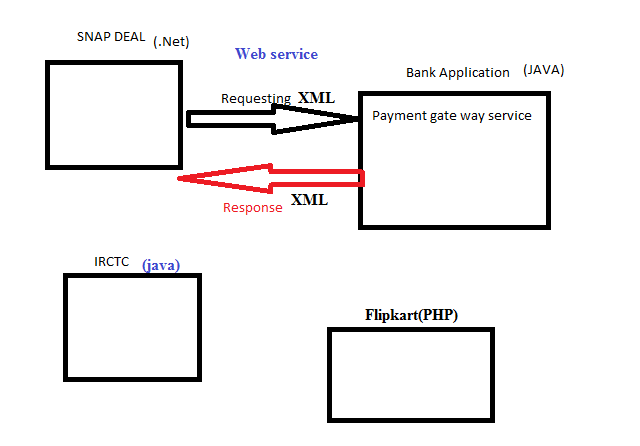
**XML/Web Services:**

Web service is a way of communication or it is a technology it allows to develop interoperable distributed applications

**Interoperability** –Platform independent & Language independent

Java is platform independent but not language independent

Web services are not only strict on Java but also various languages like c, c++, .Net etc.



**Distributed Applications-** it is a server side application it exposes the objects over the network .in this ex- bank application is a server side application it exposes the payment gate way service object , those objects are accessing by some other applications on the network, it always exchanging the data or information over the network

**Web Application-** Server side application, web application is always communicate with the

Java🡪JSP, Servlets

PHP🡪PHP

.Net🡪ASP

**Web service:**

It is a distributed application

**Languages 🡪 API**

JAVA 🡪 Socket Programming API, CORBA, RMI, EJB API’s

.Net 🡪

C 🡪

C++ 🡪

**WSI: Web services Interoperability**

Open community .all the software companies are associated members so it’s common to all the languages and technologies

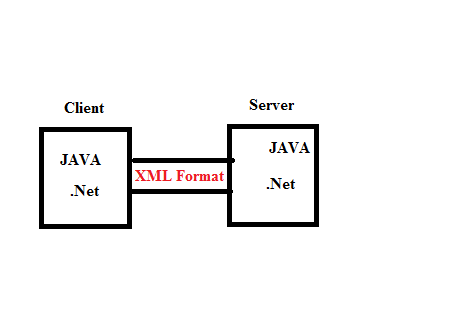
**XML**🡪

* **Extensible mark-up language**
* **Mark-up:** Enclosing the textual information between two tags i.e.; opening tag & closing tag.

**Eg:** <empName>raja</empName>

<empSalary>90000</empSalary>

* Here <empName>,<empSalary>🡪 Opening tag
* </empName>,</empSalary>🡪Closing tag
* HTML is also a Mark-up language
* **Why XML is Extensible:** XML tags are user defined tags. these tags can be extendable whenever required ,HTML tags are predefined
* **Language:** If you are design a web service application we need to design client & server application and the data should be communicate with client and server



* XML is given by W3C is a common language .it can be understand ,by any type of programming language
* Interoperable language
  + Plat form Independent
  + Language Independent
* XML version is 1.0
* XML Documents possible to maintain text based databases
* XML document we are using to transport the data from one application to another

|  |  |
| --- | --- |
| XML | HTML |
| Tags are user defined | Tags are predefined |
| Tags functionally defined by user and it is extensible | Limited |
| Case sensitive | Case insensitive |
| It describe the data | It displays the data |
| Every opening tag should maintain its closing tag | All the tags closing is optional |
| It is a mark-up language | It is also a mark-up language |

**Open Notepad🡪**

<employee>

<empNo>1001</empNo>

<empSalary>90000</empSalary>

</employee>

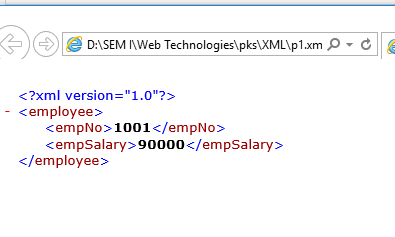
Save with .xml extensible like p1.xml

**How to view xml document**

🡪Browser Software

🡪Editor Software

**Output:**

****

**First line is prolog:** it represent version of xml, XML documents are well defined documents, because some strict rules are available in this XML elements &documents.

If we do not follow the rules will not get proper XML document

**XML documents contain:**

1. Elements
2. Attributes
3. Entity References

**Elements:**

* From opening tag to closing tag including text data called element

<employee>

<empNo>1001</empNo>

<empSalary>90000</empSalary>

</employee>

* From the above example there are 3 elements:
* employee
* empNo
* empSalary
* One element contain
  + Child elements
  + Attributes
  + Text data
  + Mix of all

**Where & When to use an Attributes /Child Attributes:**

* To define extra information of an element use child element or an attributes

Rules to write with Attribute:

Attribute values must be enclose with either single or double quotes

* + 1. Eg:<employee empNo=101>🡪invalid
    2. Eg:<employee empNo=”101”>🡪valid
    3. Eg:<employee empNo=’101’>🡪invalid
* Attributes names are unique for one element
* Duplicate elements are not allowed in an element
* Attribute order is not important in an element

**Entity References**

* **Syntax:** &Entity-reference name ;
* **Eg:**

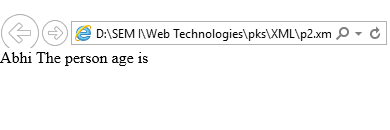
<Person >

<name>Abhi</name>

<age>The person age is <18</age >

</person>

* **Output:** is in valid

****

* Some special characters having some specialty ,so don’t give the in text , if we give it will invalid
* Following are the five predefined Entity references
  1. & lt; 🡪<( lesser than)
  2. &gt;🡪>(greater than)
  3. &amp;🡪&(ampersand)
  4. &apos;🡪’(single quotes)
  5. &quot;🡪” “(double quotes)
* **Eg:**

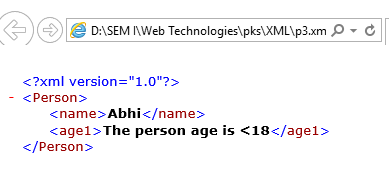
<Person >

<name>Abhi</name>

<age1>The person age is &lt;18</age1>

</Person>

**Output :**

****

**Well-formed XML Document**

**Rules:**

1. Must begin with PROLOG
2. Must be maintain a unique root element
3. Must be maintain closing tag for all the openings
4. Write all the XML elements in a proper cases ,because the xml elements are case sensitive
5. All the attributes & values we can enclose with either single quotes or double quotes
6. In the place of some special characters we can use an entity references

**DTD: Document Type Definition**

* DTD is a XML technique use to define the structure of a XML document
* DTD is a text based document with .dtd extension
* It contains
  + Element declarations
  + Attributes
  + Entity References
* Element Declaration Syntax:

<! Element element-name (content Model)>

* 1. **<!>**🡪Declaration
  2. **Element🡪**
     + - Text data
       - Child Element
       - Empty Element
       - Mix of all
  3. **Element-name 🡪**own choice
  4. **Content Model🡪** it specifies to allow the element
     + - Text data
       - Child Element
       - Empty Element
       - Mix of all

**Example of DTD:**

< ! ELEMENT employee (empno, name, salary)>

< ! ELEMENT empno (#PCDATA)>

< ! ELEMENT name (#PCDATA)>

< ! ELEMENT salary (#PCDATA)>

Here PCDATA means Parser Character Data

**XML:**

<employee>

<empno>101</empno>

<name>raja</name>

<salary>12000</salary>

</employee>

* Here, XML elements are user defined and structure defined with dtd.
* In the above model five types elements declarations are possible
  1. Text-only Element 🡪PCDATA
  2. Child only Element
  3. Empty Element
  4. Any Element
  5. Mixed Element
* Here,XML consider numbers, strings as text data

(3, 4, 5 will be explained in the cordiality concept)

**How to link/map dtd to xml:**

Use <! DOCTYPE > Declaration

**Types of DTD’s:**

1. Internal DTD’s🡪write the dtd within the xml document
2. External DTD’s🡪write the dtd in a suppurate text document & link with xml

**Internal DTD:** write the dtd within the xml document

**Syntax:** <! DOCTYPE root-element [DTD rules]>

XML Elements

**Example:**

<!DOCTYPE employee [

<!ELEMENT employee (empno, name, salary)>

<!ELEMENT empno (#PCDATA)>

<!ELEMENT name (#PCDATA)>

<!ELEMENT salary (#PCDATA)>

]>

<employee>

<empno>101</empno>

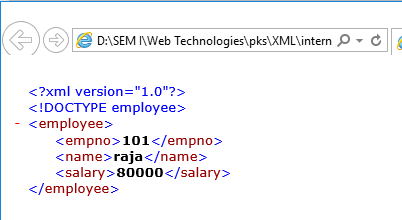
<name>raja</name>

<salary>80000</salary>

</employee>

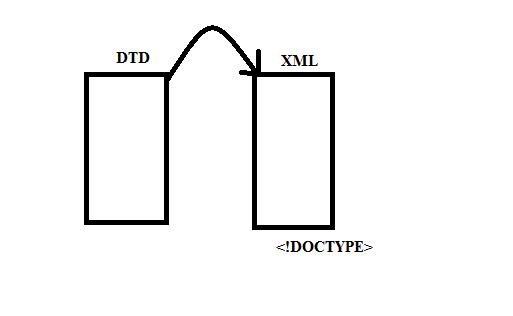
(save this file with .xml extension)

**Output:**

****

* Internal dtd’s are not reusable.

**External DTD:** write the dtd in a suppurate text document & link with xml



**We can create External DTD’s in two ways**

1. **Private DTD:**

Syntax :<! DOCTYPE root-element SYSTEM”DTD FILE NAME .dtd”>

**DTD file:**

<!ELEMENT employee (empno, name, salary)>

<!ELEMENT empno (#PCDATA)>

<!ELEMENT name (#PCDATA)>

<!ELEMENT salary (#PCDATA)>

(Save this file with .dtd extension)

**XML file**

<! DOCTYPE employee SYSTEM”dtdfilepath.dtd”>

<employee>

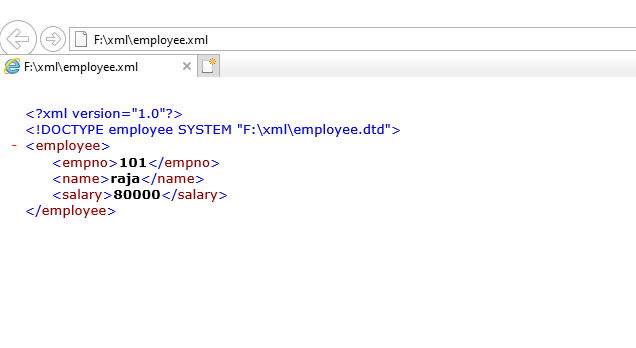
<empno>101</empno>

<name>raja</name>

<salary>80000</salary>

</employee>

**Output:**

****

**Differences between Internal, Private, and Public DTD’s:**

|  |  |  |
| --- | --- | --- |
| Internal DTD | Private DTD | Public DTD |
| If any dtd is specific to a particular xml document is called Internal DTD | If any dtd is specific to a particular **Project** is called Private DTD | If any dtd is **not specific** to a particular **Project** but also common to all is called Public DTD  Eg: Hibernet,Spring framework |

1. **Public DTD:**

**Syntax:**

<! DOCTYPE root-element PUBLIC “-//vendorname//version//EN””dtdfilename.dtd”>

Here,

- indicate not registered with ISO

+ indicate registered with ISO

Vendor name, version, EN is not mandatory

**DTD file:**

<!ELEMENT employee (empno, name, salary)>

<!ELEMENT empno (#PCDATA)>

<!ELEMENT name (#PCDATA)>

<!ELEMENT salary (#PCDATA)>

(Save this file with .dtd extension)

**XML file**

<!DOCTYPE employee PUBLIC "-//jntuhces//1.0//EN" "F:\xml\Public DTD\employee.dtd">

<employee>

<empno>101</empno>

<name>raja</name>

<salary>80000</salary>

</employee>

**Output:**

****

**How to define child elements?**

**DTD File:**

<!ELEMENT book (bookname , title)>

<!ELEMENT bookname (#PCDATA)>

<!ELEMENT title (#PCDATA)>

Here, in the above DTD book is the root element & book-name, titles are the child elements and, (coma) is the separator of the child elements in the first line)

**XML File:**

<!DOCTYPE book SYSTEM "D:\SEM I\Web Technologies\pks\XML\xml2\books.dtd">

<book>

<bookname>Core java</bookname>

<title> Core java</title>

</book>

**Ouput:**

**Here, child elements can also suppurate with vertical line (|),it indicates either-or**

**DTD File:**

<!ELEMENT book (bookname|title)>

<!ELEMENT bookname (#PCDATA)>

<!ELEMENT title (#PCDATA)>

**XML File:**

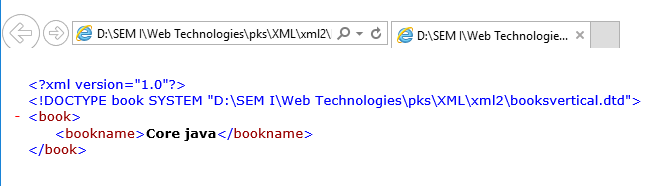
<!DOCTYPE book SYSTEM "D:\SEM I\Web Technologies\pks\XML\xml2\booksvertical.dtd">

<book>

<bookname>Core java</bookname>

</book>

**Output:**

****

**OR**

<book>

<title> Core java</title>

</book>

**Output:**

****

**Cardinality:** It specifies how many times an element can occur in an XML document

1. ⃰ (star)represents 🡪 0 ton (Zero to Many)
2. + (plus)represents 🡪 1 to n (One to Many)
3. ?(question)represents 🡪 0 or 1 (Zero to One)

<!ELEMENT books(book)> 🡪Only 1 book element

<!ELEMENT books(book\*)> 🡪0 or more book elements

<!ELEMENT>books(book-name|title,price,author+,publications?)>🡪either bookname or title,one price,1 to many authors,0 or 1 publications

**Types of ELEMENTS:**

Following are the possible elements in DTD

1. Text only Element
2. Child only Element
3. Mixed Element
4. EMPTY Element
5. ANY Element
6. **Text only Element :** 
   * It allows only text data
   * Use PCDATA data type in the content model
     + Eg: <!ELEMENT empname(#PCDATA)>
     + Eg: <!ELEMENT empname(#PCDATA)>
7. **Child only Element:**
   * It allows only child elements
     + Eg:<!ELEMENT employee(empname,empsalary)>
8. **Mixed Elements:**
   * It allows Text data & child elements
     + Eg:

<employee>

Employee name is <empname>raja</empname>and

Employee salary is<empsalary>50000</empsalary>

</employee>

1. **EMPTY:**
   * It won’t allows Text data ,child data but allows attributes
     + Eg:

<!DOCTYPE employees[

<!ELEMENT employees(employee\*)>

<!ELEMENT employees EMPTY>

]>

1. **ANY:**
   * It allows any data i.e.; text,child,mix,empty
     + Eg:

<!DOCTYPE employees[

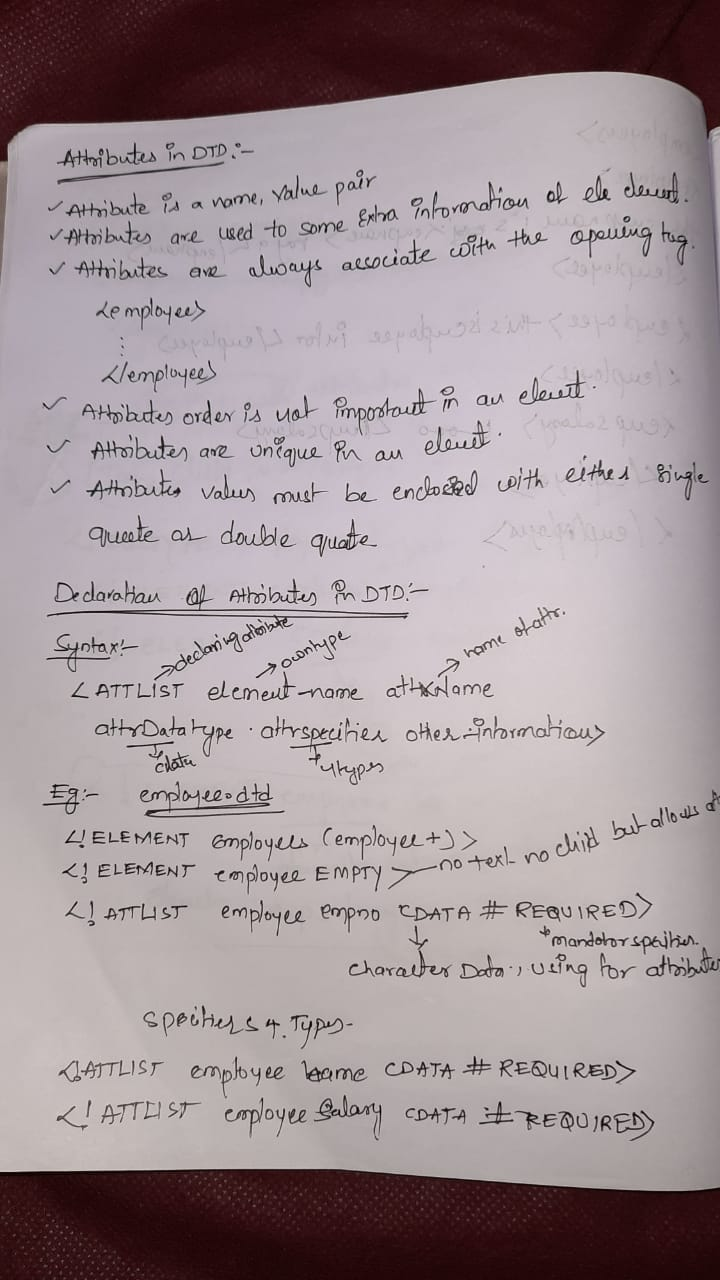
<!ELEMENT employees(employee+)>

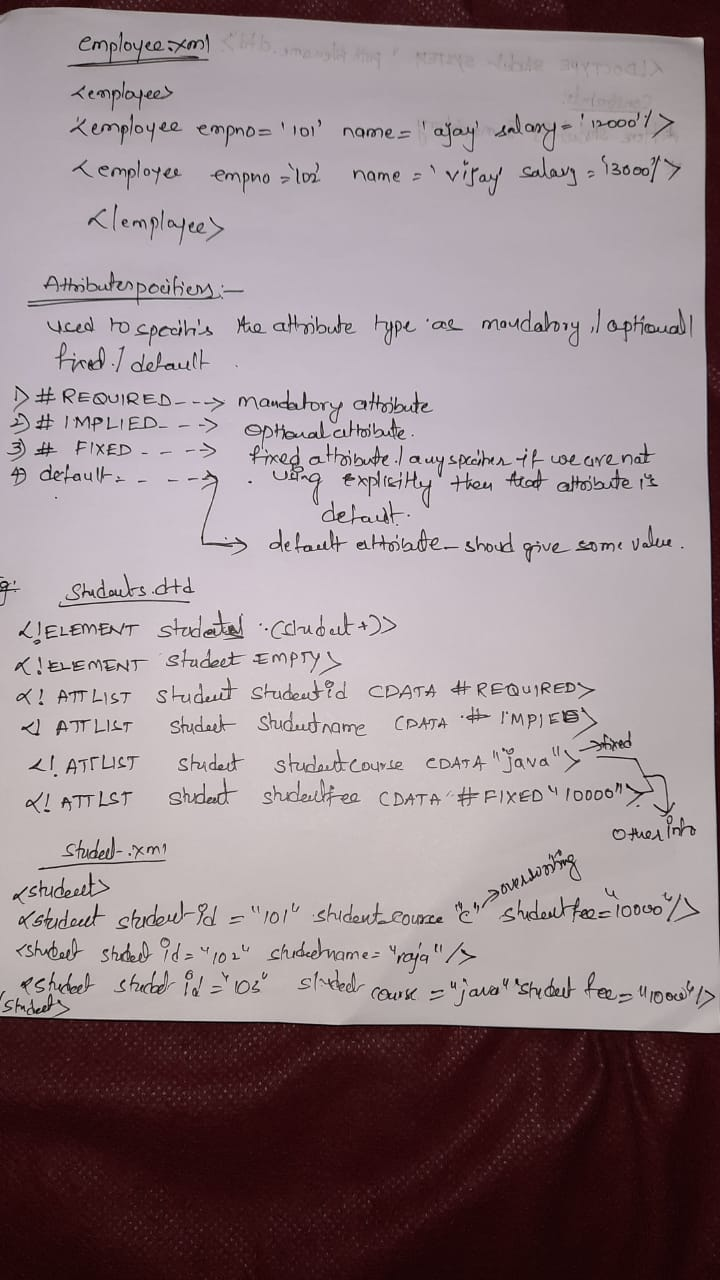
<!ELEMENT employees ANY>

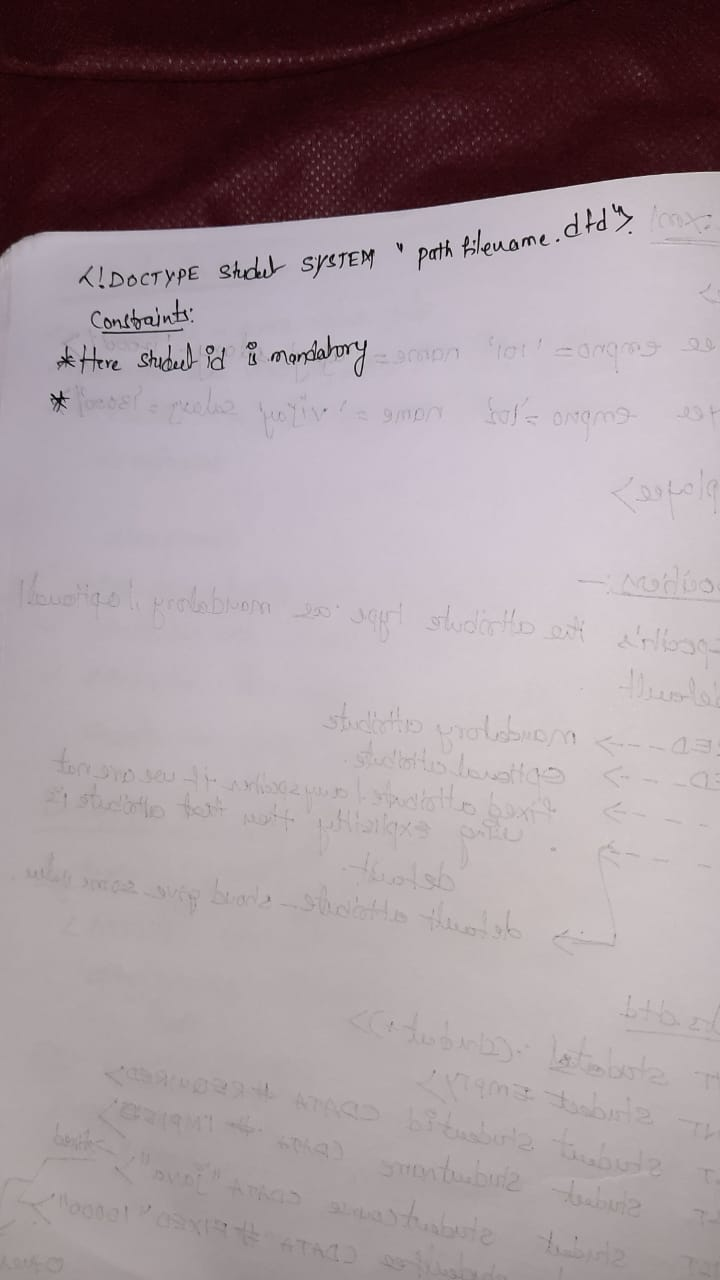
<!ELEMENT empname(#PCDATA)>

<!ELEMENT empsalary(#PCDATA)>

]>



****



**Attribute Data Types:**

What type of data does want to allow for an attribute?

* CDATA
* ENUMERATED
* ID
* IDREF
* IDREFS
* ENTITY
* NOTATIONS
* NMTOKEN
* NMTOKENS

**SYNTAX:**

<!ATTLIST element-name attr-Name attr-Datatype attr-Specifier Other-information>

**CDATA:** It allows Character Data

<!ELEMENT employees(employee)>

<!ELEMENT employee EMPTY>

<!ATTLIST employee empno CDATA #REQUIRED>

<!ATTLIST employee empname CDATA #REQUIRED>

<!ATTLIST employee salary CDATA #REQUIRED>

**ENUMERATED:** It allows any one of the value from the specified list

How to give (value1|value2|value3)

<!DOCTYPE payments [

<!ELEMENT payments (payment+)>

<!ELEMENT payment EMPTY>

<!ATTLIST payment payment-type (DD|CHECK|CASH) #REQUIRED>

]>

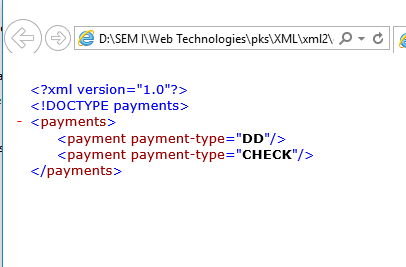
<payments>

<payment payment-type="DD"/>

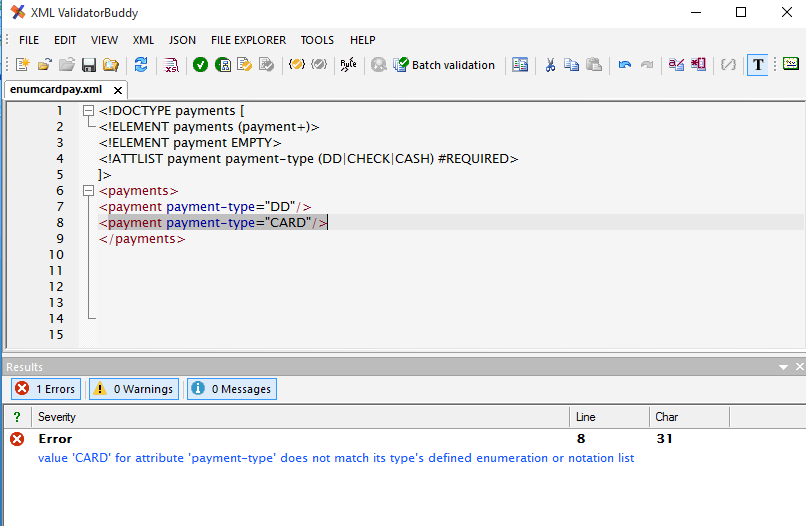
<payment payment-type="CHECK"/>

</payments>

**OUTPUT:**

****

**If you give payment type=”CARD “which is not available on the DTD specified list , then you get an error report (check below screenshot)**

****

**ID:** It allows unique values and should not start with digit but we can start with \_ (underscore)any alphabets(A-Z/ a-z)

<!DOCTYPE students[

<!ELEMENT students (student+)>

<!ELEMENT student EMPTY>

<!ATTLIST student sid ID #REQUIRED>

<!ATTLIST student sname CDATA #REQUIRED>

]>

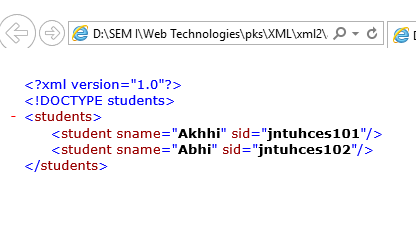
<students>

<student sid=" jntuhces101" sname="Akhhi"/>

<student sid="jntuhces102" sname="Abhi"/>

</students>

**OUTPUT:**

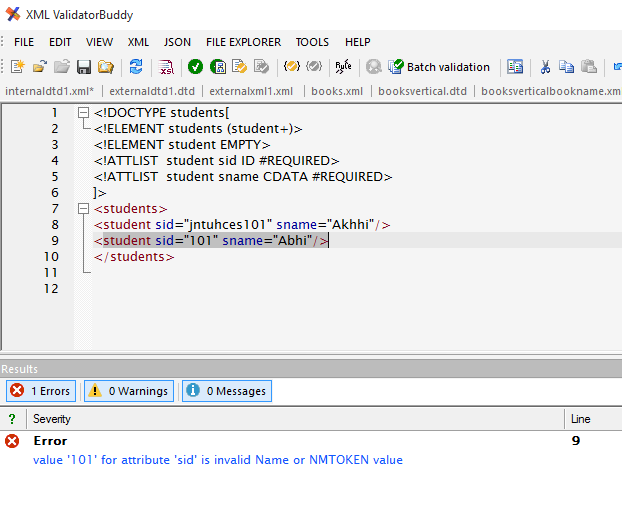
****

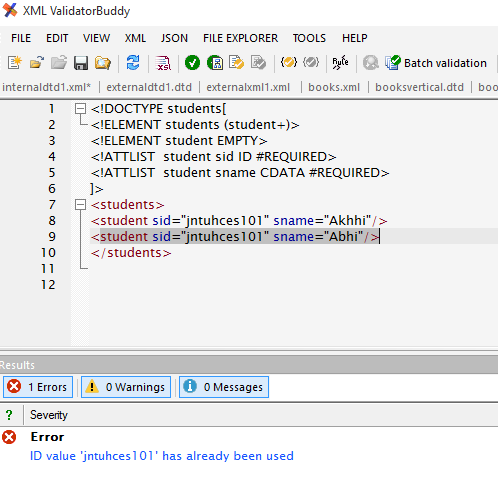
Here,

It allows unique values and should not start with digit but we can start with \_(underscore)any alphabets(A-Z/ a-z),-(hifan),0-9

But dnt start with -

It does not allows $ symbol





**IDREF:** It stores the value of some another ID element attribute

<!DOCTYPE students [

<!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 #REQUIRED>

]>

<students>

<course cid="CJ201" cname="corejava" cfee="500"/>

<course cid="AJ202" cname="adjava" cfee="1500"/>

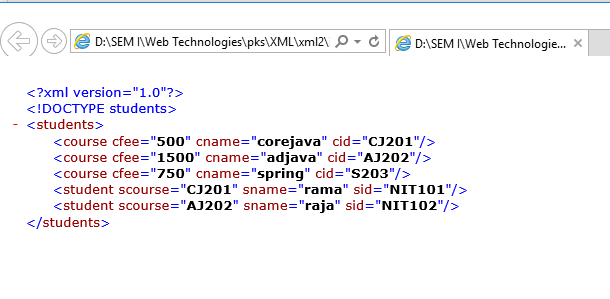
<course cid="S203" cname="spring" cfee="750"/>

<student sid="NIT101" sname="rama" scourse="CJ201"/>

<student sid="NIT102" sname="raja" scourse="AJ202"/>

</students>

**OUTPUT:** in the above ex cid="CJ201” value is referencing to scourse="CJ201"

****

**IDREFS:** It stores the value of some another ID element attribute, referring list of values

<!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" cfee="500"/>

<course cid="AJ202" cname="adjava" cfee="1500"/>

<course cid="S203" cname="spring" cfee="750"/>

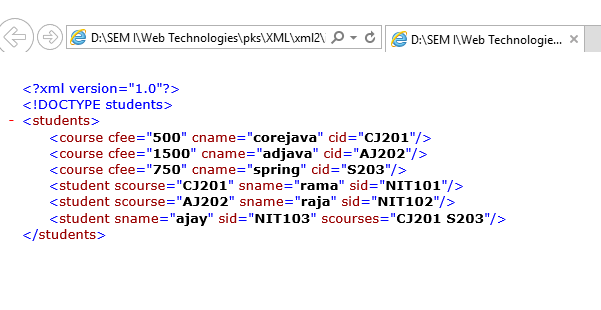
<student sid="NIT101" sname="rama" scourse="CJ201"/>

<student sid="NIT102" sname="raja" scourse="AJ202"/>

<student sid="NIT103" sname="ajay" scourses="CJ201 S203"/>

</students>

**OUTPUT:**

****

|  |  |  |
| --- | --- | --- |
| ID | IDREF | IDREFS |
| Attribute type value must be unique in the XML | Attribute stores some another element ID attribute type value | Similar to IDREF |
| Stores one value | Stores one value | Stores list of values |

**NMTOKEN:** one of the type DTD, using to stores the attributes

If any attribute datatype is NMTOKEN the attribute always valid XMl , value must be valid XML means value contain alphabetical letters(A-Z),digits(0-9) special charcters - \_  **.**

Will not allow the following special characters +? @#$

**(NOTE-CDATA:** It is similar toNMTOKEN but it contains more restrictions to accept the values, it can allows all NMTOKEN acceptable values and also +? @#$ )

**EX:**

<!DOCTYPE courses[

<!ELEMENT courses (course)+>

<!ELEMENT course EMPTY>

<!ATTLIST course cid ID #REQUIRED>

<!ATTLIST course cname NMTOKEN #REQUIRED>

]>

<courses>

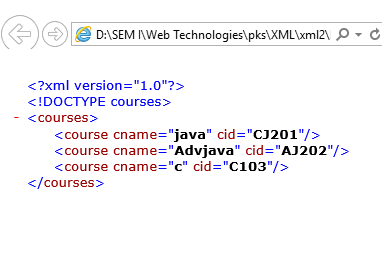
<course cid="CJ201" cname="java"/>

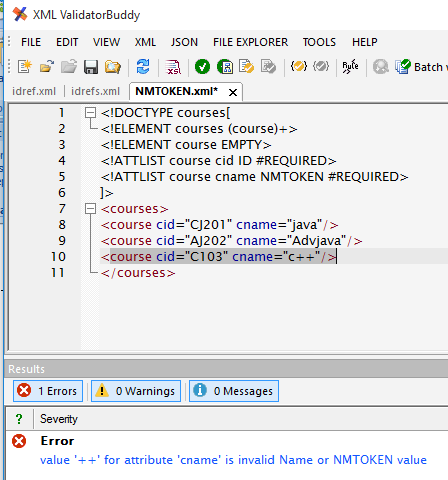
<course cid="AJ202" cname="Advjava"/>

<course cid="C103" cname="c"/>

</courses>

**OUTPUT:**

****

****

|  |  |
| --- | --- |
| NMTOKEN | NMTOKENS |
| Always allows single value | Allows list of values |

**NOTATION:** Is acting as shortcut to some specific types of media types

**NOTE:** Corresponding element content model should not be EMPTY

**Ex:** Represent photo type

<photo phototype=”image/jpeg”/>

<photo phototype=”image/gif”/>

image/jpeg🡪 jpg.

image/gif🡪gif

application/json🡪json

**SYNTAX:**

<!NOTATION NOTATION\_NAME SYSTEM “IMAGE/JPEG”>

**EX:**

<!DOCTYPE photos[

<!NOTATION JPG SYSTEM "image/jpeg">

<!NOTATION GIF SYSTEM "image/gif">

<!ELEMENT photos (photo+)>

<!ELEMENT photo (#PCDATA)>

<!ATTLIST photo phototype NOTATION (JPG|GIF) #REQUIRED>

]>

<photos>

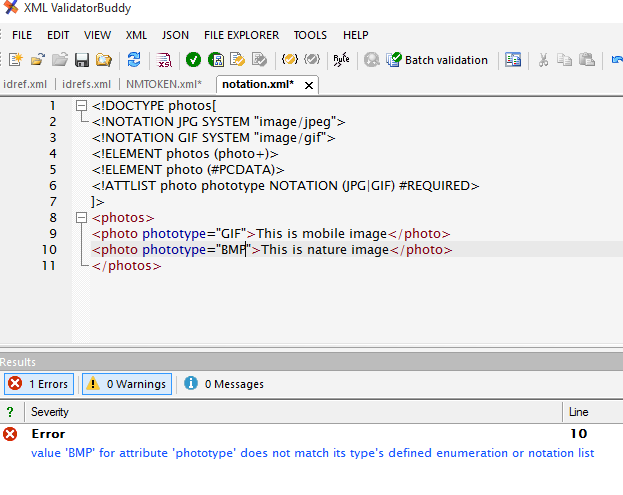
<photo phototype="GIF">This is mobile image</photo>

<photo phototype="JPG">This is nature image</photo>

</photos>

**OUTPUT:**

****

****

**ENTITY:** Is acting for a long length repeated string

Possible to use as Parsed (PCDATA), and unparsed (CDATA) data

Element content 🡪PCDATA

Attribute content🡪CDATA

**ENTITY Creation for Parsed Content:**

<!ENTITY entity-name “requiredvalues”>

<!ENTITY JNTUHCES “JNTUH COLLEGE OF ENGINEERING SULTANPUR”>

In the xml-------1000

&EntityReferenceName;

&JNTUHCES;

**EX:**

<!DOCTYPE courses[

<!ENTITY JNTUHCES "JNTUH College of Engineering Sultanpur">

<!ELEMENT courses (course\*,inst\*)>

<!ELEMENT course (#PCDATA)>

<!ELEMENT inst (#PCDATA)>

]>

<courses>

<course>B.Tech CEE</course>

<course>B.Tech CSE</course>

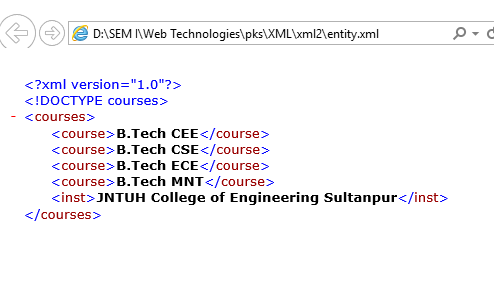
<course>B.Tech ECE</course>

<course>B.Tech MNT</course>

<inst>&JNTUHCES;</inst>

</courses>

**OUTPUT**

****

**XSD:**

XML Schema Definition, both XSD, DTD is used to define the structure of an XML document

|  |  |
| --- | --- |
| DTD | XSD |
| * It is an XML technique to define the structure of an XML document | * It is an XML technique to define the structure of an XML document |
| * Old technique | * New technique |
| * Very less no of data types to restrict the data | * Many more data types to restrict the data |
| * Not allows to create our own data types | * It allows to create our own data types |
| * Three cardinality operators available here | * Many no of cardinality constants are available here |
| * It is possible to allow the elements following times 0-N,1-n,0-1 | * Here it can also supports 0-5,5-10,10-15 |
| * It doesn’t having any namespace concepts | * But it has namespace concepts |
| * Only PCDATA is using for element declaration |  |

**Name space discussion**

Package com.jntuhces.service;

Public class AccountService {

:

:

:

}

Class Test {

P S V main(string[] args){

Com.jntuhces.serive.AccountService a;

If there is fully classified services there are no ambiguity problems

**Ex:**

Class Test{

P s v main (String[] args){

Java.util.Date.d1;

Java.sql.Date.d2;

}

}

If no packages in java will ambiguities, similarly namespace also created in XML

Like c🡪library functions

Java🡪packages

.Net🡪namespaces

XSD🡪namespaces

**Data Types:**

1. **Simple Types** 
   1. **Built in Types (44**)
      1. **Primitive Types(19)**

String, Boolean, decimal, float, double, duration, datetime, time, date,

gYearMonth, gYear,gMonthDay,gDay,gMonth,hexbinary,base64Binary,anyURI, QName, NOTATION

* + 1. **Derived Types(25)**

NormalizedStrings,token,language,NMTOKEN,NMTOKENS,Name,NCName,ID,IDREF,TDREFS,ENTITY,ENTITIES,integer,nonpositiveInteger,negitiveInteger,long,Int,short,byte,nonNegitiveInteger,unsignedLong,unsignedInt,UnsignedShort,unsignedByte,positiveInteger.

* 1. **User-derived Types**
     1. Atomic Types
     2. Non-Atomic Types

1. **Complex Types** 
   1. EMPTY
   2. Simple Content
   3. Complex Content
      1. Sequence
      2. Choice
      3. All

**XSD Elements:**

**Simple Elements:** If any elements contains only the text data is called Simple Element , doesn’t contain child elements, attributes

<empNO>101</empNo>

<name>raja</name>

**Syntax:**

<element name=”element-name” type=”data type”>

</element>

**Ex:**

<element name=”empNo” type=”int”/>

<element name=”name” type =”string”/>

**Complex Elements:** It allows Text data and it contains child elements or attributes or both

<employee>

<empNO>101</empNo>

<name>raja</raja>

</employee>

Here, <employee>is a Complex element,<empNo>,<name> are the child elements

* + - 1. Elements with Text-Data and Attributes
      2. Elements with Empty Content and attributes
      3. Elements with Child Elements And/OR attributes
      4. Elements with Mixed Content And/OR attribute

**1st Way:** A Complex Element with Child Element

<schema>

<element name=”element-name”>

<complexType>

<sequence>

<element name=”child1” type=”dt”/>

<element name=”child1” type=”dt”/>

----

-----

</sequence>

</complexType>

</element>

</schema>

In this Complex Elements use any one of the data type of complex content data types.

**EX:**

**XSD:**

<schema>

<element name=”employee”>

<complexType>

<sequence>

<element name=”empNo” type=”int”/>

<element name=”name” type=”string”/>

<element name=”salary” type=”decimal”/>

</sequence>

</complexType>

</element>

</schema>

**XML:**

<employee>

<empNo>101</empNo>

<name>raja</name>

<salary>9000</salary>

</employee>

**XSD Attributes Declaration**

**Syntax:**

<attribute name=”attribute-name” type=”attribute-type”/>

**Ex:**

<attribute name=”courseId” type=”int”/>

<attribute name =”courseName” type=”String”/>

<attribute name =”courseFee” type=”decimal”/>

**Attribute Declaration:**

**Mandatory Attribute:**

<attribute name=”courseId” type=”int” use = “required”/>

**Optional Attribute:**

<attribute name=”courseId” type=”int” />

**Default Attribute:**

<attribute name=”courseName” type=”string” default=”java”/>

**Fixed Attribute:**

<attribute name=”courseFee” type=”decimal” fixed=”1000”/>

**In XSD complex Type with Attributes:**

<schema>

<element name=”course”>

<complexType>

<attribute name=”courseId” type=”int”/>

<attribute name =”courseName” type=”String”/>

<attribute name =”courseFee” type=”decimal”/>

</complexType>

</element>

</schema>

**In XML:**

<course courseId=”101” courseName=”java” courseFee=”2000”/>

**Complex Type Elements**

1. Elements with Child Elements And/or Attributes
2. Elements with Mixed Content And/or attributes
3. Elements with Text-Data and attributes
4. Elements with Empty Content and attributes

**Elements with Child Elements**

<schema>

<element name=”employee”>

<complexType>

<sequence>

<element name=”empNo” type=”int”/>

<element name=”name” type=”string”/>

<element name=”salary” type=”decimal”/>

</sequence>

</complexType>

</element>

</schema>

**Elements with Mixed Content**

<schema>

<element name=”employee”>

<complexType mixed=”true”>

<sequence>

<element name=”empNo” type=”int”/>

<element name=”name” type=”string”/>

<element name=”salary” type=”decimal”/>

</sequence>

</complexType>

</element>

</schema>

**XML:**

<employee>

The EmpNo is <empNo>101</empNo>

And Name is <name>Sagar</name>

And salary is <salary>2000</salary>

</employee>

**Elements with Text-Data and Attributes**

<schema>

<element name=”employee”>

<complexType >

<simpleContent>

<extension base =”string”>

<attribute name=”empNo” type=”int”/>

<attribute name=”name” type=”string”/>

</extension>

</simpleContent>

</complexType>

</element>

</schema>

**XML:**

<employee – (attributes here)------>

This is jntuhces emplpoyee

<employee>

**Note: <restriction>, <extension > both are using for data types.**

**Elements with Empty Content and Attributes**

<schema>

<element name=”course”>

<complexType >

<element name=”courseId” type=”int”/>

<element name=”courseName” type=”string”/>

<element name=”courseFee” type=”decimal”/>

</complexType>

</element>

</schema>

**XML:**

<course courseId=””>

courseName =””

courseFee=””/>

**XSD Namespace Concept:**

In java package concept🡪is used to group all the classes as a single unit.

Whenever a class is present in a package at the time of using of that class we will usefully qualified names.

Fully qualified name means 🡪class Name along with its package Name

If we are using fully qualified names it is possible to avoid some kind of ambiguity problems

Class Test{

Java.util.Date.d1;

Java.sql.Date.d2;

}

}

In XSD Namespace Concept🡪 namespaces are used to group all the elements as a single unit.

At the time of using the element we will use fully qualified name

Here fully qualified name means 🡪 NameSpace+ElementName

Java-🡪Package packageName;

For one .java file only one package declaration statement possible to use

XSD🡪targetNamespace