**XML**

**XML**  stands for extensible markup Language . Markup Language is the language using which you can build other Language Like .HTML , XML.

XML defined and governed by w3org. The first and final virgin of XML is XML 1.0. XML is the document which represent data . Unlike C and C++ , JAVA etc. XML is not programing Language . It is the defacto standard for exchanging information between computer systems. When we represent data in xml document it would be more structured and has well defined semantics attached to it “semantics” here represents what a particular data field is representing or stands for , so for that any person reading the xml document would be able to interpret in the same manner

Another significant feature of xml is , once written it is portable across the platform (windows Linux etc.) . No changes are required to carry the data across platform.

Every Language has keywords. If you take Example as C , it has keywords like (if , for ,while, do , break, continue) . but when It comes to xml there are no keywords or reserved words. What you write will become the element of that xml document.

**4.1 XML Element**

Element in an xml is written in angular braces for e.g. <beans>. In xml there are two types of elements as follows

1. Start Element/Opening tag:-Start element is the element which is written in <elementname> indicating the start of block.
2. 2) End Element/End tag:- End Element is the Element which is written in </elementname> indicating the end of a block.

As everything is written in terms of start and end elements, xml is said to be more structure in nature. An xml Element may contain content or may contain other element under it. So xml elements which contain other element in it are called as compound element or xml containers.

**4.2 XML Attribute**

If we want to have supplementary information attached to an element instead of having it as content or another element , we can write it as an Attribute of the element

Example:-

<book type=”entertainment”>

<isbn>isbn1001</isbn>

</book>

In the above example “book” is an element which contains once attributes type which act as an supplementary information. Isbn is the sub- element of the book element.

**Well-formness:**

**A**s how any programing language has syntax is writing its code well formness of xml document talks about how to write an xml document. Well-formness indicates indicates the readability nature of an xml document . In other way if an xml document is said to be well-formed than it is readable in nature.

Following are the rules that describe the well-formness of an xml document.

1. Every XML document must start PROLOG :- prolog stands for processing instruction , and typically used for understanding about the version of XML used and data encoding used .

Example :- <?xml version=”1.0” encoding=”uft-8”?>

1. Root Element :- XML document must contain a root element and should be the only

One root element. All the other element should be the children of root element.

1. Level constraint:- Every start element must have an end element and the level at

which you open a start element . the same leve you need to close your end element as well.

Example :-

<<?xml version=”1.0” encoding=”uft-8”?>

<student>

<info>

<rollno>42</rollno>

<name>john</info> (-- info is closed in-correctly--)

</name>

</student>

If any xml is said to follow the above defined the above defined rules then it is termed as well as formed.

**4.4 XML Usage :-**

An xml document is used in two scenarios

1. **Used for transferring information :-** As said earlier XML is a document represented data and is used for carrying information between two computer system in an Interoperable manner.
2. **Configurations:-** IN J2EE world every component/ resource you develop like servlet or EJB it has to be deployed into a web server. For e.g. in a servlet application, the path with which the servlet has to be eccessible should be specified to the servlet container so that it can map the incoming request to the servlet . This is done in web .xml . When we provide the configuration information in xml file the main advantage is the same xml document can be used in the different platforms without any changes.

**Validatity**

Every xml in-order to parse (read)should be well-formedin nature. As said earlier well-formness of an xml document indicates whether it is readable or not, it doesn’t talks about whether the data contained in it is valid or not.

Validatity of the xml document would be defined by application which is going to process your xml document . let’s consider a scenario as follows.

You want to get driving license. In order to get a driving license you need to follow certain process like filling the RTA forms and signing them and submitting to the RTA department.

Instead of this can you write your own format of letter requesting the driving license from an RTA department , which seems to be not relevant because driving license is something that would be issued by the RTA department. So . the whole and sole authority of defining what would a person has to provide data to get a driving license will lies in the hands of RTA department rather than you.

In the same way when an application is going to process your xml document , the authority of defining what should be there as part of that xml is lies in the hands of the application which is going to process your document.

For Example let’s consider the bellow xml fragment

Example:-Purchase order xml document

<?xml version =”1.0” encoding=”uff-8”?>

<purchaseorder>

<Orderitems>

<item>

<itemcode>IC324</itemcode>

<quantity>24</quantity>

</item>

<item>

<itemcode>IC324</itemcode>

<quantity>abc</quantity>

</item **>**

**</**orderitem>

<purchaseorder>

In the above xml even though it confirm to all the well-formness rules it can not be use for business transaction, As a second<quality> element carry the data as “abc” which doesn’t make any sense.

So in order to check the validity we need to define the validation criteria of an xml document in either DTD and XSD document.

**5)DTD:-**

DTD stands for the document type definition. It is the document which define the structure of an xml document.

In xml document we have to types of element, element which carries the data are called simple element. An element which contain sub element under it is called compound element.

In a language before using a variable we need to declare it. Similarly before using an element in xml first we need to declare in DTD while declaring the element In DTD we need to tell weather it is simple or compound element.

So we need to understand a syntax of how to declare single and compound element in DTD

5.1) **syntax for declaring single element of an XML**

**<**!Element elementname (CONTENT- TYPE OR CONTENT MODEL)>

**For** Exp…

**<!ELEMENT**  itemCode (#PCDATA)>

**As** per the above declaration , it means in my xml we should have an element whose name is itemCode and it contains content of type #PCDATA(parsable character DATA).

In an xml characters likes “<”,”>”, “’,”’ and “&” are valid characters and if the text content contains those characters the parsers are going to parse or will expand them as further entities.

**5.2) Syntax for compound element of an XML**

<!ELEMENT elementname(sub-elem1, sub-elem2…….)

For e.g…

<!ELEMENT itemcode(#PCDATA)>

<!ELEMENT itemcode(#PCDATA)>

<!ELEMENT item(itemcode, quantity)>

As per the above declaration , we declare the element itemcode in XML in which contain child element like itemcode quantity.

**In XML**

**<**item**>**

**<**itemcode**>**Ic200</itemcode>

**<**quantity>35</quantity**>**

</item>

We need to follow certain rules while declaring element in DTD.

1. Element name should follow the java variable naming convension.
2. Content of an XML is always case-sensitive . so if we declare element name in the small case in DTD , it should appear in small case in XML as well.
3. The child element separate in the above example is used as “,” comma. this also called the sequence separator. This means in the item element the first child also should be itemCode followed by quantity.

Let’s take a complete Xml and derive how does the DTD looks like for it:-

Here are few of guideline in creating a DTD from XML

1. Identify all the single element of XML from bottom of XML and draft them in DTD.
2. Identify the compound element of the XML from lower level in XML and draft in the DTD.

XML Document

<?xml version =”1.0” encoding=”utf-8”?>

<purchase order>

<orderitem>

<item>

<itemcode>IC323</itemcode>

<quantity>24</quantity>

</item>

<item>

<itemcode>IC324</itemcode>

<quantity>abc</quantity>

</item>

</orderitem>

</purchaseorder>

For the above xml the DTD looks like shown below:-

<?xml version=”1.0”encoding=”utf-8”>

<!Element purchaseorder(orderItem)>

<!Element orderItem (item+)>

<!Element item(itemcode,quantity)>

<!Element itemcode(#PCDATA)>

<!Element quantity(PCDATA) >

**5.3 Occurrence of an element under another element**

In the above xml if you observe the order item can contain any number of item element in it , but atleast one item element must be there for a purchase order.

This is called the occurrence of an elementunder another element . to indicate this we use three symbols.

? – it represent the sub element under the parent element can appear 0 or 1 time

+ - indicates the sub element must appear atleast once and can repeat any number of times (0-n).

\*- indicates the sub element is option and can repeat any number of times .

You will mark the occurrence of an element under another element is follow.

<!Element elementname (sub-ele1(?/+/\*), sub-elem2(?/+/\*)>

Leavimg any element without any symbol indicate it is a mandatory

And max can repeat only once .

**5.4 Element with any contents**

Element declare with content type as any , can contain any combination of the parsable data.

<ELEMENT elementname ANY >

<ELEMENT mailbody ANY>

IN an e-mail body part we have content which can be mixture of the any parsable character which can be as ANY type.

**5.5 Element with either /or content**

Let’s consider an example where in an e- mail the following element will be there to , from subject and mailbox . In these element to and form or mandatory element and either subject or mailbox body should be present but not both. To declare the same we need to use or content separator instead of sequence separator.

<!Element mail (to,, form (subject | mailbody))>

For the above declaration the xml looks as bellow.

|  |  |
| --- | --- |
| <mail>  <to> toaddr.com</to>  <from>fromaddr.com<from>  <subject>myweb</subject>  <mail> | <mail>  <to> toaddr.com</to>  <from>fromaddr.com<from>  <mailbody >myweb</mailbody>  <mail> |

**5.6 declaration mixed content**

We can even declare an element with mixture of Passable data and elements called mixed content as follows.

**5.7 Declaration attribute for an element**

Syntax:-

<!ATTLIST elementname attributename attribute type attribute value>

As shown above to declare an attribute you need to use the tag ATTLIST and element name stands for which element you want to declare the attribute and attribute name stands for what is the attribute you want yo have in that element

The attribute type can be following:

|  |  |
| --- | --- |
| Type | Declaration |
| CDATA | The value is character data |
| (en1|en2….) | The value must be once form the enumerated list of values. |
| ID | The value is unique id. |
| IDREF | The value is the id of another element |
| NMTOKEN | The value is a valid xml element name |
|  |  |

The attribute value can be following:

|  |  |
| --- | --- |
| Value | Description |
| #Required | The attribute is required |
| #Implied | The attribute is not required |
| #Fixed Value | The attribute value is fixed. |

**5.7.1 Default attribute value**

<!ELEMENT ShippingAddress Type CDATA “permanent”>

In xml:-<shipping address type=”permanent” >…..</shipping address>

**5.7.2 # Required**

**<**!ATTLIST Shipping Address type CDATA #Required> this indicates type attribute is mandatoryin shipping address element.

**5.7.3 #Implied**

**<**!ATTLIST Shipping Address type CDATA #Implied> this indicates type attribute in shipping address element is optional.

**5.7.4 #Fixed**

**<**!ATTLIST Shipping Address type CDATA #Fixed “permanent”> this indicates type attribute is mandatory in shipping address element must contain only the value as permanent.

**5.7.5 #Enumerated Attribute Values**

**<**!ATTLIST Shipping Address type (permanent|temporary) “permanent”> this indicates type attribute in shipping address element should contain only two possible values either permanent or temporary.

**5.8 Drawback With DTD’s.**

**DTD** is not type safe which means when we declare simple elements we indicate it should contain data of type (#PCDATA). #PCDATA means Parsable character data means any data that is computer represent format . So it’s indicates an element can contain any type of data irrespective of whether it is int or float or string. You can not impose stating my element should contain int type data or float. This is limitation with DTD documents.

**6 XML Schema Document (XSD)**

**XSD** stands for xml schema document. XSD is also an XML Document . It is owned by w3org. The latest virgin of XSD is 1.1.

Even through we can

# XSD - Miscellaneous Data Types

XSD has a few other important data types, such as **Boolean, binary,** and **anyURI.**

## <xs:boolean> data type

The <xs:boolean> data type is used to represent true, false, 1 (for true) or 0 (for false) value.

### <xs:boolean> Example

Element declaration in XSD −

<xs:element name = "pass" type = "xs:boolean"/>

Element usage in XML −

<pass>false</pass>

## Binary data types

The Binary data types are used to represent binary values. Two binary types are common in use.

* **base64Binary** − represents base64 encoded binary data
* **hexBinary**− represents hexadecimal encoded binary data

### <xs:hexbinary> Example

Element declaration in XSD −

<xs:element name = "blob" type = "xs:hexBinary"/>

Element usage in XML −

<blob>9FEEF</blob>

## <xs:anyURI> data type

The <xs:anyURI> data type is used to represent URI.

### <xs:anyURI> Example

Element declaration in XSD −

<xs:attribute name = "resource" type = "xs:anyURI"/>

Element usage in XML −

<image resource = "http://www.tutorialspoint.com/images/smiley.jpg" />

## Numeric Data Types

Following is the list of commonly used numeric data types.

|  |  |
| --- | --- |
| **S.No.** | **Name & Description** |
| 1. | **byte**  A signed 8 bit integer |
| 2. | **decimal**  A decimal value |
| 3. | **int**  A signed 32 bit integer |
| 4. | **integer**  An integer value |
| 5. | **long**  A signed 64 bit integer |
| 6. | **negativeInteger**  An integer having only negative values (..,-2,-1) |
| 7. | **nonNegativeInteger**  An integer having only non-negative values (0,1,2,..) |
| 8. | **nonPositiveInteger**  An integer having only non-positive values (..,-2,-1,0) |
| 9. | **positiveInteger**  An integer having only positive values (1,2,..) |
| 10. | **short**  A signed 16 bit integer |
| 11. | **unsignedLong**  An unsigned 64 bit integer |
| 12. | **unsignedInt**  An unsigned 32 bit integer |
| 13. | **unsignedShort**  An unsigned 16 bit integer |
| 14. | **unsignedByte**  An unsigned 8 bit integer |

## Restrictions

Following types of restrictions can be used with Miscellaneous data types except on boolean data type −

* enumeration
* length
* maxLength
* minLength
* pattern
* whitespace

SAX Example

* In the above class we are overriding five methods which would be triggered based on the type of elements the parser is pointing to on the source XML document.

(a) startDocument- Would be triggered at the start of the XML document

(b)startElement- Whenever the parser encounters the starting element, it raises this method call by passing the entire element information to the method.

(c) characters – This method would be invoked by the parser, whenever it encounters data portion b/w the XML elements. To this method it passes the entire XML as character array from which the current data portion begins and second integer representing the number of characters the data span to.

(d) endElement – Would be triggered by the parser, when it encounters a closing element or ending element.

(e) endDocument – Would be triggered by the parser once it reaches end of the document.

(f) Once the handler has been created , We need to call the parser class parse() method by passing the source as XML and the handler as the object of handler class which we created above. So , that the parser will reads the XML elements and triggers a respective method call on the handler object provided.