you can use XML to create a specific document format, such as HTML
- XML is a markup language – information about the data is embedded in the document with the data itself. As a result, XML is self-describing. This makes it well suited for data interchange.
- XML is infrastructure – it is the core building block for a wide range of other technologies.
- XML is verbose. Since XML is text-based, it is generally larger than the equivalent binary representation.

XML is a meta markup language for text documents / textual data. XML allows to define languages („applications“) to represent text documents / textual data

But, XML is not

- ✓ A replacement for HTML (but HTML can be generated from XML)
- ✓ A presentation format (but XML can be converted into one)
- ✓ A programming language (but it can be used with almost any language)
- ✓ A network transfer protocol (but XML may be transferred over a network)
- ✓ A database (but XML may store into a database)

An XML *document* is a basic unit of XML information composed of elements and other markup in an orderly package. An XML *document* can contains wide variety of data. For example, database of numbers, numbers representing molecular structure or a mathematical equation.

- ✓ Hierarchical, human-readable format
  - A “sibling” to HTML, always parsable
  - “Lingua franca” of data: encodes documents and structured data
  - Blends data and schema (structure)
- ✓ Core of a broader ecosystem
  - Data – XML
  - Schema – DTD and XML Schema
  - Programmatic access – DOM and SAX
  - Query – XPath, XSLT, XQuery
  - Distributed programs – Web services

HTML Versus XML

- While HTML does for display, XML is designed to do for data exchange. XML will do a good job of communicating information.
- HTML is designed for a *specific* application, to convey information to humans (usually visually, through a web browser), whereas XML has no specific application; it is designed for whatever use we need it for.
- Because HTML has its specific application, it also has a finite set of specific markup constructs (<p>, <ul>, <h2>, and so on), which are used to create a correct HTML document.
- In theory, any web browser will understand an HTML document because all it has to do is understand this finite set of tags. But, in practice, web pages that displayed properly in one web browser and not in another, but this is usually a result of nonstandard HTML tags, which were created by browser vendors instead of being part of the HTML specification itself.
- In an XML document, any XML parser will be able to retrieve information from that document, even though we can't guarantee that any application will be able to understand *what that information means*. That is, just because a parser can tell us that there is a piece of data. For example, for element <middle> that the information contained, there is any software in the world that knows what a <middle> is, what it is used for, or what it means. In other words, it is possible to create XML documents to describe any information as we need. XML can also provide more capabilities in powerful technological services than HTML.

Generally, when we come to the basic difference XML is not a replacement for HTML because XML and HTML were designed with different goals:

- XML was designed to transport and store data, with focus on what data is.
- HTML was designed to display data, with focus on how data looks.

Therefore,

- HTML
  - Designed for a specific application, namely, presenting and linking hypertext

documents

- HTML describes both structure (e.g. <p>, <h2>,) and appearance (e.g.
, <font>, <i>)
- HTML uses a fixed, unchangeable set of tags
- HTML is used to mark up text so it can be displayed to users
- HTML is for humans
    - HTML describes web pages
    - Not case sensitive

- XML describes *structure* and *content* ("semantics")
    - The presentation is defined separately from the structure and the content
    - XML describes only content, or "meaning"
    - Used for easily distribute electronic documents on the world wide web
    - Support valid and well formedness of the document
    - In XML tags are not fixed. We can create any type of tags as we need hence xml tags are not predefined
    - Is a prolog
    - XML is used to mark up data so it can be processed by any computers
    - XML :-
        - XML describes data
        - The rules are strict and errors are not allowed. It is case sensitive
        - Current versions of most browsers can display XML.
        - However, browser support of XML is spotty at best
- The similarity of both are based on SGML(standardized generalized markup language). HTML and XML look similar, because they are both SGML languages (SGML = Standard Generalized Markup Language)
    - Both HTML and XML use elements enclosed in tags (e.g. <body>This is an element</body>)
    - Both use attributes (e.g., <font **face**="Verdana" **size**="+1" **color**="red">)
    - Both use entity tags (<,>, &, ", ')

XML properties, Syntax Rules, XML Comments, and Building on XML Notations

I. Properties of XML

- **XML does not do anything:-**. XML was created to structure, store, and transport information. The following example is a note to DMU from AASTU, stored as XML:

```
<note>
<to>DMU</to>
<from>AASTU</from>
<heading>To Remind the Meeting</heading>
<body>please don't forget</body>
</note>
```

The above XML document does not do anything. It is just pure information wrapped in tags.

- **XML is Plain Text:-**Software that can handle plain text can also handle XML. The functional meaning of the tags depends on the nature of the application.
- The names of XML-elements and XML-attributes are case-sensitive, which means the name of start and end elements need to be written in the same case.
- **With XML we can invent ours Tags: -**xml tags can be invented by the author of the XML document. That is because the XML language has no predefined tags because XML allows the author to define his own tags and his own document structure.
- **XML is a complement to HTML:-**XML is not a replacement for HTML. In most web applications, XML is used to transport data, while HTML is used to format and display the data. XML is a software- and hardware-independent tool for carrying information.
- **XML is everywhere:-**It is the most common tool for data transmissions between all sorts of applications, and is becoming more and more popular in the area of storing and describing information.
- **XML Separates Data from HTML:-**data can be stored in separate XML files. The data stored is independent from the whole system or web.
- **XML Simplifies Data Sharing:-**XML data is stored in plain text format. This provides a

software- and hardware-independent way of storing data. So, it is easier to create data that different applications can share.

- **XML Simplifies Data Transport:-**One of the most time-consuming challenges for developers is to exchange data between incompatible systems over the Internet/ web. Exchanging data as XML greatly reduces this complexity, since the data can be read by different incompatible applications.

- **XML Simplifies Platform Changes:-**Upgrading to new systems (hardware or software platforms), is always very time consuming. Large amounts of data must be converted and incompatible data is often lost.XML data is stored in text format. This makes it easier to expand or upgrade to new operating systems, new applications, or new browsers, without losing data.

- **XML Makes Data More Available:-**Since XML is independent of hardware, software and application, it can make data more available and useful. With XML, our data can be available to all kinds of "reading machines" (Handheld computers, voice machines, news feeds, etc), and make it more available for blind people, or people with other disabilities.

- **XML is Used to Create New Internet Languages: -**A lot of new Internet languages applications are created with XML. Here are some examples:-

  - ✓ XHTML the latest version of HTML
  - ✓ WSDL for describing available web services
  - ✓ WAP and WML as markup languages for handheld devices
  - ✓ RSS languages for news feeds
  - ✓ RDF and OWL for describing resources and ontology
  - ✓ SMIL for describing multimedia for the web

II. XML syntax rules

The syntax rules of XML are very simple and logical. The following are some rules of XML syntax:-

- ✓ **All XML Elements Must Have a Closing Tag:-**it is illegal to omit the closing tag for all xml tags. All elements must have a closing tag, i.e all tags are container types.
- ✓ **The XML declaration is case sensitive** and must begin with **"<?xml…….?>"** where "xml" is written in lower-case. And the dots show as we can incorporate one or more attributes.
- ✓ **XML Elements must be properly nested**
  - The more interior the tag will be closed first. All elements **must** be properly nested within each other:

    This text is bold and italic
- ✓. 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 Attribute Values must be quoted**

XML elements can have attributes in name/value pairs and the attribute value must always be quoted. Study the two XML documents below. The first one is incorrect, the second is correct:

```
Ex 1:<note date=12/11/2007>
  <to>DMU</to>
  <from>AASTU</from>
</note>

Ex2:-
<note date="12/11/2007">
  <to>DMU</to>
  <from>AASTU</from>
```

```
</note>
```

The error in the first document is that the date attribute in the note element is not quoted.

✓Entity References

Some characters have a special meaning in XML like html. If we place a character like "<" inside an XML element, it will generate an error because the parser interprets it as the start of a new element.

- This generate error: <message>if salary < 1000 then</message>
- But, <message>if salary < 1000 then</message> is correct.

There are 5 predefined entity references in XML:

Entity reference	Symbol	Meaning
<	**<**	**less than**
>	**>**	**greater than**
&	**&**	**ampersand**
'	**'**	**Apostrophe**
"	**"**	**quotation mark**

✓**White-space is preserved in XML:** -With XML, the white-space in a document is not truncated.

☞ An Example of XML Document

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<note>
  <to>DMU</to>
  <from>AASTU</from>
  <heading>To Remind</heading>
 <body>Don't forget me this weekend!</body>
</note>
```

The first line is the **XML declaration**. It defines the XML version (1.0) and the encoding used (ISO-8859-1 = Latin-1/West European character set).The next line describes the **root element** of the document (like saying: "this document is a note"):**<note>.**The next 4 lines describe 4 **child**

elements of the root (to, from, heading, and body). Finally the last line defines the end of **the root element**: </note>.

```
<?xml version="1.0"?>
<contact-info>
   <name>Tanmay Patil</name>
   <company>TutorialsPoint</company>
   <phone>(011) 123-4567</phone>
</contact-info>
```

Document Prolog

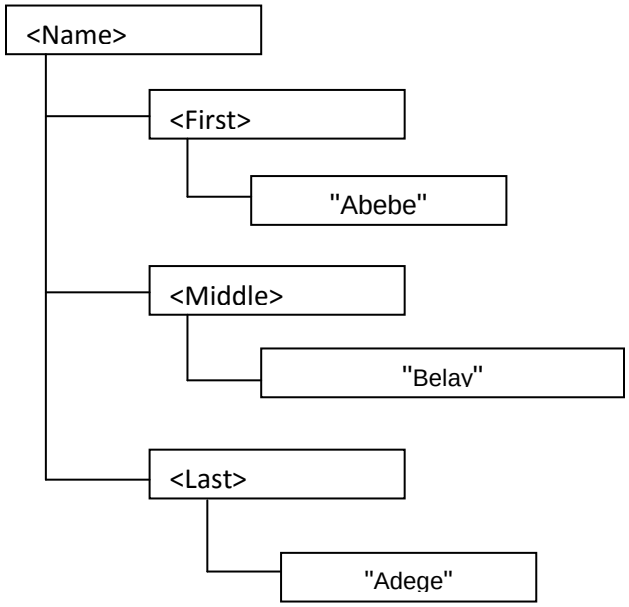
Document Elements

1.1. Hierarchies of Information in XML

Software developers have been using a structure called an ***object model*** to group information into hierarchies. In an object model, all of the information being modeled is divided into various objects, and the objects themselves are then grouped into a hierarchy.

XML groups information in hierarchies. The items in our documents relate to each other in parent/child and sibling/sibling relationships. The items are called ***elements***.

Consider the following example:



☞ <Name> is a parent of <first>. <first>, <middle>, and <last> which are all siblings to each other (they are all children of <Name>).

- Note that the texts also a child of the element. For example, the text Abebe is a child of <First>.This structure is also called a ***tree***, and any parts of the tree that contain children are called **branches**, while parts that have no children are called ***leaves***.
- Because the <name> element has only other elements for children, and not text, it is said to have *element content*. Conversely, because <first>, <middle>, and <last> have only text as children, they are said to have *simple content*. Elements can contain both text and other elements, in which case they are said to have *mixed content*, as shown in the following example:

```
<doc>
<parent>this is some <em>text</em> in my element</parent>
</doc>
```

  - Here, <parent> has three children:
    - this is some
    -
    - in my element
  - Relationships can also be defined by making the family tree analogy work: <doc> is an *ancestor* of ; is a *descendant* of <doc>.

1.2. Applications of XML

XML can be used anywhere. It is platform- and language-independent, which means it doesn't matter that any computer may be using, for example, a Visual Basic application on a Microsoft operating system, and another computer might be a UNIX machine running Java code. Anytime one computer program needs to communicate with another program, XML is a potential fit for the exchange format. The following are some uses of XML:-

i. Reducing Server Load

- Web-based applications can use XML to reduce the load on the web servers by keeping all information on the client, and then sending the information to those servers in one big XML document.
- For example, a consulting company may write a timesheet application whereby employees can enter how much time they've spent on different tasks; the time entered would be used to bill their clients appropriately.

- In the developed and completed documents, an XML document could be sent to the server, with the entire user's data and receive data.

ii. Website Content

- There are technologies such as CSS and XSLT that can be used to transform XML from one format to another, or to "style" XML for viewing in a browser.
- Xml allows for some very powerful applications of the data sets.
- XML documents can transform into HTML for display (by XSLT), or transformed into a number of other presentation formats.
- Some websites use XML entirely for their content. This XML can then be transformed into HTML via XSLT, or displayed directly in browsers via CSS. In fact, the web servers can even determine dynamically what kind of browser is retrieving the information, and then decide what to do—for example, transform the XML into HTML for older browsers, and just send the XML straight to the client for newer browsers, reducing the load on the server.

iii. Distributed Computing

- XML can also be used as a means for sending data for distributed computing, where objects on one computer call objects on another computer to do work.
- Can be done using xml and html without numerous standards for distributed computing: DCOM, CORBA, and RMI/IIOP.

iv. E-Commerce

- XML helps for streamline their processes, decreasing costs and increasing response times.
- Whenever one company needs to send data to another, XML is the perfect format for the exchange.
- XML is also a good fit for many other applications, depending up on users' interest.

XML has the following possible advantages

- ✓ Truly Portable Data
- ✓ Easily readable by human users
- ✓ Very expressive (semantics near data)
- ✓ Very flexible and customizable (no finite tag set)
- ✓ Easy to use from programs

- ✓ Easy to convert into other representations (XML transformation languages)
- ✓ Many additional standards and tools
- ✓ Widely used and supported

1.3. XML Well-formed

Well formed to XML means that it has no syntax, spelling, punctuation, grammar errors, etc. in its markup. Hence, an XML document must be well formed to be processed by XML parsers.

☞ An XML document (with or without a DTD) is *well-formed* if

- Tags are syntactically correct
- Every tag has an end tag
- Tags are properly nested
- There is a root tag
- A start tag does not have two occurrences of the same attribute

A software module called an XML processor is used to read XML documents and provide access to their content and structure. It is assumed that an XML processor is doing its work on behalf of another module, called the **application**.

An XML processor is more commonly called a ***parser which*** used to read XML documents and provide access to their content and structure, as it simply parses XML and provides the application with any information it needs. Some of the better-known XML parsers include the following:

- **Microsoft Internet Explorer Parser**—with the release of IE5, the XML implementation was upgraded to reflect the XML version 1.0.
- **Apache Xerces**—can parse Java and C++, Perle. These tools are free, and the distribution of the code is controlled by the GNU Public License (GPL).
- **Expat**—Expat is an XML 1.0 parser toolkit written in C. It is free for both private and commercial use.
- **XML editors:-** easy, free and applicable for different tasks

- **Dreamweaver** or **Cooktop** have XML parsers built into the software application, but we can also check for well-formed XML with most Internet browsers.
- The most recent versions of Internet Explorer, Netscape, Mozilla or Firefox have at least some simple XML parsing functionality built in and can check XML documents for well-formed markup.

Each parser can have different error messages for the same mistake, but the most common errors are not having closing tags, element names not matching up, not closing quotation marks for attributes, and incorrect order.

☞ Since XML requires elements to have opening and closing tags, missing a closing tag will cause what XML calls a "fatal" error - the parser will shut down and give an error message.

1.3.1. Tags and text elements

- Xml file should be saved as filename.xml
- The text starting with a **<** character and ending with a **>** character is an XML *tag*.
- The tags are paired, so that any opening tag (for example, **<name>**) must have a closing tag (**</name>**), which are called *start-tags* and *end-tagsrespectivelly*. The end-tags are the same as the start-tags except that they have a / right after the opening " **<**" character.
- Example:-

```
<?xml version="1.0"?>
<weatherReport>
<date>7/14/97</date>
<city>North Place</city>
<state>NX</state>
<country>USA</country>
   High Temp: <high scale="F">103</high>
   Low Temp: <low scale="F">70</low>
   Morning: <morning>Partly cloudy, Hazy</morning>
   Afternoon: <afternoon>Sunny &amp; hot</afternoon>
   Evening: <evening>Clear and Cooler</evening>
</weatherReport>
```

- Documents outside tags are not recognized by XML parsers.
- Each tags and documents are called xml elements

✓ Sometimes an element has no PCDATA (Parsed Character Data). Recall our earlier example in which the **<middle>** element contained no name. In this case, there is also an option of writing this element using the special ***empty element***syntax (this syntax is also called a *self-closing tag)*:<middle/> is equivalent with <middle></middle>.

✓ See the following example:-

<middle />Or this <middle/>

But not like this

<middle/ >Or this<middle / >

✓ For example, if we rewrote our <name> example without child elements, instead of a start-tag and end-tag we would probably use an empty element, like this: <name first=”John” middle=”Fitzgerald Johansen” last=”Doe”/>Or, for readability, XML authors will often write the XML like this:

```
<name first=”John”
        middle=”Fitzgerald Johansen”
        last=”Doe”
    />
```

✓ As noted earlier, the order of attributes is considered irrelevant.

✓ PCDATA:-

- CDATA cannot contain the string "]]>" anywhere in the XML document.
- Nesting is not allowed in CDATA section.

1.3.2. XML Attributes

☞ In addition to tags and elements, XML documents can also include attributes. Attributes are simple name/value pairs associated with an element.

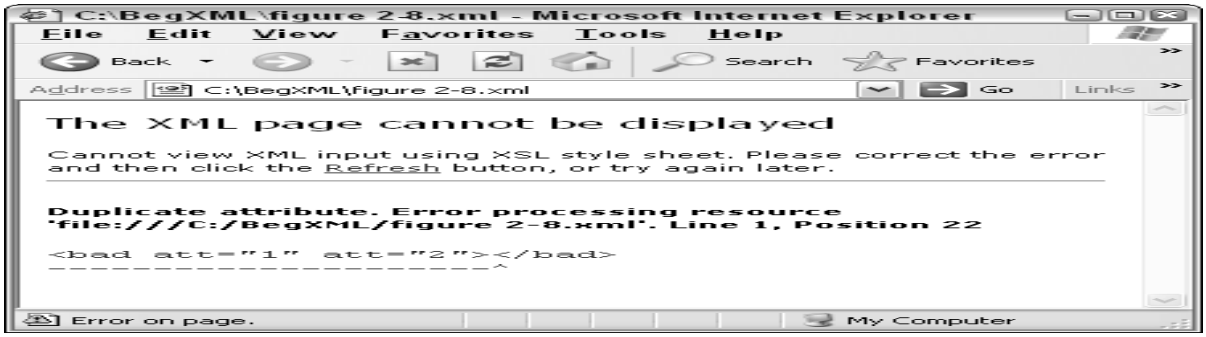
☞ They are attached to the start-tag, but not to the end-tag, as shown in the following code:-

```
<name nickname=”Shiny John”>
        <first>John</first>
        <middle>Fitzgerald Johansen</middle>
```

```
<last>Doe</last>
</name>
```

- Attributes must have values—even if that value is just an empty string (such as “”)—and those values must be in quotes. The following example, is not legal in XML: <input checked>, <input checked=true>
- Either single quotes or double quotes are possible , but they have to match. For example, to make this into well-formed XML, we can use <input checked='true'> Or <input checked="true">. But not use like:- <input checked="true'>

- In addition, we can't have more than one attribute with the same name on an element like <bad att="1" att="2">this is sample code for attributes</bad>. If we use the tag shown here and run in xml parser, the following message will be displayed.



- When we write xml, first we should "normalize" the data in an attribute before it passes it on to the application by a bit of pre-processing of the text. For example strip out newline characters and replace them with a single space as shown below:-

```
<test myAttr='some data
        Goes
here'>some other data</test>=>
```

```
<test myAttr='some data goes
here>
```

Example:
```
<?xml version="1.0">
<article>
<section id="1" title="Intro">
This article is about <index>XML</index>.
</section>
```

```
<section id="2" title="Main Results">
<name>Weikum</name><cite idref="Weik01"/> shows
the following theorem (see Section <ref idref=„1"/>)
<theorem id="theo:1" source="Weik01">
For any XML document x, ...
</theorem>
</section>
<literature>
<cite id="Weik01"><author>Weikum</author></cite>
</literature>
</article>
```

1.3.3. XML Comments

Using *comments,* it is possible to insert into an XML document text that isn't really part of the document, but rather is intended for people who are reading the XML markup itself.

Comments may not be as relevant to XML as they are to programming languages; after all, this is just data, and it's self-describing to boot. Still, we never know when they're going to come in handy, and there are cases where comments can be very useful, even in data.

Comments start with the string **<!--**and end with the string **-->**, as shown here:

```
<name nickname='Shiny John'>
<first>John</first>
<!--John lost his middle name in a fire-->
<middle></middle>
<last>Doe</last>
</name>
```

Note:-

- We can't have a comment inside a tag, so the following is illegal: <middle></middle **<!--**John lost his middle name in a fire**-->**>

- It is impossible to use the double-dash string (--) inside a comment, so the following is also illegal: <!--John lost his middle name -- in a fire-->

The XML specification states that an XML parser doesn't need to pass these comments on to the application, meaning that we should never count on being able to use the information inside a comment from our application. Comments are only there for the benefit of someone reading our XML markup.

1.3.4. XML Declarations

It is often very handy to be able to identify a document as being of a certain type.

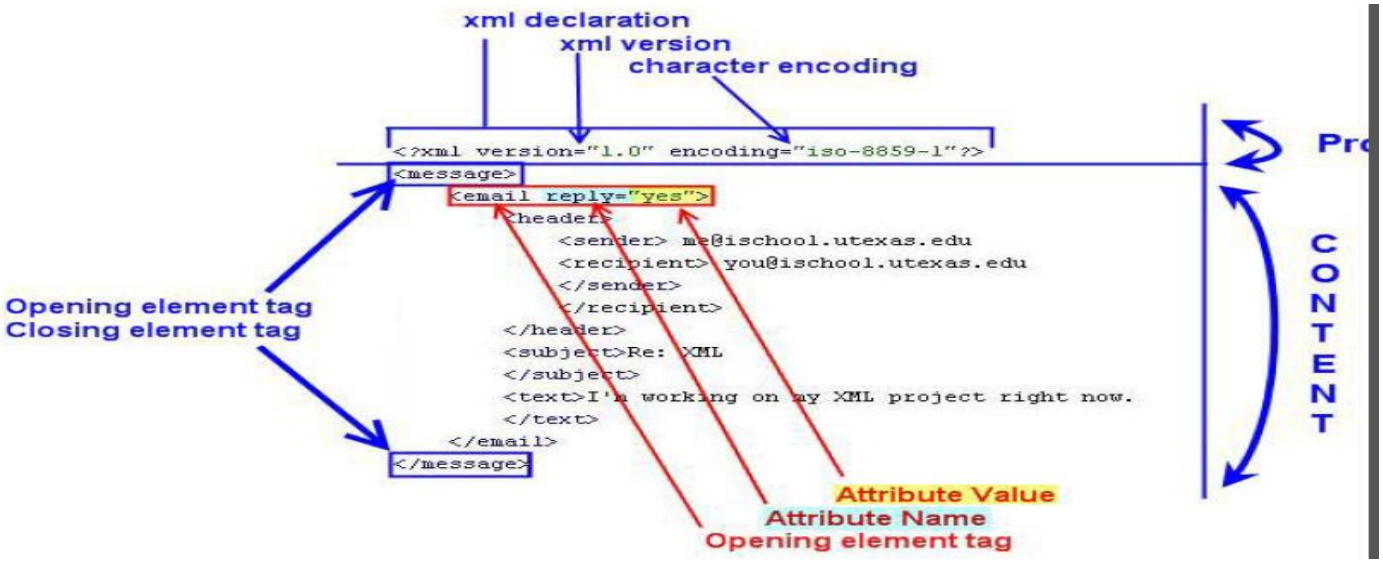
- It is `declared as:-`

```
<?xml
version="version_number"
encoding="encoding_declaration"
standalone="standalone_status"
?>
```

followed by xml root and its child tags as well as documents

- XML declaration can be written as:-
  - With no parameters as <?xml >
  - Declaration with version definition <?xml version="1.0"?>
  - Declaration with all parameters defined <?xml version="1.0" encoding="UTF-8" standalone="no"?>

❖ Example 1:



Example 2:

```
<?xml version='1.0' encoding='UTF-16' standalone='yes'?>
<name nickname='Shiny John'>
<first>John</first>
<!--John lost his middle name in a fire-->
<middle/>
<last>Doe</last>
</name>
```

Note:

- The XML declaration starts with the characters **<?xml** and ends with the characters **?>**.
- If we include a declaration, we must include the version, but the encoding and standalone attributes are optional.
- The version, encoding, and standalone attributes must be in that order.
- The version should be 1.0 or 1.1 values.
- The XML declaration must be right at the beginning of the file.

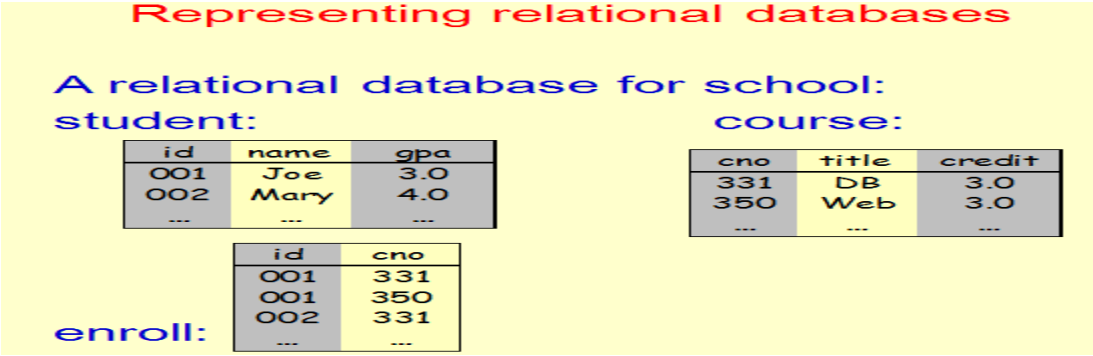
For example, an XML declaration can be as full as the previous one or as simple as the following: <?xml version='1.0'?>. The two sections describe more fully the encoding and standalone attributes of the XML declaration:-

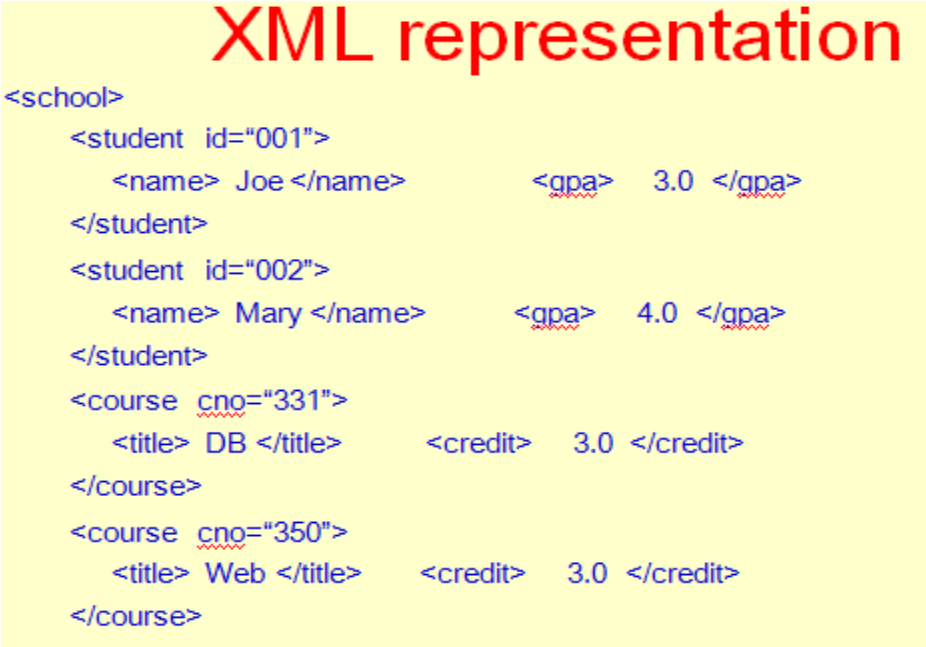
- ✓ The version attribute specifies which version of the XML specification the document adheres to.

- ✓ Unless we're working with some Unicode data that just won't work under the 1.0 specification, we should always specify 1.0 for the version.
- ✓ If the standalone attribute is included in the XML declaration, it must be set to either yes or no:
  - ➢ Yes specifies that the document exists entirely on its own, without depending on any other files.
  - ➢ No indicates that the document may depend on an external DTD.

This little attribute actually has its own name: the Standalone Document Declaration, or SDD. The XML Recommendation doesn't actually require a parser to do anything with the SDD. It is considered more of a hint to the parser than anything else.

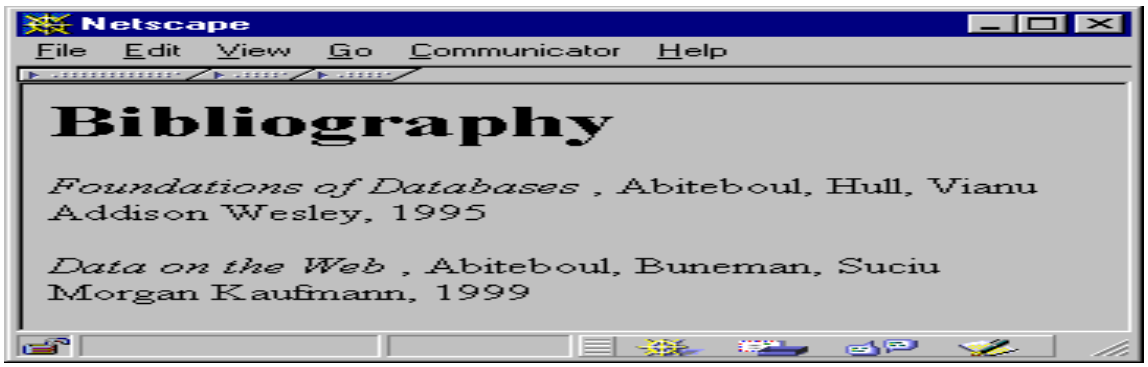
Example 1:-Let we see the following example how a database represent in XML





```
    <enroll>
      <id> 001 </id>          <cno>   331 </cno>
    </enroll>
    <enroll>
      <id> 001 </id>          <cno>   350 </cno>
    </enroll>
    <enroll>
      <id> 002 </id>          <cno>   331 </cno>
    </enroll>
</school>
```

Example 2:- let we convert from HTML to XML



The equivalent XML code will be:-

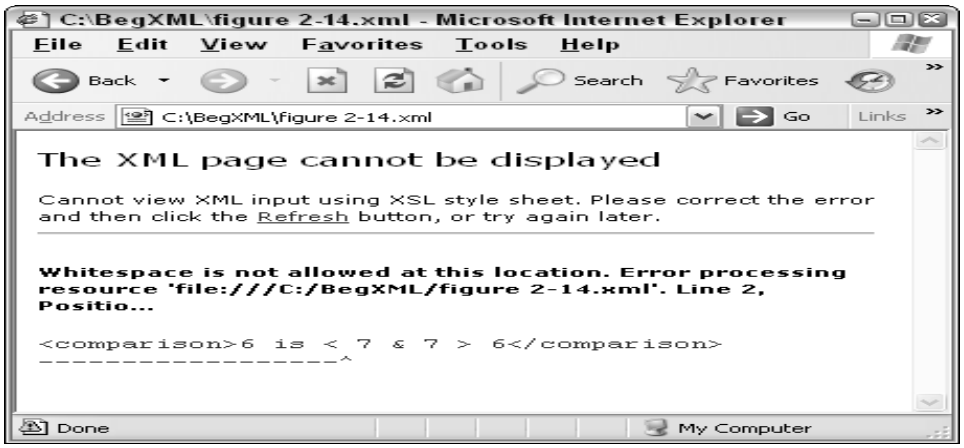
```
<bibliography>
<book><title> Foundations… </title>
<author>Abiteboul</author>
<author> Hull </author>
<author>Vianu</author>
<publisher> Addison Wesley </publisher>
<year> 1995 </year>
</book>
  …
```

```
</bibliography>
```

1.4. PCDATA

- ✓ There are some reserved characters that couldn't be included in PCDATA because they are used in or pre-defined for xml parsers.
- ✓ Written as:

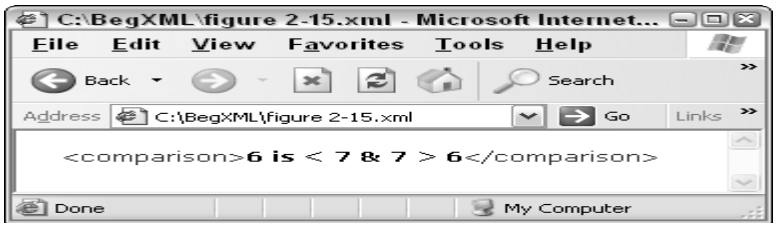
  <[CDATA[ANT STATEMENTS]]>
- ✓ This part is written in xml code to make the xml parser jump from interpreting whether the symbols are reserved or other types.
- ✓ For example: <comparison>6 is < 7 & 7 > 6</comparison> is wrong writing. Viewing the preceding XML in Internet Explorer results in the error shown



- ✓ To make the above code is correct, we should write as <***comparison***>< ***[CDATA6 is < 7 & 7 > 6]]***></***comparison***> in the xml code.
- ✓ Therefore, we can't put raw **<** or **&** characters into PCDATA. There are two ways we can get around this: ***escaping characters***, or enclosing text in a ***CDATA section*****.**

- ✓ To escape the <or **&**characters, simply replace any <character with **<**and any **&**character with **&**. (In addition, escape the >character with **>**. It isn't necessary, but it does make things more consistent, as needed to escape all of the <characters.) .
- ✓ See the following example could be made well formed by doing the following:

<comparison>6 is < 7 & 7 > 6 </comparison. This displays properly in the browser, as shown



- ☞ Using CDATA sections, the XML parser not to parse the text, but to let it all go by until it gets to the end of the section. CDATA sections look like this: <comparison><![CDATA[6 is < 7 & 7 > 6]]></comparison>

- ☞ Everything starting after the **<! [CDATA[and ending at the]]>** is ignored by the parser, and passed through to the application as it is.

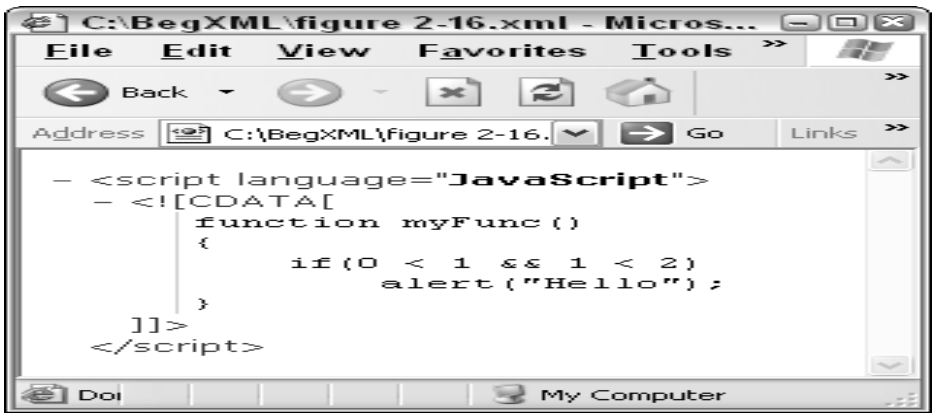
- ☞ Unfortunately, the CDATA syntax introduces another complexity to XML markup: The character sequence**]]>** is not allowed, either in a CDATA section or out. If really needed to have those three characters together, it 'd have to use this:]]>

- ☞ In these trivial cases, CDATA sections may look more confusing than the escaping did, but in other cases it can turn out to be more readable. For example, consider the following example, which uses a CDATA section to keep an XML parser from parsing a section of JavaScript:

```
<script language='JavaScript'>
<![CDATA[function myFunc()
{
```

```
if(0 < 1 && 1 < 2)
alert("Hello");
}
]]></script>
```

The following will be displayed.



Notice: - The vertical line at the left-hand side of the CDATA section indicates that although the CDATA section is indented for readability; the actual data itself starts at that vertical line. As seen exactly what whitespace is included in the CDATA section. In JavaScript, we'll probably find the if statement much easier to read than the following: if(0 < 1 && 1 < 2)

The following example uses a CDATA section to insert an XML into an XML document:

```
<?xml version="1.0"?>
<example>
<[CDATA[
<?xml version="1.0"?>
<entry>
<name>John Doe</name>
<emailhref="mailto:jdoe@emailaholic.com"/>
</entry>]]>
</example>
```

1.5. Errors in XML

There are two types of errors in xml: *errors* and *fatal errors*.

- An ***error*** is simply a violation of the rules in the recommendation, where the results are undefined; the XML processor is allowed to recover from the error and continue processing.
- ***Fatal errors*** are more serious: a parser is not allowed to continue as normal when it encounters a fatal error. (It may, however, keep processing the XML document to search for further errors.). This is called draconian error handling. Any error that causes an XML document to cease being well formed is fatal error.

The reason for this drastic handling of non-well–formed XML is simple: It would be hard for parser writers to try to handle "well-formedness" errors, and it is extremely simple to make XML well formed. (Web browsers don't force documents to be as strict as XML does, but this is one of the reasons why web browsers are so incompatible; they must deal with *all* the errors they may encounter, and try to figure out what the person who wrote the document was really trying to code.)

Draconian error handling doesn't just benefit the parser writers; it also benefits us when we're creating XML documents. If XML document is written that doesn't properly follow XML's syntax, we can find our mistake right away and fix it. Conversely, if the XML parser tried to recover from these errors, it might misinterpret what we were trying to do, but we wouldn't know about it because no error would be raised. In this case, bugs in our software would be much harder to track down, instead of being caught right at the beginning when we were creating our data. Even worse, XML document is sent to someone else, his or her parser might interpret the mistake differently.