XML

XML:- Extensible Markup Language

Markup:- Enclosing the textual information in between two - Tag is called Markup Language

Example:-

opening Tag ↑ ↗ closing Tag

```
<empName> raja </empName>
<empSalary> 5000 </empSalary>
```

Extensible:- XML Tags are user Defined Tags. So These Tags - functionality also defining by user. So These Tags - functionality Possible to Extened as per the application requirement. because of this reason only it is called as Extensible Markup Language.

HTML is a HyperText Markup Language. HTML Tags functionality Limited because these Tags are Predefined Tags.

Language:- To Exchange the data client to Server and Server to client always we are using XML formate. so To Provide the Communication between one Language application to another Language application also always we are using XML formate only. So its a Mediator between to Language. So its called the Language.

Difference Between XML and HTML

XML	HTML
• XML Tags are user defined Tags	• HTML Tags are define Defined Tags
• XML Tags functionality defining by user and Extensible	• HTML Tags functionality Limited because HTML Tags are Predefined
• XML Tags are Case-sensitive	• HTML Tags are Not Case-Sensitive
• XML Use to define the data	• HTML Use To display the data.

- XML is given by W3C and it is having only 1.0 Version
- XML describe the data
- ~~Possible~~ XML documents possible to maintain text based data bases.
- XML documents, we are using to transport the data one app to another app.

Example :-

File:- Emp.xml

```
<employee>
<empNo> 1001 </empNo>
<empName> Rama </empName>
<empSalary> 9000 </empSalary>
</employee>
```

- How to view the XML documents ?

To view the XML Document we can use following things.

1. Browser Software (ie, chrome, mozilla)
2. XML editor Software.

While opening emp.xml file in Browser then it will look like below

```
<?XML version = "1.0" ?>
<employee>
<empNo> 1001 </empNo>
<empName> Rama </empName>
<empSalary> 9000 </empSalary>
</employee>.
```

- It is called Prolog
- It is automatically generated by Browser
- Prolog always start with ?XML
- It is representing the XML-version

- It is the well defined structure of XML document.
- If we are doing any mistake in the XML then it will not ~~show~~ show in Browser in Proper format.

XML Documents Contains

- Elements
- Attributes
- Entity References

Elements: Opening Tag to Closing Tag including Text Data is called an Elements.

```
<employee>
<empNo> 101 </empNo>
<name> rama </name>
<salary> 9000 </salary>
</employee>
```

here having the 4 Elements.

- Elements Contains other Elements that is called Child Elements
- Element Contains ~~Child~~ Attributes
- Element Contains Text-data
- Element Contains mix of all

Attributes: — attributes describes the extra information about the elements.

```
<employee empNo = "101" name = "rama" >
</employee>
```

- Attribute values must be enclose with either single quotes (OR) double quotes.
- Attribute can not be duplicate in an element.
- Attributes order is not important in an Element.

Entity References: — In XML some symbol/special character having specific meaning. So that special character we are using in text data, syntactically that are considering as mistake.

Example:—

```
<person>
<name> rama </name>
<age> the person age is < 18 </age>
</person>
```

this symbol is illegal here and it is showing as opening tag.

Instead of that '<' symbol we can use <. and this EntityReference automatically convert in < symbol.

Example :-

```
<Person>
<name> rama </rama>
<age> the Person age is &lt; 18 </age>
</Person>
```

- Output while opening this in browser.

```
<?xml version="1.0"?>
<Person>
<name> rama </name>
<age> the Person age is < 18 </age>
</Person>
```

- XML provide 5 Predefine EntityReferences

<	<	illegal to use syntetically
>	>	
&	&	legal to use but recommended to use EntityReferences
'	'	
"	"	

- To use EntityReferences the syntex is given below

&EntityReferencesName;

↓ here we can give our name means we can create our own EntityReferences.

- XML Dolument is wellformed Document. It must began with following things.

--> PROLOG
--> must be maintain an unique root element.
--> Must be maintain closing tags for all the opening tags.
--> All the XML elements we can write with Proper Case (Because the XML elements are case Sensitive
--> In the place Some ~~critical~~ special characters We can use an Entity References.

DTD

DTD → Document Type Defination

DTD → is a XML technique used to define the structure of a XML document.

DTD → is a text based document with .dtd extension

DTD → Contains Elements Declarations,
Attribute Declarations,
EntityReferences Declarations

Element Declaration Syntax :-

```
<!ELEMENT Element-Name (content-model)>
```

Example :-

```
<!ELEMENT employee (empno, name, salary)>
<!ELEMENT empno (#PCDATA)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT salary (#PCDATA)>
```

PCDATA → Parsable character Data

- XML :- Writing XML for above DTD

```
<employee>
<empno> 101 </empno>
<name> rama </name>
<salary> 18000 </salary>
</employee>
```

- In the content-model five types of elements declarators are possible
  1 → Text-only Element
  2 → Child-only Element
  3 → Empty Element
  4 → Any Element
  5 → Mixed Element

- In the above XML empno, name and salary elements are Text-only Element because these are having Text Data like (Numeric., string). and for defining the Text only Element in DTD we Define the #PCDATA Data-Type. That is Define in above DTD Example.
- The above XML employee element is child only Elment because it is having empno, name and salary as a child element. and same we define in DTD in above DTD Example.

How to link the DTD with XML?

To map the DTD with XML in XML Document we can use <!DOCTYPE> declaration

Types of DTD's

1) Internal DTD's (this DTD always Define in XML file only)
2) External DTD's (This DTD always Define seprately means DTD file and XML file will be seperately)

Example of Internal DTD's

```
<!DOCTYPE employee [
<!ELEMENT employee (empno, name, salary)>
<!ELEMENT empno (#PCDATA)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT salary (#PCDATA)>
]>
<employee>
<empno> 101 </empno>
<name> rama </name>
<salary> 10000 </salary>
</employee>
```

This is Internal DTD's

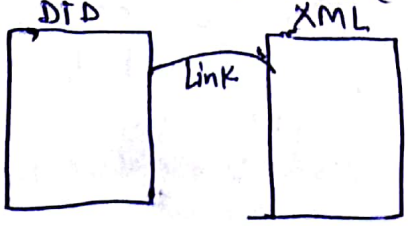
Syntax for Defining Internal DTD's

```
<!DOCTYPE rootElement [DTD rules]>
  XML Element
```

If we are Defining Internal DTD then this DTD's will be specific to the same XML file where we are defining. It will not be reusabled.

EXTERNAL DTD's

here we define the DTD and XML seperately and we will link the DTD with XML file. To Linking the DTD TO XML we use the <!DOCTYPE>



TYPE OF EXTERNAL DTD

1) Private DTD's
2) Public DTD's

1) Private DTD's :- It is specific to a particular project.

```
<!DOCTYPE rootElement SYSTEM "DTDFILENAME.dtd">
```

Example

employee.dtd

```
<!ELEMENT employee (empNo, name, salary)>
<!ELEMENT empNo (#PCDATA)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT salary (#PCDATA)>
```

empoloyee.XML

```
<!DOCTYPE employee SYSTEM "G://XML/employee.dtd">
  <employee>
  <empNo> 101 </empNo>
   <name> Sathish </name>
   <salary> 12000 </salary>
   </employee>
```

2) Public DTD's :- It is not specific to particular project. It is common to all project. Ex- HIBERNATE, SPRING

```
<!DOCTYPE rootElement PUBLIC
    "-//Vendorname//version//EN" "DTDFILENAME.dtd">
```

These things are optional. It is not mandatory to define but "" need to define mandatory.

Example:-

employee.dtd (it is define above)

employee.xml

```
<!DOCTYPE employee PUBLIC
    "-//nareshit//1.0//EN" "employee.dtd">
<employee>
<empNo> 101 </empNo>
<name> sathish </name>
<salary> 12000 </salary>
</employee>
```

Example DTD

both are mandatory to define

```
<!ELEMENT book (book-name , title)>
<!ELEMENT book-name (#PCDATA)>
<!ELEMENT title (#PCDATA)>
```

XML

①
```
<book>
<book-name> CoreJava </book-name>
</book>
```

②
```
<book>
< title> Core Java </title>
</book>
```

③
```
<book>
<book-name> CoreJava </ book-name>
< title > CoreJava </ title>
</ book>
```

in above DTD we are using , to separating the child Element means both Element is mandatory to define while defining XML. So in above XML ① and ② are having errors. because there we did not define both child Element. but ③ XML is fine for above DTD.

But if we use | instead of , then we can define either <book-name> or <title> child element. means ① and ② XML are fine for above DTD but ③ XML will be wrong.

→ either book-name OR title not both to define

```
<!ELEMENT book ( book-name | title ) >
<!ELEMENT book-name (#PCDATA) >
<!ELEMENT title (#PCDATA)>
```

Cardinality Operators :-

D (5)

in XML Document Document can occur 0 to n number of times means one XML Element can occurs 0 to n number of times in XML Document, so to specify How many number of times an Element can occur in a XML Document we can work with what cardinality operators.

in DTD we have only 3 type of cardinality operators.

① * → 0 to n
② + → 1 to n
③ ? → 0 OR 1

Example :

DTD

```
<!ELEMENT books (book*)>
<!ELEMENT book ('(book-name | title), Price,
                 author+, Publication?)
<!ELEMENT book-name (#PCDATA)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT Price (#PCDATA)>
<!ELEMENT author (#PCDATA)>
<!ELEMENT Publication (#PCDATA)>
```

Write the XML for above DTD

```
<!DOCTYPE books SYSTEM "G:// XML / books.dtd">
<books>
<book>
<book-name> Core Java </book-name>
< Price > 500 </Price>
< author> Sathish </authors>
<author> Jhon </author>
< publication> NIT </Publication>
</book>
</books>
```

TYPES OF ELEMENTS

1) Text-only Elements
2) Child only Elements
3) Mixed Elements
4) EMPTY Elements
5) ANY Elements

1) Text-only Elements

To Declare Text-only Elements allways we are using (#PCDATA) as a Data type in the Content Model.

Example:

```
<!ELEMENT empname (#PCDATA)>
<!ELEMENT empSalary (#PCDATA>
```

2) Child-only Elements

If any elements allows only the child Elements then this type of elements call the child-only Elements.

Example:- DTD

```
<!ELEMENT employee (empname, empsalary)>
<!ELEMENT empname (#PCDATA)>
<!ELEMENT empsalary (#PCDATA)>
```

XML

```
<!DOCTYPE employee SYSTEM "G://emp.dtd">
<employee> ——→ child only element
<empname> Raja </empname> — Text-only Elemt
<empsalary> 12000 </empSalary> — Text-only elemt
</employee>
```

3) MIXED Elements :-

If any Element allows Chid Element and Text Data then this type of element is called MIXED Elements.

Example :-

<employee>
the employee name is <empname> raja </empname>
and employee salary is <empsalary> 12000 </empsalary>
</employee>

How to Declare MIXED Elements in DTD ?

DTD

```
<!ELEMENT employees (employee*)>
<!ELEMENT employee (#PCDATA | empname | empsalary)*>
<!ELEMENT empname (#PCDATA)>
<!ELEMENT empsalary (#PCDATA)>
```

XML

```
<!DOCTYPE employees SYSTEM "G://employee.dtd">
<employees>
<employee>
 the employee name is
<empname> raja </empname> and employee
 salary is <empsalary> 12000 </empsalary>
</employee>
</employees>
```

4) EMPTY Elements :-

Empty Elements means if any Element not allow text data and not allow child Elements then that type of Elements is called Empty Elements. EMPTY Element may contain Attributes.

best example in html
 and <hr /> not contain any child elements and not contain any text data so these are EMPTY ELEMENTS

Example :- DTD :-

```
<!DOCTYPE employees [
<!ELEMENT employees (employee*)>
<!ELEMENT employee EMPTY>
]>
```

XML:-

```
<employees>
<employee></employee>
<employee/>
</employees>
```

→ we can not provide the space here if we are giving any space between opening & closing tag then it consider that space as a character. Instead of providing opening & closing element we can provide like this <employee />.

⑤ ANY ELEMENT :-

If an element allows text data / child element / mixed content / EMPTY content. then this type of Element is called ANY ELEMENT.

Example:-

```
<!DOCTYPE employees [
 <!ELEMENT employees (employee+)>
 <!ELEMENT employee &ANY>
 <!ELEMENT empname (#PCDATA)>
 <!ELEMENT empsalary (#PCDATA)>
 ]>
<employees>
<employee>
the empname is <empname> raja </empname>
</employee>
<employee> this is Employee Information </employee>
<employee> </employee>
</employee>
 <empsalary> 12000 </empsalary>
</employee>
</employees>
```

Attributes :-

- Attributes is a name value pair.
- Attributes are used provide some extra information of an element.
- Attributes order is not important in an element
- Attributes are unique in an element.
- Attributes values must be enclose with either single quotes (OR) double quotes.

* To declare the Attribute in dtd use the following Syntax :-

<!ATTLIST element-name attrName attrDatatype attrSpecifier other-information>

DTD :-

```
<!ELEMENT employees (employee+)>
<!ELEMENT employee EMPTY>
<!ATTLIST employee empno CDATA #REQUIRED>
<!ATTLIST employee name CDATA #REQUIRED>
<!ATTLIST employee salary CDATA #REQUIRED>
```

employees.xml

```
<employees>
<employee empno='101' name='ajay' salary='12000'/>
<employee empno='102' name='vijay' salary='13000'/>
</employees>
```

attribute specifier :-

used to specify an attribute is a.

- mandatory attribute
- optional attribute
- fixed attribute
- default attribute

1) #REQUIRED ---> mandatory attribute
2) #IMPLIED ---> optional attribute
3) #FIXED ---> fixed attribute

any specifier if we are not using explicitly then that attribute is default.

4) default ---> default attribute.

```
<!ELEMENT Students (Student+)>
<!ELEMENT student EMPTY>
<!ATTLIST student studentid CDATA #REQUIRED>
<!ATTLIST student studentname CDATA #IMPLIED>
<!ATTLIST student studentCourse CDATA
                                "Java">
<!ATTLIST student studentfee CDATA #FIXED "1000">
```

Students.xml

```
<students>
<student studentid = "101" StudentCourse = "C"
                    Studentfee = "1000">
<student studentid = "102" studentname = "RaJa" />
<student studentid = "103" studentCourse = "Java"
                    Studentfee = "1000" />
</students>
```

attributes datatypes :-

- CDATA
- ENUMERATED
- ID
- IDREF
- IDREFS
- ENTITY
- ENTITIES
- NOTATION
- NMTOKEN
- NMTOKENS

• CDATA ----> Character Data

If any Attribute datatype is CDATA then we can use the Attribute value as any character. there is no restriction for CDATA.

empno = "101"
empname = "XXX"
Salary = "3000"

Example -

```
<!ELEMENT employees (employee+)>
<!ELEMENT employee EMPTY>
<!ATTLIST employee empno CDATA #REQUIRED>
<!ATTLIST employee empname CDATA #REQUIRED>
<!ATTLIST employee salary CDATA #REQUIRED>
```

• ENUMERATED :- DTD:-

```
<!DOCTYPE Payments [
<!ELEMENT Payments (Payment+)>
<!ELEMENT payment EMPTY>
<!ATTLIST Payment Payment-type (DD|CHEQUE|CASH)
                  #REQUIRED>
]>
```

If Datatype is Enumerated then the attribute allows any one of the value from the specified list.

So in above DTD Payment-type is a Enumerated type So the Payment-type value Either DD (OR) CHEQUE (OR) CASH.

So we can say anywhere if we have a requirment to allow only Particular value from the list then we can go for enumerated.

XML for above DTD:-

```
<Payments>
<Payment payment-type="DD"/>
<Payment payment-type="CHEQUE"/>
</Payments>
```

How to created Enumerated attribute

(Value1 | value2 | value3 .....)

ID:-

- for ID TYPE attributes value is unique in XML
- ID TYPE value should not start with digit and -.
- We can start with _ (OR) A-Z a-z
- ID Type allows _ - A-Z a-z 0-9.

Example:-

```
<!DOCTYPE students [
<!ELEMENT students (student+)>
<!ELEMENT student EMPTY>
<!ATTLIST student sid ID #REQUIRED>
<!ATTLIST student sname CDATA #REQUIRED>
]>

<students>
<student sid = "NIT101" sname = "ajay">
<student sid = "NIT102" sname = "vijay">
</students>
```

- This sid should be unique for every student. otherwise we will get error like give id value already been used.
- this value should be start with _ (OR) A-Z a-z. but we can't start with digit and - (hightun)
- ID Types allows _ - A-Z a-z 0-9

IDREF:-

one Element ID attribute value if we want to store into another element attribute then we can use IDREF datatype.

Example:-

```
<!DOCTYPE students [
<!ELEMENT students (course*, student*)>
<!ELEMENT course EMPTY>
<!ATTLIST course cid ID #REQUIRED>
<!ATTLIST course cname CDATA #REQUIRED>
<!ATTLIST course cfee CDATA #REQUIRER>
<!ELEMENT student EMPTY>
<!ATTLIST student sid ID #REQUIRED>
```

```
<!ATTLIST Student Sname CDATA #REQUIRED>
<!ATTLIST student Scourse CDATA #IDREF #REQUIRED>
]>
<students>
<course Cid = "CJ201" Cname = "coreJava" Cfee = "500"/>
<course Cid = "AJ202" Cname = "AdvJava" Cfee = "1500"/>
<course Cid = "S203" Cname = "Spring" Cfee = "750"/>
<student Sid = "NIT101" Sname = "rama"
         Scourse = "CJ201"/>
<student Sid = "NIT102" Sname = "raja"
         Scourse = "AJ202" />
</students>
```

* In the above example in course element Cid is the ID datatype and in student element Scourse attribute is IDREF Data type which is having the Reference of ID Cid attribute. means Scourse is storing the some other element attribute and some other element attribute must be ID Data type.

IDREFS:-

It is similar to IDREF only but difference is IDREF type attribute allow one value and IDREFS attribute allow List of value or collection of values.

Example:-

```
<!ELEMENT students (course*, student*)>
<!ELEMENT course EMPTY>
<!ATTLIST Course cid ID #REQUIRED>
<!ATTLIST Course cname CDATA #REQUIRED>
<!ATTLIST course cfee CDATA #REQUIRED>
<!ELEMENT student EMPTY>
<!ATTLIST student sid ID #REQUIRED>
<!ATTLIST student Sname CDATA #REQUIRED>
<!ATTLIST student Scourse IDREF #IMPLIED>
<!ATTLIST student Scourses IDREFS #IMPLIED>
]>
<students>
<course cid = "CJ201" cname = "coreJava "efee="500">
```

```
<Course Cid = "AJ202" Cname = "advjava" Cfee = "1500" />
<Course Cid = "S203" Cname = "Spring" Cfee = "500" />
<Student Sid = "NIT101" Sname = "Rama"
        Scourse = "CJ201" />
<Student Sid = "NIT102" Sname = "Raja"
        Scourse = "AJ202" />
<Student Sid = "NIT103" Sname = "Ajay"
        Scourses = "CJ201 S203" />
</Students>
```

Scourses is type of IDREFS: Data type so here we are providing collection of value with will be separated with space like "CJ201 S203".

Summerizing the ID, IDREF and IDREFS

ID:- attribute type value must be unique in the XML

IDREF:- attribute stores same another element ID attribute type value.

IDREFS:- Similar to IDREF but IDREF stores one value ~~but~~ and IDREF stores list of values.

NMTOKEN:-

If any Attribute Datatype is NMTOKEN the Attribute value always a valid XML.

CDATA and NMTOKEN almost same but when attributedata type is CDATA there are No Restriction for ~~attribute~~ ~~value~~ acceptable value. but if attribute datatype is NMTOKEN the value Must be a valid XML.

Valid XML means it allow following things.

- A-Z a-z
- 0-9
- – (hiphen)
- _ (underscore)
- . (dot)

} allow

+ ! @ # $ } not allow

Example:-

```
<!DOCTYPE Courses [
<!ELEMENT Courses (course)+>
<!ELEMENT Course EMPTY>
<!ATTLIST Course Cid ID #REQUIRED>
<!ATTLIST Course Cname NMTOKEN #REQUIRED>
]>
<Courses>
<Course Cid="CJ101" Cname="Java"/>
<Course Cid="AJ102" Cname="AdvJava"/>
<Course Cid="C103" Coname="C"/>
</Course
```

here due to Cname is NMTOKEN we can not give the value C++ OR C#. It will not allow with NMTOKEN. It should be valid XML but if we want to allow C++ (OR) C# then we have to define Cname Attribute data type as CDATA.

NMTOKENS :-

NMTOKEN and NMTOKENS Both are same but there is a one difference that NMTOKEN allow the Single value but NMTOKENS allow the list of values. If we want to represent the single value then we can go for NMTOKEN. If we want to represent the List of values then we can go for NMTOKENS.

NOTATION :- If we want to work with a shortname for this media type. then we can use NOTATION Data type.

```
<photo phototype="image/JPeg"/>
<photo phototype="image/gif"/>
```

for image/JPeg or image/gif we can create the Notation and we can use that notation instead of the full name.

image/gif ---> gif image/JPeg -> JPG

image/gif ---> gif
image/jpeg ---> jpg
application/json ---> json

These are the shortcut for the media type which is created by Notation.

* How to create the Notation ?

<!NOTATION NOTATION_NAME SYSTEM "IMAGE/JPEG">

→ its the type

- SYSTEM represent the private Notation.
- PUBLIC represent the public Notation.

Example :-

```
<!DOCTYPE photos [
<!ELEMENT photos (photo+)>
<!ELEMENT photo (#PCDATA)>
<!ATTLIST photo phototype NOTATION (JPG|GIF)
                          #REQUIRED>
]>
<!NOTATION JPG SYSTEM "image/jpeg">
<!NOTATION GIF SYSTEM "image/gif">
<photos>
<photo phototype="GIF"> this is mobile
        image </photo>
<photo phototype="JPG"> this is Nature
        image:</photo>
</photos>
```

* When we are working with NOTATION TYPE the corresponding Element Content-model should not be Empty.

So in above Example <photo> Element is define as #PCDATA that is allowing the text data type.

ENTITY :-

Entity is possible to declare for PCData as well as for CDATA.

Element Content --> PCDATA (Parsable Data)
Attribute Content --> CDATA (UnParsable Data)

ENTITY Creation for Parsed Contents :-

<!ENTITY entity-name "requiredvalues">

<!ENTITY nit "Nareshit">

*Entity is acting as a shortcut for long length repeated string. means if we are using one long length string multiple place in XML then it will increase the burden instead of this we can use Entity. to define the long length string at multiple place.

Above we created one Entity with nit then this Entity we can use with below type.

Syntax:- &EntityReferenceName;

&nit;

Example:-

```
<!DOCTYPE Courses [
  <!ENTITY nit "NARESHIT">
  <!ELEMENT Courses (Course, inst)*>
  <!ELEMENT Course (#PCDATA)>
  <!ELEMENT Inst (#PCDATA)>
  ]>
<Courses>
<Courses Java </course>
< Inst > &nit </ Inst>
</Courses>
```