HTML,XHTML and XML

- XHTML: The Extensible Hypertext Markup Language
- A reformulation of HTML in XML with namespaces for HTML 4.0 strict, transitional and frameset DTDs
- Modularizes HTML for sub setting /combining with other tag-sets

- Whereas HTML was an application of SGML, a very flexible markup language, XHTML is an application of XML, a more restrictive subset of SGML
- The changes from HTML to transitional XHTML are minor, and are mainly to increase conformance with XML.
- The most important change is the requirement that all tags be well-formed.
- XHTML uses *lower-case* for tags and attributes: This is in direct contrast to established traditions which began around the time of HTML 2.0, when most people preferred uppercase tags.
- In XHTML, all attributes, even numerical ones, must be quoted. (This was mandatory in HTML as well, but often ignored.)
- All elements must also be closed, including empty elements such as img and br. This can be done by adding a closing slash to the start tag: and
.
- Attribute minimization (e.g., <option selected>) is also prohibited.

XML

Stands for e**X**tensible **M**arkup **L**anguage

- It was designed to carry data, not to display data.
- In XML, Tags are not predefined.
- XML is just a plain text.
- XML is not a replacement for HTML.

Possible Advantages of Using XML

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

XML v/s DATABASE

- XML is faster
- Solution for application using data; Avoids any dependency of your application to any other DBMS s/w.

1.HTML was designed **to display data with focus on how data looks** while XML was designed to be a software and hardware independent tool used to **transport and store data, with focus on what data is**.

2.HTML is a **markup language** itself while XML provides **a framework for defining markup languages**.

3.HTML is a **presentation language** while XML is **neither a programming language nor a presentation language**.

4.HTML is **case insensitive** while XML is **case sensitive**.

5.HTML is **used for designing a web-page** to be rendered on the client side while XML is used basically to **transport data** between the application and the database.

6.HTML has it **own predefined tags** while what makes XML flexible is that **custom tags** can be defined and the tags are invented by the author of the XML document.

7.HTML is **not strict** if the user does not use the **closing tags** but XML makes it **mandatory** for the user the close each tag that has been used.

8.HTML does **not preserve white space** while XML **does**.

9.HTML is about displaying data,hence **static** but XML is about carrying information,hence **dynamic**.

- there are 3 components for XML content:
  - the XML document
  - DTD (Document Type Declaration)
  - XSL (Extensible Stylesheet Language)
- The DTD and XSL do not need to be present in all cases

A well-formed XML document

- elements have an open and close tag, unless it is an empty element
- attribute values are quoted
- if a tag is an empty element, it has a closing / before the end of the tag
- open and close tags are nested correctly
- there are no isolated mark-up characters in the text (i.e. < > &]]>)
- if there is no DTD, all attributes are of type CDATA by default

A valid XML document

- has an associated DTD and complies with the constraints in the DTD

XML basics

- <?xml ?> the XML declaration

- not required, but typically used

- attributes include:

  version

  encoding – the character encoding used in the document

  standalone –if an external DTD is required

<?xml version=”1.0” encoding=”UTF-8”>

<?xml version=”1.0” standalone=”yes”>

XML basics

- <!DOCTYPE …> to specify a DTD for the document

  2 forms:

  <!DOCTYPE *root-element* SYSTEM “*URIofDTD*”>

  <!DOCTYPE *root-element* PUBLIC “*name*” “*URIofDTD*”>

XML basics

- <!-- --> comments

- contents are ignored by the processor
- cannot come before the XML declaration
- cannot appear inside an element tag
- may not include double hyphens

XML basics

- <tag> text </tag> an element

  - can contain text, other elements or a combination

  - element name:

    -must start with a letter or underscore and can have any number of letters, numbers, hyphens, periods, or underscores

  - **case-sensitive**;

  - may not start with *xml*

XML basics

Elements (continued)

- can be a *parent, grandparent, grandchild, ancestor,* or *descendant*
- each element tag can be divided into 2 parts
  – *namespace*:*tag name*

XML basics

- Namespaces:

  - not mandatory, but useful in giving uniqueness to an element

  - help avoid element collision

  - declared using the xmlns:*name*=*value* attribute; a URI is recommended for *value*

  - can be an attribute of any element; the scope is inside the element's tags

XML basics

- Namespaces (continued):

  - may define more than 1 per element

  - if no name given after xmlns prefix, uses the default namespace which is applied to all elements in the defining element without their own namespace

  - can set default namespace to an empty string to ensure no default namespace is in use within an element

XML basics

- key=”value” an attribute
  - describes additional information about an element

<tag key=”value”> text</tag>

- value must always be quoted
- key names have same restrictions as element names
- reserved attributes are
  - xml:lang
  - xml:space

XML basics

- <tag></tag> OR <tag/> empty element

  - has no text

  - used to add nontextual content or to provide additional information to parser

- <? ?> processing instruction

  - for attributes specific to an outside application

XML basics

- <![CDATA[ ]]>

  - to define special sections of character data which the processor does not interpret as markup

  - anything inside is treated as plain text

```
<BOOKS>
<book id="123" loc="library">
  <author>Hull</author>
  <title>California</title>
  <year> 1995 </year>
</book>
<article id="555" ref="123">
  <author>Su</author>
  <title> Purdue</title>
</article>
</BOOKS>
```

Why Use a DTD?

- With a DTD, each of your XML files can carry a description of its own format.
- With a DTD, independent groups of people can agree to use a standard DTD for interchanging data.
- Your application can use a standard DTD to verify that the data you receive from the outside world is valid.
- You can also use a DTD to verify your own data.

DTD

XML File:

```
<?xml version="1.0"?>
 <!DOCTYPE note SYSTEM "note.dtd">
 <note>
  <to>John</to>
  <from>Smith</from>
  <heading>Reminder</heading>
  <body>Meeting</body>
 </note>
```

note.dtd:

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

Declaring an element

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

Eg:

For the ‘to’ Element in the XML file the DTD can be:

<! ELEMENT to #PCDATA>

For the ‘note’ Element in the XML file the DTD can be:

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

How to tackle many occurrences?

```
<note>
 <to>ABC</to>
 <from>XYZ</from>
 <heading>Reminder</heading>
 <body>cnbmvnbmn</body>
  .
  .
  .
  .
  .
  .
  .
  .

 <to>LMN</to>
 <from>PQR</from>
 <heading>Reminder</heading>
 <body>mbvnbmvnb</body>
</note>
```

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

Some regular expressions:

* Zero or more

\+ 1 or more

| OR

? Zero or 1

DTD – Attributes

General Syntax:

<!ATTLIST element-name attribute-name attribute-type default-value>

DTD example:

<!ATTLIST payment type CDATA "check">

XML example:

<payment type="check" />

More on Attributes Types

Type	Description
CDATA	The value is character data
(*en1*|*en2*|..)	The value must be one from an enumerated list
ID	The value is a unique id
IDREF	The value is the id of another element
IDREFS	The value is a list of other ids

ID , IDREF ,IDREFS

```
<bank>
        <customer ID="C101" IDREFS="A101 A102" >    </customer>
        <customer ID="C102" IDREF="A102" >   </customer>

        <account ID="A101" IDREF="C101">   </account>
        <account ID="A102" IDREFS="C101 C102">   </account>
</bank>
```

- *This means Customer with ID C101 has two accounts A101 and A102 and he shares a joint account with C102 for the account A102.*

More on Attribute values

Value	Explanation
value	The default value of the attribute
#REQUIRED	The attribute is required
#IMPLIED	The attribute is not required
#FIXED *value*	The attribute value is fixed

```
<!DOCTYPE notedetails[
<!ELEMENT notedetails (note+)
<!ELEMENT note((to,from,heading,body) >
<!ELEMENT to (#PCDATA)>
 <!ELEMENT from (#PCDATA)>
 <!ELEMENT heading (#PCDATA)>
 <!ELEMENT body (#PCDATA)>

 <!ATTLIST note DATE CDATA #REQUIRED>
 <!ATTLIST note TIME CDATA #IMPLIED>

 ]>
```

DTD Limitations

- Individual text elements cannot be further typed.
- There is lack of typing in IDs and IDREFs.

- DTD (Document Type Definition) (.dtd)
- Specifies a relatively simple syntax for a set of XML files (validation criteria)
- Can't specify complex relationships or data formats
- Does not support XML namespaces
- The DTD is non-heirarchical (flat, linear)
- DTD is not an XML language (which makes it more difficult to parse or transform)

? Optional (zero or one)

* Zero or more

+ One or more

#PCDATA Parsed character data

#CDATA Unparsed character data

(#PCDATA [| element]*)* Mixed-content mode; any number of elements can be interspersed with text

#REQUIRED The attribute value must be specified in the document.

#IMPLIED The value need not be specified in the document. If it isn't, the application will have a default value it uses."defaultValue" The default value to use, if a value is not specifiedin the document.

#FIXED "fixedValue" The value to use. If the document specifies any value at all, it must be the same.

DTD

- A Document Type Definition (**DTD**) allows the developer to create a set of rules to specify legal content and place restrictions on an XML file
- If the XML document does not follow the rules contained within the DTD, a parser generates an error
- An XML document that conforms to the rules within a DTD is said to be **valid**

Why Use a DTD?

- A single DTD ensures a common format for each XML document that references it
- An application can use a standard DTD to verify that data that it receives from the outside world is valid
- A description of legal, valid data further contributes to the interoperability and efficiency of using XML

Some Example DTD Declarations

- Example 1: The Empty Element

```
<!ELEMENT Bool (EMPTY)> <!--DTD declaration of empty
   element-->
<Bool Value="True"></Bool> <!--Usage with attribute in XML
   file-->
```

- Example 2: Elements with Data

```
<!ELEMENT Month (#PCDATA)> <!--DTD declaration of an element->
<Month>April</Month> <!—Valid usage within XML file-->

<Month>This is a month</Month> <!—Valid usage within XML file-->
<Month> <!—Invalid usage within XML file, can't have children!-->
<January>Jan</January>
<March>March</March>
</Month>
```

Some Example DTD Declarations

- Example 3: Elements with Children

  To specify that an element must have a single child element, include the element name within the parenthesis.

```
<!ELEMENT House (Address)> <!—A house has a single address–>
<House> <!—Valid usage within XML file–>
<Address>1345 Preston Ave Charlottesville Va 22903</Address>
</House>
```

  An element can have multiple children. A DTD describes multiple children using a *sequence,* or a list of elements separated by commas. The XML file must contain one of each element in the specified order.

```
<!--DTD declaration of an element–>
  <!ELEMENT address (person,street,city, zip)>
  <!ELEMENT person  (#PCDATA)>
  <!ELEMENT street  (#PCDATA)>
  <!ELEMENT city    (#PCDATA)>
  <!ELEMENT zip     (#PCDATA)>
  <!—Valid usage within XML file–>
  <address>
  <person>John Doe</person>
  <street>1234 Preston Ave.</street>
  <city>Charlottesville, Va</city>
  <zip>22903</zip>
  </address>
```

Other way is
XSLT

- Xsl : extensible stylesheet language.
- Xslt : xsl transform.

(language for xml transformation)

XSLT is a language for transforming XML documents into XHTML documents or to other XML documents.

- **CSS = Style Sheets for HTML**
  - HTML uses predefined tags, and the meaning of each tag is **well understood**.
  - The <table> tag in HTML defines a table - and a browser knows **how to display it**.
  - Adding styles to HTML elements are simple. Telling a browser to display an element in a special font or color, is easy with CSS.

- **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!

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**.

- **Correct Style Sheet Declaration**
- The root element that declares the document to be an XSL style sheet is <xsl:stylesheet> or <xsl:transform>.

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

or

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

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 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).

XSLT <xsl:value-of> Element

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

Book.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml-stylesheet type="text/xsl"
   href="simplexslt.xsl"?>
<CATALOG>
   <BOOK>
    <TITLE>Empire Burlesque</TITLE>
    <AUTHOR>Bob Dylan</ARTIST>
    <COUNTRY>USA</COUNTRY>
    <COMPANY>Columbia</COMPANY>
    <PRICE>10.90</PRICE>
    <YEAR>1985</YEAR>
   </BOOK>
<CATALOG>
```

```
<?xml version="1.0" encoding="ISO-8859-1"?>
  <xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:template match="/">
   <html>
   <body>
   <h2>My Collection</h2>
   <table border="1">
    <tr>
     <th>Title</th>
     <th>author </th>
    </tr>
    <tr>
     <td><xsl:value-of select="catalog/book/title"/></td>
     <td><xsl:value-of select="catalog/book/author"/></td>
    </tr>
   </table>
   </body>
   </html>
  </xsl:template>

  </xsl:stylesheet>
```

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

```
<?xml version="1.0" encoding="ISO-8859-1"?>
    <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>
```

- **The <xsl:if> Element**
- To put a conditional if test against the content of the XML file, add an <xsl:if> element to the XSL document.
- **Syntax**
- <xsl:if test="*expression*">
  ...some output if the expression is true...
  </xsl:if>

- <?xml version="1.0" encoding="ISO-8859-1"?>
  <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">
  <xsl:if test="price > 10">
  <tr>
  <td><xsl:value-of select="title"/></td>
  <td><xsl:value-of select="artist"/></td>
  </tr>
  </xsl:if>
  </xsl:for-each>
  </table>
  </body>
  </html>
  </xsl:template>
  </xsl:stylesheet>