UNIT-V

HISTORY OF XML

In the 1970's, Charles Goldfarb, Ed Mosher and Ray Lorie invented GML at IBM. GML was used to describe a way of marking up technical documents with structural tags. The initials stood for Goldfarb, Mosher and Lorie.Goldfarb invented the term "mark-up language" to make better use of the initials and it became the Standard Generalised Markup Language.

In 1986, SGML was adopted by the ISO.

SGML is just a specification for defining markup languages.

SGML (Standardized Generalized Markup Language) is the mother of all markup languages like HTML, XML, XHTML, WML etc...

In 1986, SGML became an international standard for defining the markup languages. It was used to create other languages, including HTML, which is very popular for its use on the web. HTML was made by Tim Berners Lee in 1991.While on one hand SGML is very effective but complex, on the other, HTML is very easy, but limited to a fixed set of tags. This situation raised the need for a language that was as effective as SGML and at the same time as simple as HTML. This gap has now been filled by XML.

The development of XML started in 1996 at SunMicrosystems. Jon Bosak with his team began work on a project for remoulding SGML. They took the best of SGML and produced something to be powerful, but much simpler to use.The World Wide Web Consortium also contributes to the creation and development of the standard for XML.

INTRODUCTION

Extensible Markup Language (**XML**) is a markup language that **defines** a set of rules for encoding documents in a format which is both human-readable and machine-readable. It is defined by the W3C's **XML** 1.0 Specification and by several other related specifications, all of which are free open standards.

The essence of XML is in its name: Extensible Markup Language.
Extensible- XML is extensible. It lets you define your own tags, the order in which they occur, and how they should be processed or displayed.

Markup -The most recognizable feature of XML is its tags, or elements. In fact, the elements you'll create in XML will be very similar to the elements you've already been creating in your HTML documents. However, XML allows you to define your own set of tags.

Language- XML is a language that's very similar to HTML. It's much more flexible than HTML because it allows you to create your own custom tags. However, it's important to realize that XML is not just a language. XML is a meta-language: a language that allows us to create or define other languages. For example, with XML we can create other languages, such as MathML (a mathematical markup language), and even tools like XSLT.

XML was designed to describe data and is a cross-platform, software and hardware independent tool for transmitting or exchanging information. It is an open-standards-based technology which is both human and machine readable. XML are best suited for use in documents that have large amount of similarity.

In future Web development it is most likely that XML will be used to describe the data, while HTML will be used to format and display the same data. XML specification includes the syntax and grammar of XML documents as well as DTD.

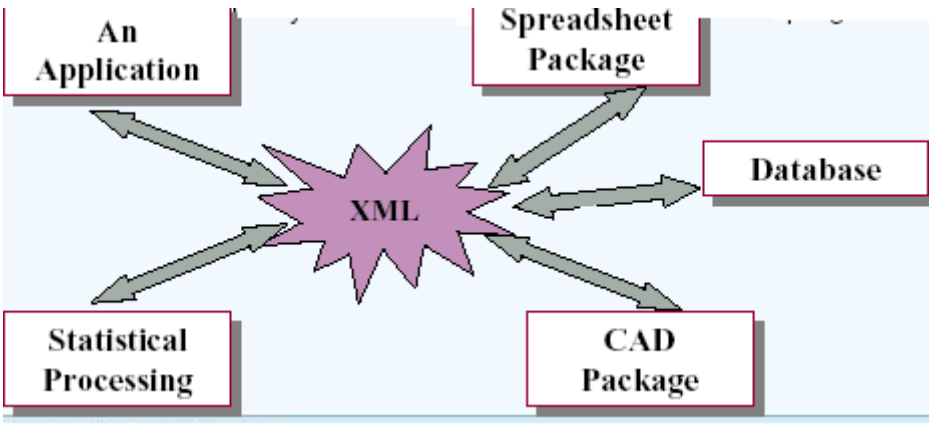
APPLICATIONS OF XML

1.Refined search results - With XML-specific tags, search engines can give users more refined search results. A search engine seeks the term in the tags, rather than the entire document, giving the user more precise results.
2.EDI Transactions - XML has made electronic data interchange (EDI) transactions accessible to a broader set of users. XML allows data to be exchanged, regardless of the computing systems or accounting applications being used.
3.Cell Phones - XML data is sent to some cell phones, which is then formatted by the specification of the cell phone software designer to display text, images and even play sounds!
4.File Converters - Many applications have been written to convert existing documents into the XML standard. An example is a PDF to XML converter.
5.VoiceXML - Converts XML documents into an audio format so that a user can listen to an XML document.

USES OF XML:

XML is widely used for the following purposes

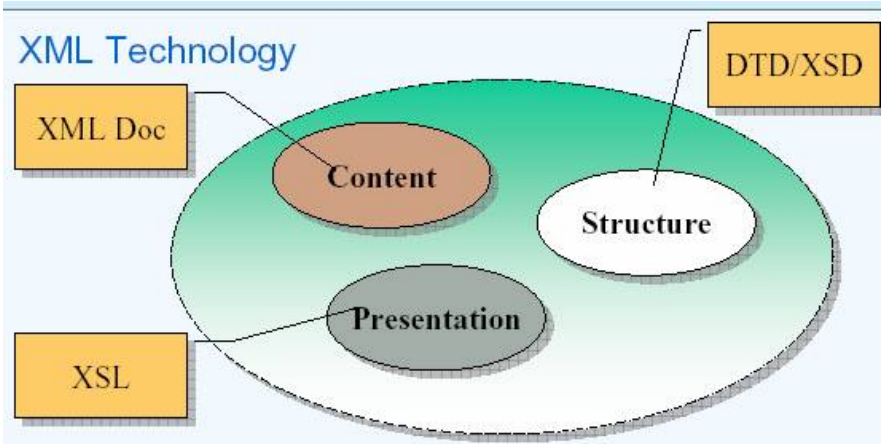
1. Storing data in a structured manner. (Tree structure)
2. Storing configuration information – typically data in an application which is not stored in a database - Most server software have configuration files in XML formats XML documents can also be used as a mini data store. This data can be also used to present it on a variety of targets including browsers, print media, etc.
3. Transmitting data between applications - Overcomes Problems in Client Server applications which are cross platform in nature Ex: A Windows program talking to a mainframe , Little and Big Endian problems, Data type size variations across platforms



When XML data is transferred across different systems, the data contained in an XML document can be read using a software entity called parser. Most of the popular databases (Oracle, MS SQL Server, Sybase, DB2, etc.) provide their own mechanisms to store and retrieve data as XML. Some of them also provide parsers to work with the XML documents programmatically.

XML is a key technology when it comes to Web Services. .NET uses XML extensively. It is used as a data format for everything - configuration files, metadata, RPC, object serialization.

XML TECHNOLOGY



XSD - XML Schema Definition
DTD - Document Type Definition.
XSL - Extensible Stylesheet Language.

The structured data is contained in an XML document, a text file with .xml as extension. We can use CSS, like you use it for HTML, to provide style sheets for XML data display. For more advanced features, power, and flexibility for the presentations, you could use XSL (XML Style sheet Language) to build the style sheets. To enforce structural constraints and rules on the data contained in an XML document, you could code a DTD (Document Type Definition). Due to certain limitations that were inherent in DTDs, W3C came up with a newer specification to serve the same purpose as DTDs – the Schemas. The schemas are contained in a .xsd file, and DTDs in a .dtd file. XML Schema is an XML based alternative to DTD.

ADVANTAGES OF XML

- It is a simultaneously human- and machine-readable format.
- It supports Unicode, allowing almost any information in any written human language to be communicated.
- It can represent the most general computer science data structures: records, lists and trees.
- Its self-documenting format describes structure and field names as well as specific values.
- The strict syntax and parsing requirements make the necessary parsing algorithms extremely simple, efficient, and consistent.
- XML is heavily used as a format for document storage and processing, both online and offline.

- It is based on international standards.
- It allows validation using schema languages such as XSD and Schematron, which makes effective unit-testing, firewalls, acceptance testing, contractual specification and software construction easier.
- The hierarchical structure is suitable for most (but not all) types of documents.
- It manifests as plain text files, which are less restrictive than other proprietary document formats.
- It is platform-independent, thus relatively immune to changes in technology.
- Forward and backward compatibility are relatively easy to maintain despite changes in DTD or Schema.
- Its predecessor, SGML, has been in use since 1986, so there is extensive experience and software available.

DISADVANTAGES OF XML

- XML syntax is redundant or large relative to binary representations of similar data.
- The redundancy may affect application efficiency through higher storage, transmission and processing costs.
- XML syntax is too verbose relative to other alternative 'text-based' data transmission formats.
- No intrinsic data type support: XML provides no specific notion of "integer", "string", "boolean", "date", and so on.
- The hierarchical model for representation is limited in comparison to the relational model or an object oriented graph.
- Expressing overlapping (non-hierarchical) node relationships requires extra effort.
- XML namespaces are problematic to use and namespace support can be difficult to correctly implement in an XML parser.
- XML is commonly depicted as "self-documenting" but this depiction ignores critical ambiguities.

XML Syntax

The syntax rules of XML are very simple and logical. The rules are easy to learn, and easy to use.

1.All XML Elements Must Have a Closing Tag

In HTML, some elements do not have to have a closing tag:

This is a paragraph.

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

This is a paragraph.

Note: You might have noticed from the previous example that the XML declaration did not have a closing tag. This is not an error. The declaration is not a part of the XML document itself, and it has no closing tag.

2.XML Tags are Case Sensitive

XML tags are case sensitive. The tag <Letter> is different from the tag <letter>.

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

<Message>This is incorrect</message>

<message>This is correct</message>

Note: "Opening and closing tags" are often referred to as "Start and end tags". Use whatever you prefer. It is exactly the same thing.

3.XML Elements Must be Properly Nested

In HTML, you might see improperly nested elements:

This text is bold and italic

In XML, all elements **must** be properly nested within each other:

This text is bold and italic

In the example above, "Properly nested" simply means that since the element is opened inside the element, it must be closed inside the element.

4.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>
```

5.XML Attribute Values Must be Quoted

XML elements can have attributes in name/value pairs just like in HTML.

In XML, the attribute values must always be quoted.

INCORRECT:

```
<note date=12/11/2007>
  <to>Tove</to>
  <from>Jani</from>
</note>
```

CORRECT:

```
<note date="12/11/2007">
  <to>Tove</to>
  <from>Jani</from>
</note>
```

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

6.Entity References

Some characters have a special meaning in XML.

If you 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 will generate an XML error:

<message>if salary < 1000 then</message>

To avoid this error, replace the "<" character with an **entity reference**:

<message>if salary < 1000 then</message>

There are 5 pre-defined entity references in XML:

<	<	less than
>	>	greater than
&	&	ampersand
'	'	Apostrophe
"	"	quotation mark

Note: Only the characters "<" and "&" are strictly illegal in XML. The greater than character is legal, but it is a good habit to replace it.

7.Comments in XML

The syntax for writing comments in XML is similar to that of HTML.

<!-- This is a comment -->

8.White-space is Preserved in XML

XML does not truncate multiple white-spaces in a document (while HTML truncates multiple white-spaces to one single white-space):

XML:	Hello Tove
HTML:	Hello Tove

9.XML Stores New Line as LF

Windows applications store a new line as: carriage return and line feed (CR+LF).
Unix and Mac OSX uses LF.
Old Mac systems uses CR.
XML stores a new line as LF.

Well Formed XML

XML documents that conform to the syntax rules above are said to be "Well Formed" XML documents.

XML Elements

An XML document contains XML Elements.

An XML element is everything from (including) the element's start tag to (including) the element's end tag.

An element can contain:

- other elements
- text
- attributes
- or a mix of all of the above...

```
<bookstore>
  <book category="CHILDREN">
    <title>Harry Potter</title>
    <author>J K. Rowling</author>
    <year>2005</year>
    <price>29.99</price>
  </book>
  <book category="WEB">
    <title>Learning XML</title>
    <author>Erik T. Ray</author>
    <year>2003</year>
    <price>39.95</price>
  </book>
</bookstore>
```

In the example above, <bookstore> and <book> have **element contents**, because they contain other elements. <book> also has an **attribute** (category="CHILDREN"). <title>, <author>, <year>, and <price> have **text content** because they contain text.

Empty XML Elements

An element with no content is said to be empty.
In XML, you can indicate an empty element like this:
<element></element>
Or
you can use an empty tag, like this (this sort of element syntax is called self-closing):
<element />
The two forms above produce identical results in an XML parser.
Note: Empty elements do not have any content, but they can have attributes!

XML Naming Rules
XML elements must follow these naming rules:

- Element names are case-sensitive
- Element names must start with a letter or underscore
- Element names cannot start with the letters xml (or XML, or Xml, etc)
- Element names can contain letters, digits, hyphens, underscores, and periods
- Element names cannot contain spaces

Any name can be used, no words are reserved (except xml).

Best Naming Practices
Create descriptive names, like this: <person>, <firstname>, <lastname>.
Create short and simple names, like this: <book_title> not like this: <the_title_of_the_book>.
Avoid "-". If you name something "first-name", some software may think you want to subtract "name" from "first".
Avoid ".". If you name something "first.name", some software may think that "name" is a property of the object "first".

Naming Styles
There are no naming styles defined for XML elements. But here are some commonly used:

Style	Example	Description
Lower case	<firstname>	All letters lower case
Upper case	<FIRSTNAME>	All letters upper case
Underscore	<first_name>	Underscore separates words
Pascal case	<FirstName>	Uppercase first letter in each word
Camel case	<firstName>	Uppercase first letter in each word except the first

XML Elements are Extensible
XML elements can be extended to carry more information.
Look at the following XML example:

```
<note>
  <to>Tove</to>
  <from>Jani</from>
  <body>Don't forget me this weekend!</body>
</note>
```

Let's imagine that we created an application that extracted the <to>, <from>, and <body> elements from the XML document to produce this output:

MESSAGE

To: Tove
From: Jani
Don't forget me this weekend!

Imagine that the author of the XML document added some extra information to it:

```
<note>
  <date>2008-01-10</date>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

Should the application break or crash?

No. The application should still be able to find the <to>, <from>, and <body> elements in the XML document and produce the same output.One of the beauties of XML, is that it can be extended without breaking applications.

Difference between HTML and XML

	HTML	XML
Definition	Markup language for displaying web pages in a web browser. Designed to display data with focus on how the data looks	Markup language defines a set of rules for encoding documents that can be read by both humans and machines. Designed with focus on storing and transporting data.
Date when invented	1990	1996
Extended from	SGML	SGML
Type	Static	Dynamic
Usage	Display a web page	Transport data between the application and the database. To develop other mark up languages.
Processing/Rules	No strict rules. Browser will still generate data to the best of its ability	Strict rules must be followed or processor will terminate processing the file
Language type	Presentation	Neither presentation, nor programming
Tags	Predefined	Custom tags can be created by the author
White Space	Cannot preserve white space	Preserves white space
Limitations	Data does not know itself very well. Data cannot change in	Cannot be used as a subtype of a sql_variant instance.

	response to environment. Data cannot be easily maintained. Cannot store or call on variables. Lacks the capability to define new structures by defining relationships between classes. Tags are not useful for exchanging the document between applications.	Does not support casting or converting to either text or ntext. Does not support the following column and table constraints. XML provides its own encoding. Collations apply to string types only. Cannot be compared or sorted. Cannot be used in Distributed Partitioned Views. Not well supported by browsers.

XML STYLE SHEETS

A cascading style sheet is a type of style sheet that provides a simple mechanism for adding styles to an XML to HTML document.A style sheet is a text file contains definitions for elements.

Elementname
{
Property1:value;property 2:value;
}

The CSS file needs to be associated with the XML document for applying formatting specification A CSS can be applied to an XML document using the processing instruction:
<? XML: stylesheet type=”text/css” href=”path-name”?>

In the above statement :
1.XML:stylesheet instructs the browser that the XML document uses a stylesheet.
2.Types specifies the name of the CSS file used to format the XML document.

emp.xml

```
<?xml version=”1.0”?>
<?xml-stylesheet type=”text/css” href=”emp.css”?>
<employees>
<emp>
<id>1001</id>
<name>tarun</name>
<job>officer</job>
<salary>8700</salary>
</emp>
<emp>
<id>1002</id>
<name>Arun</name>
<job>manager</job>
<salary>13000</salary>
</emp>
</employees>
```

emp.css

```
id{display:block;color:red;text-align:center;font-size=20pt;}
name{display:block;color:blue;font-size=15pt;}
job{display:block;colo:green;font-size=12pt;}
salary{display:block;black:font-size=12pt;}
```

1.CSS doesn't support the reorder, sort, and display of elements based on a condition.
2. For such advanced formatting, XML supports eXtensible StyleSheet Language(XSL)

eXtensible Stylesheet Language(XSL)

XSL stands for e**X**tensible **S**tylesheet **L**anguage.The World Wide Web Consortium (W3C) started to develop XSL because there was a need for an XML-based Stylesheet Language.

XSL = Style Sheets for XML

XML does not use predefined tags (we can use any tag-names we like), and therefore the meaning of each tag is **not well understood**.A <table> tag could mean an HTML table, a piece of furniture, or something else - and a browser **does not know how to display it**.XSL describes how the XML document should be displayed!

XSL originally consisted of three parts:

- XSLT - a language for transforming XML documents
- XPath - a language for navigating in XML documents
- XSL-FO - a language for formatting XML documents

. **XSLT** stands for **XSL** Transformations.

- XSLT stands for XSL Transformations
- XSLT is the most important part of XSL
- XSLT transforms an XML document into another XML document
- XSLT uses XPath to navigate in XML documents
- XSLT is a W3C Recommendation

XSLT = XSL Transformations

XSLT is the most important part of XSL.XSLT is used to transform an XML document into another XML document, or another type of document that is recognized by a browser, like HTML and XHTML. Normally XSLT does this by transforming each XML element into an (X)HTML element.

With XSLT you can add/remove elements and attributes to or from the output file. You can also rearrange and sort elements, perform tests and make decisions about which elements to hide and display, and a lot more.

A common way to describe the transformation process is to say that **XSLT transforms an XML source-tree into an XML result-tree**.

XSLT Uses XPath

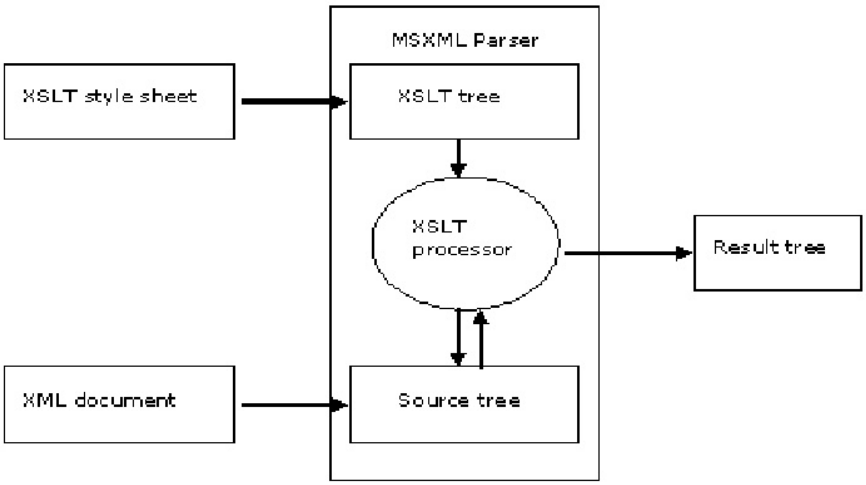
XPath (the XML Path language) is a language for finding information in an XML document.

XSLT uses XPath to find information in an XML document. XPath is used to navigate through elements and attributes in XML documents.

How Does it Work?

In the transformation process, XSLT uses XPath to define parts of the source document that should match one or more predefined templates. When a match is found, XSLT will transform the matching part of the source document into the result document.

Analyzing the working of the XSLT
The XSLT processor applies the transformation information to the source document and builds the result tree as shown in the following figure:



Rendering XML Documents

XSLT <xsl: template> Element

An XSL style sheet consists of one or more set of rules that are called templates.A template contains rules to apply when a specified node is matched.

The <xsl:template> Element

The <xsl:template> element is used to build templates.

The **match** attribute is used to associate a template with an XML element. The match attribute can also be used to define a template for the entire XML document. The value of the match attribute is an XPath expression (i.e. match="/" defines the whole document).

Example

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

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

```
  <h2>My CD Collection</h2>
  <table border="1">
    <tr bgcolor="#9acd32">
      <th>Title</th>
      <th>Artist</th>
    </tr>
    <tr>
      <td>.</td>
      <td>.</td>
    </tr>
  </table>
  </body>
  </html>
</xsl:template>
</xsl:stylesheet>
```

Output:-

My CD Collection

Title	Artist
.	.

Example Explained

Since an XSL style sheet is an XML document, it always begins with the XML declaration: **<?xml version="1.0" encoding="UTF-8"?>**.

The next element, **<xsl:stylesheet>**, defines that this document is an XSLT style sheet document (along with the version number and XSLT namespace attributes).

The **<xsl:template>** element defines a template. The **match="/"** attribute associates the template with the root of the XML source document.

The content inside the <xsl:template> element defines some HTML to write to the output.

The last two lines define the end of the template and the end of the style sheet.

The result from this example was a little disappointing, because no data was copied from the XML document to the output. In the next step you will learn how to use the **<xsl:value-of>** element to select values from the XML elements.

XSLT <xsl:value-of> Element

The <xsl:value-of> element is used to extract the value of a selected node.

The <xsl:value-of> element can be used to extract the value of an XML element and add it to the output stream of the transformation:

Example

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
```

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

<xsl:template match="/">
  <html>
  <body>
  <h2>My CD Collection</h2>
  <table border="1">
    <tr bgcolor="#9acd32">
      <th>Title</th>
      <th>Artist</th>
    </tr>
    <tr>
      <td><xsl:value-of select="catalog/cd/title"/></td>
      <td><xsl:value-of select="catalog/cd/artist"/></td>
    </tr>
  </table>
  </body>
  </html>
</xsl:template>
</xsl:stylesheet>
```

Output:-

My CD Collection

Title	Artist
Empire Burlesque	Bob Dylan

Example Explained

Note: The **select** attribute, in the example above, contains an XPath expression. An XPath expression works like navigating a file system; a forward slash (/) selects subdirectories.

The result from the example above was a little disappointing; only one line of data was copied from the XML document to the output. In the next step you will learn how to use the **<xsl:for-each>** element to loop through the XML elements, and display all of the records.

XSLT <xsl:for-each> Element

The <xsl:for-each> element allows you to do looping in XSLT.

The <xsl:for-each> Element

The XSL <xsl:for-each> element can be used to select every XML element of a specified node-set:

Example

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

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

```
 <html>
 <body>
 <h2>My CD Collection</h2>
 <table border="1">
  <tr bgcolor="#9acd32">
   <th>Title</th>
   <th>Artist</th>
  </tr>
  <xsl:for-each select="catalog/cd">
  <tr>
   <td><xsl:value-of select="title"/></td>
   <td><xsl:value-of select="artist"/></td>
  </tr>
  </xsl:for-each>
 </table>
 </body>
 </html>
</xsl:template>
</xsl:stylesheet>
```

Note: The value of the **select** attribute is an XPath expression. An XPath expression works like navigating a file system; where a forward slash (/) selects subdirectories.

HYPERLINKS IN XML(Xlinks)

XLink is a specification for enhanced and improved linking capabilities that are designed for use with XML documents. XLink is an acronym for XML Linking Language. It allows XML documents to establish a linking relationship between more than one document and create linking documents that reside in a separate location from the linked documents.

In XML, any element can become a source of a link. An element that contains a link is called a linking element. XLinks are of two types : Simple and Extended

Simple Links

* A simple link is similar to the link created in HTML using the ANCHOR element. It connects an element in a source document to a target document.
* Simple links are inline links. An inline link is a part of the element and can connect in one direction.
* A simple link can have only one resource identifier or locator that contains data about the link.
* Simple links have attributes that suggest how a browser or a processor should display and traverse the link. For example, the xlink:show attribute describes how a browser should display the remote resource.
* Simple links allow you to specify whether the remote resource should be displayed in the same browser window, a new window, or within the current document.
* <linkitem xml:link=”simple” href://www.bbc.co.in>click here for the news<linkitem>
  In XML, an element can function as a link by using the xml:link attribute.

Extended Links

* An extended link can point to several resources at a time. It can be used to create links to other documents from your documents, even though you have no write privileges in the other documents.
* Extended links associate an XML element to multiple links.

Extended links are of two types:

1. Out-of-line links-A link that is described by a resource but which is not one of the links in the resource is called an outofline link.
->An example is a catalog document that describes all the links in a Web site without actually participating in any of the links.
2.Multi-directional links-An extended link may also be a multi-directional link that joins several documents in a single link, which can be traversed from any one of its resources. Extended links enable you to specify multiple links from one source.
XLink uses the following attributes for creating simple or extended links:
1. href
2. role
3. title
4. show
5. actuate
6. type

1. The href attribute
(i)A linking element must have a resource locator, that is, the resource targeted by the link.
(ii)The href attribute of XLink is used to specify the target of the link.

2. The role attribute
(i)The role attribute is a means of providing applications with additional information about a link.
(ii)Applications that use XLink can get the information about the role of a link by referring to the value of the role attribute.

3. The title attribute
(i)The title attribute allows you to specify a label to provide information to the user.
(ii)While the role attribute is meant for the system and the application, the title attribute is meant for providing supplementary information to the user.

4. The show attribute
(i)The show attribute can take any one of the three values, namely, embed, replace, or new.
The behaviors represented by these values are described in the table below:

5. The actuate attribute
(i)The actuate attribute allows you to specify the timing of the link.
(ii)The actuate attribute can take one of the following predefined values:

Value	Behavior
embed	If this value is specified, the contents of the linked object will appear embedded within the document from which this link was activated.
replace	If this value is specified, the content of the linked object will replace the document from which the link was activated. This is the default behavior of HTML links.
new	If this value is specified, the content of the linked object will appear separate from the document from which the link was activated, such as in a new browser window.

Value	Behavior
OnRequest	This specifies that the link should be traversed only when the user requests it by clicking on the linking element.
IIMC	k is traversed once the link is loaded. For example, you may set the actuate attribute to OnLoad for an image that is to be embedded in the linking document. The link will be traversed after the image is loaded.

6.The type attribute

(i)The type attribute is used to specify the type of the link that is to be created for an element.
(ii)The value of this attribute can be set to simple, extended, resource, locator, or arc.

*** simple**

->A simple link is similar to an HTML hyperlink. It connects an element in a source document to a target document.

->This means that the link can only be used in one direction, from the source document to the target document.

*** extended**

->When you set the value of the type attribute to extended, it allows you to create multi-directional links between many documents.

*** resource**

->XML allows you to create multi-directional links in which you can have an element that can be both the source as well as the target of a link.

->The source and the target have been given the generic name *resource*.

->A resource can be either local or remote. A local resource is contained inside the extended link element.

*** locator**

->This link type is used to represent remote resources.

->A remote resource exists outside the extended link element.

***arc**

->An arc describes a traversal path between two links.

E.g

```
<? Xml version=”1.0”>
<REFERENCE
Xlink:type=”simple”
Xlink:href=”kk.xml”
Xlink:role=”booklink”
Xlink:title=”XML Document”
Xlink:actuate=”onRequest”>
XML LINKs
</REFERENCE>
```

X-pointer

- *XPointer*, the XML Pointer Language, defines an addressing scheme for the individual parts of an XML document.
- It can be used by any application that needs to identify a part or a location of an XML document.
- XPointer is a new specification designed to link to portions of a document without having to link to the entire document.
- XPointer is a proposal at the W3C that concerns addressing links to specific points within documents.
- X-pointer purposes to define many more relationships other than parent and child. It can also be used to specify relationships between elements that are siblings-the next sibling or the previous sibling.

XML DOCUMENT OBJECT MODEL (DOM)

The DOM is a W3C (World Wide Web Consortium) standard.

The DOM defines a standard for accessing documents like XML and HTML:

"The W3C Document Object Model (DOM) is a platform and language-neutral interface that allows programs and scripts to dynamically access and update the content, structure, and style of a document."

The DOM is separated into 3 different parts / levels:

- Core DOM - standard model for any structured document
- XML DOM - standard model for XML documents
- HTML DOM - standard model for HTML documents

The DOM defines the **objects and properties** of all document elements, and the **methods** (interface) to access them.

The XML DOM is:

- A standard object model for XML
- A standard programming interface for XML
- Platform- and language-independent
- A W3C standard

The XML DOM defines the objects and properties of all XML elements, and the methods (interface) to access them.

In other words: The XML DOM is a standard for how to get, change, add, or delete XML elements.

Using the XML parser

To read and update - create and manipulate - an XML document, you need an XML parser. The Microsoft XML parser is a COM component .XML passer reads the XML document and creates a DOM tree structure that represents the various components of the XML document.

The Microsoft XMLDOM parser features a language-neutral programming model that:

- Supports JavaScript, VBScript, Perl, VB, Java, C++ and more
- Supports W3C XML 1.0 and XML DOM
- Supports DTD and validation

If you are using JavaScript , you can create an XML document object with the following code:

```
var xmlDoc = new ActiveXObject("Microsoft.XMLDOM")
```

If you are using VBScript you create the XML document object with the following code:

```
set xmlDoc = CreateObject("Microsoft.XMLDOM")
```

If you are using VBScript in an Active Server Page (ASP), you can use the following code:

```
set xmlDoc =
Server.CreateObject("Microsoft.XMLDOM")
```

Loading an XML file into the parser

The following code loads an existing XML document (note.xml) into the XML parser:

```
<script language="JavaScript">
var xmlDoc = new
ActiveXObject("Microsoft.XMLDOM")
xmlDoc.async="false"
xmlDoc.load("note.xml")
// ..... processing the document goes here
</script>
```

The first line of the script creates an instance of the Microsoft XML parser. The third line tells the parser to load an XML document called note.xml. The second line assures that the parser will halt execution until the document is fully loaded.

Advantages of XML DOM

1. XML structure is traversable.
   Each node can be randomly accessed(one or more times)by traversing the tree.
2. XML structure is modifiable.
   Since the XML structure is resident in memory, values can be added, changed and removed.
3. The DOM standard is maintained by the world Wide Web Consortium.

Disadvantages of XML DOM

1.Resource intensive:-Since the XML structure is resident in memory,the larger the XML structure is,the more memory it will consume.

2.Relative Speed:- In comparison to SAX(Simple API for XML),DOM can be much slower due to its resource usage/needs.

Uses of XML DOM

1. The XML DOM defines a standard way for accessing and manipulating XML documents
2. The DOM presents an XML document as a tree-structure.
3. Knowing the XML DOM makes working easier with XML.

XML QUERY LANGUAGE

XQL (XML Query Language) is a way to locate and filter the elements (data fields) and text in an Extensible Markup Language (XML) document. XML files are used to transmit collections of data between computers on the Web. XQL provides a tool for finding and/or selecting out specific items in the data collection in an XML file or set of files. It is based on the pattern syntax used in the Extensible Stylesheet Language (XSL) and is proposed as an extension to it.

XML can represent files, graphics, web services etc. As developers store and more information in XML format, they also have an increasing need to be able to search and update XML documents. XML needed a new query language needs to have the expressive power befitting a universal query language for disparate data source.

XQuery is something of a hybrid –a powerful language with grammar similar in SQL, but more widely applicable. XQuery can also transform the content and structure of XML documents. Use XQuery expression such as element and attribute constructors to express the structure of the result document.

Applications of XQuery-

XQuery is timely not just because XML's popularity is on the rise. It's an all-in-one tool that does more for less-perfect for these tough economic times. The remainder of this article presents five application scenarios that use XQuery to solve real problems.

Application 1: Log files Intelligence

Application log files hold a tremendous amount of information. Every day, processes on servers track operations and append information to text files somewhere on the server farm. The log files do not have to be in XML format; the XQuery language is defined on an abstract XML data model.

Application 2: An Executive Dashboard

Keeping tabs on a company's performance is at the top of the priority list for executives these days. That's why so-called executive dashboards are becoming so popular. Building executive dashboards generally involves collecting real-time data from multiple sources and displaying that in a web portal, often in summary or graphic form.

Application 3: Stock Analysis Automation

In the financial world, news is a business tool. If you can find a way to compose news stories automatically from wire sources you can save staff time. In addition, building stories targeted to a specific user or client can increase customer satisfaction. XQuery is well suited to both tasks, because it lets perform targeted full-text search even across structured data sources such as stock prices, and can compose the results into a custom format.

Application 4: Supply Chain Band Aid

One of the costliest areas in a supply chain application is integration. This typically takes the form of incompatible message formats. The buyer has decided to move to something more standard. As a supplier, you agree in principle with open standards, but you have time to move your whole applications to the new standard.

Application 5: Trade Audit Server

Several industries require that trade documents be saved in their original form. Companies often want to hold on to these documents in case disputes arise-showing an actual copy of a trade is proof that it happened. In this scenario, a trading firm keeps all the trading documents in a collection of documents named "tradelogs"

Anyone with only modest knowledge of XML and SQL can learn XQuery relatively quickly. Because XQuery works against many different data types and formats. It's tremendously useful whether you are searching a file system, a relational database, a document management system, a web service or all of these simultaneously.

<u>XML Related Technologies</u>

The following are the technologies that are important to the understanding and development of XML applications.

1.XHTML---Extensible HTML-XHTML is the reformulation of HTML 4.01 in XML. XHTML 1.0 is the latest version of HTML.

2.CSS---Cascading Style Sheets-CSS style sheets can be added to XML documents to provide display information.

3.XSL--- Extensible Style Sheet Language -XSL consists of three parts: XML Documents Transformation (renamed XSLT), a pattern matching syntax (renamed XPath), and a formatting object interpretation.

4.XSLT---XML Transformation-XSLT is far more powerful than CSS. It can be used to transform XML files into many different output formats.

5.XPath---- XML Pattern Matching-XPath is a language for addressing parts of an XML document. XPath was designed to be used by both XSLT and XPointer.

4.XLink----XML Linking Language-The XML Linking Language (XLink), allows elements to be inserted into XML documents in order to create links between XML resources.
5.XPointer - XML Pointer Language-The XML Pointer Language (XPointer), supports addressing into the internal structures of XML documents, such as elements, attributes, and content.
6.DTD - Document Type Definition-A DTD can be used to define the legal building blocks of an XML document.
7.Namespaces-XML namespaces defines a method for defining element and attribute names used in XML by associating them with URI references.
8.XSD - XML Schema-Schemas are powerful alternatives to DTDs. Schemas are written in XML, and support namespaces and data types.
9.XDR - XML Data Reduced-XDR is a reduced version of XML Schema. Support for XDR was shipped with Internet Explorer 5.0 when XML Schema was still a working draft. Microsoft has committed full support for XML Schema as soon as the specification becomes a W3C Recommendation.
10. DOM - Document Object Model-The DOM defines interfaces, properties and methods to manipulate XML documents.
11. XQL - XML Query Language-The XML Query Language supports query facilities to extract data from XML documents.
12.SAX-Simple API for XML-SAX is another interface to read and manipulate XML documents.

XML DTD

1. A **document type definition** (**DTD**) is a set of markup declarations that **define** a document type for an SGML-family markup language (SGML, XML, HTML). A **Document Type Definition** (**DTD**) **defines** the legal building blocks of an XML document. It **defines** the document structure with a list of legal elements and attributes.

Why Use a DTD?

With a DTD, independent groups of people can agree on a standard for interchanging data.

With a DTD, you can verify that the data you receive from the outside world is valid and also use a DTD to verify your own data.

A DTD can be declared inline inside an XML document, or as an external reference.

Internal DTD Declaration
If the DTD is declared inside the XML file, it should be wrapped in a DOCTYPE definition with the following syntax:
<!DOCTYPE root-element [element-declarations]>

Example XML document with an internal DTD:

```
<?xml version="1.0"?>
<!DOCTYPE note [
<!ELEMENT note (to,from,heading,body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
]>
<note>
<to>Tove</to>
<from>Jani</from>
```

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

The DTD above is interpreted like this:

- **!DOCTYPE note** defines that the root element of this document is note
- **!ELEMENT note** defines that the note element contains four elements: "to,from,heading,body"
- **!ELEMENT to** defines the to element to be of type "#PCDATA"
- **!ELEMENT from** defines the from element to be of type "#PCDATA"
- **!ELEMENT heading** defines the heading element to be of type "#PCDATA"

!ELEMENT body defines the body element to be of type "#PCDATA"

"#PCDATA"- parsed character data. XML parsers normally parse all the text in an XML document.When an XML element is parsed, the text between the XML tags is also parsed:
<message>This text is also parsed</message>

External DTD Declaration

If the DTD is declared in an external file, it should be wrapped in a DOCTYPE definition with the following syntax:

<!DOCTYPE root-element SYSTEM "filename">

Example-

```
<?xml version="1.0"?>
<!DOCTYPE note SYSTEM "note.dtd">
<note>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

The file "note.dtd" which contains the DTD:

```
<!ELEMENT note (to,from,heading,body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
```

XML DTD-A DTD defines the legal elements of an XML document. The purpose of a DTD is to define the legal building blocks of an XML document. It defines the document structure with a list of legal elements.

DTD - XML Building Blocks

The main building blocks of both XML and HTML documents are elements.
All XML documents are made up by the following building blocks:

- Elements
- Attributes
- Entities
- PCDATA
- CDATA

Elements

Elements are the **main building blocks** of both XML and HTML documents.
Examples of HTML elements are "body" and "table". Examples of XML elements could be "note" and "message". Elements can contain text, other elements, or be empty. Examples of empty HTML elements are "hr", "br" and "img".
Examples:
<body>some text</body>

<message>some text</message>

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

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

Entities
Some characters have a special meaning in XML, like the less than sign (<) that defines the start of an XML tag.
Most of you know the HTML entity: " ". This "no-breaking-space" entity is used in HTML to insert an extra space in a document. Entities are expanded when a document is parsed by an XML parser.
The following entities are predefined in XML:

Entity References	Character
<	<
>	>
&	&
"	"
'	'

PCDATA
PCDATA means parsed character data.
Think of character data as the text found between the start tag and the end tag of an XML element.
PCDATA is text that WILL be parsed by a parser. **The text will be examined by the parser for entities and markup**.
Tags inside the text will be treated as markup and entities will be expanded.
However, parsed character data should not contain any &, <, or > characters; these need to be represented by the & < and > entities, respectively.

CDATA
CDATA means character data.
CDATA is text that will NOT be parsed by a parser. Tags inside the text will NOT be treated as markup and entities will not be expanded.

DTD - Elements

In a DTD, elements are declared with an ELEMENT declaration.

Declaring Elements

In a DTD, XML elements are declared with the following syntax:

<!ELEMENT element-name category>

or

<!ELEMENT element-name (element-content)>

Empty Elements

Empty elements are declared with the category keyword EMPTY:

<!ELEMENT element-name EMPTY>

Example:

<!ELEMENT br EMPTY>

XML example:

Elements with Parsed Character Data

Elements with only parsed character data are declared with #PCDATA inside parentheses:

<!ELEMENT element-name (#PCDATA)>

Example:

<!ELEMENT from (#PCDATA)>

Elements with any Contents

Elements declared with the category keyword ANY, can contain any combination of parsable data:

<!ELEMENT element-name ANY>

Example:

<!ELEMENT note ANY>

Elements with Children (sequences)

Elements with one or more children are declared with the name of the children elements inside parentheses:

<!ELEMENT element-name (child1)>

or

<!ELEMENT element-name (child1,child2,...)>

Example:

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

When children are declared in a sequence separated by commas, the children must appear in the same sequence in the document. In a full declaration, the children must also be declared, and the children can also have children. The full declaration of the "note" element is:

<!ELEMENT note (to,from,heading,body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>