UNIT-II
XML

WHAT IS XML?

XML stands for extensible markup language. XML is a markup language much like HTML. XML was designed to carry data, not to display data. XML tags are not predefined. You must define your own tags. XML is designed to be self-descriptive. XML is a w3c recommendation.

THE DIFFERENCE BETWEEN XML AND HTML:

XML is not a replacement for HTML. XML and HTML were designed with different goals. XML was designed to transport and store data, with focus on what data is. HTML was designed to display data, with focus on how data looks. HTML is about displaying information, while XML is about carrying information.

XML does not do anything: May be it is a little hard to understand, but XML does not do anything. XML was created to structure, store, and transport information. The following example is a note to chythu, from ravi, stored as XML:

```
<note>
<to> chythu </to>
<from> ravi </from>
<heading>reminder</heading>
<body>don't forget me this weekend!</body>
</note>
```

The note above is quite self-descriptive. It has sender and receiver information, it also has a heading and a message body. But still, this XML document does not do anything. It is just information wrapped in tags. Someone must write a piece of software to send, receive or display it.

With XML you invent your own tags: The tags in the example above (like <to> and <from>) are not defined in any XML standard. These tags are "invented" by the author of the XML document. That is because the XML language has no predefined tags. The tags used in HTML are predefined. HTML documents can only use tags defined in the HTML standard (like <p>, <h1>, etc.). XML allows the author to define his/her own tags and his/her own document structure. XML is not a replacement for HTML; XML is a complement to HTML. It is important to understand that XML is not a replacement for HTML. In most web applications, XML is used to transport data, while HTML is used to format and display the data.

My best description of XML is this: XML is a software- and hardware-independent tool for carrying information. XML is a W3C recommendation. XML became a W3C recommendation February 10, 1998. XML is everywhere. XML is now as important for the web as HTML was to the foundation of the web. XML is the most common tool for data transmissions between all sorts of applications.

XML separates data from HTML: If you need to display dynamic data in your HTML document, it will take a lot of work to edit the HTML each time the data changes. With XML, data can be stored in separate XML files. This way you can concentrate on using HTML for

layout and display, and be sure that changes in the underlying data will not require any changes to the HTML. With a few lines of JavaScript code, you can read an external XML file and update the data content of your web page.

Unique Features of XML:

a) **Sharing of data:** In the real world, computer systems and databases contain data in incompatible formats. XML data is stored in plain text format. This provides a software- and hardware-independent way of storing data. This makes it much easier to create data that can be shared by different applications.
b) **Transporting data:** One of the most time-consuming challenges for developers is to exchange data between incompatible systems over the internet. Exchanging data as XML greatly reduces this complexity, since the data can be read by different incompatible applications.
c) **Change of platform:** Upgrading to new systems (hardware or software platforms), is always time consuming. Large amounts of data must be converted and incompatible data is often lost. XML data is stored in text format. This makes it easier to expand or upgrade to new operating systems, new applications, or new browsers, without losing data.
d) **Availability of data:** Different applications can access your data, not only in HTML pages, but also from XML data sources. With XML, your data can be available to all kinds of "reading machines" (handheld computers, voice machines, news feeds, etc), and make it more available for blind people, or people with other disabilities.

Structure of XML:

XML documents form a tree structure that starts at "the root" and branches to "the leaves". An example of XML document is shown below. XML documents use a self-describing and simple syntax:

```
<?XML version="1.0" encoding="iso-8859-1"?>
<note>
  <to> Chythu </to>
  <from> Ravi </from>
<heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

The first line is the XML declaration. It defines the XML version (1.0) and the encoding used (ISO-8859-1 = latin-1/west European character set). The next line describes the root element of the document (like saying: "This document is a note"):
<note>
The next 4 lines describe 4 child elements of the root (to, from, heading, and body):
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
And finally the last line defines the end of the root element:
</note>

You can assume, from this example, that the XML document contains a note to Tove from Jani. Don't you agree that XML is pretty self-descriptive? XML documents form a tree structure XML documents must contain a root element. This element is "the parent" of all other elements. The elements in an XML document form a document tree. The tree starts at the root and branches to the lowest level of the tree. All elements can have sub elements (child elements):

```
<root>
  <child>
    <subchild>.....</subchild>
  </child>
</root>
```

The terms parent, child, and sibling are used to describe the relationships between elements. Parent elements have children. Children on the same level are called siblings (brothers or sisters). All elements can have text content and attributes (just like in html).

Example:

```
<bookstore>
  <book category="cooking">
    <title lang="en">Everyday Italian</title>
    <author>giada de laurentiis</author>
    <year>2005</year>
    <price>30.00</price>
  </book>
  <book category="children">
    <title lang="en">harry potter</title>
    <author>j k. rowling</author>
    <year>2005</year>
    <price>29.99</price>
  </book>
  <book category="web">
    <title lang="en">learning XML</title>
    <author>erik t. ray</author>
    <year>2003</year>
    <price>39.95</price>
  </book>
</bookstore>
```

The root element in the example is <bookstore>. All <book> elements in the document are contained within <bookstore>. The <book> element has 4 children: <title>, <author>, <year>, and <price>.

XML syntax rules:

The syntax rules of XML are very simple and logical. The rules are easy to learn, and easy to use. All XML elements must have a closing tag. In HTML, Elements do not have to have a closing tag:

```
<p>this is a paragraph
<p>this is another paragraph
```

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

this is a paragraph
this is another 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.

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.

XML elements must be properly nested: In html, you might see improperly nested elements:

this text is bold and italic <!--This is wrong-->

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.

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>
```

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. Study the two XML documents below. The first one is incorrect, the second is correct:

- <note date=12/11/2007>
  <to>tove</to>
  <from>jani</from>
  </note>
- <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.

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 predefined 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.

Comments in XML: The syntax for writing comments in XML is similar to that of html.

<!-- this is a comment -->

white-space is preserved in XML

HTML truncates multiple white-space characters to one single white-space:

HTML: hello tove

Output: hello tove

With XML, the white-space in a document is not truncated.

XML stores new line as lf in windows applications, a new line is normally stored as a pair of characters: carriage return (cr) and line feed (lf). In UNIX applications, a new line is normally stored as an lf character. Macintosh applications also use an lf to store a new line. XML stores a new line as lf.

Basic Building Blocks of XML Documents:

All XML documents are made up the following four building blocks:

i. Elements/Tags
ii. Attributes
iii. Entities
iv. Character Data
    a. Parsed Character Data (PCDATA)
    b. Unparsed Character Data (CDATA)

i) XML Elements/Tags:

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.

XML Naming Rules:

XML elements must follow these naming rules:

- Names can contain letters, numbers, and other characters
- Names cannot start with a number or punctuation character
- Names cannot start with the letters XML (or XML, or XML, etc)
- Names cannot contain spaces
- Any name can be used, no words are reserved.

Best Naming Practices:

- Make names descriptive. Names with an underscore separator are nice: <first_name>, <last_name>.
- Names should be short and simple, like this: <book_title> not like this: <the_title_of_the_book>.
- Avoid "-" characters. If you name something "first-name," some software may think you want to subtract name from first.
- Avoid "." characters. If you name something "first.name," some software may think that "name" is a property of the object "first."
- Avoid ":" characters. Colons are reserved to be used for something called namespaces (more later).
- XML documents often have a corresponding database. a good practice is to use the naming rules of your database for the elements in the XML documents.
- Non-english letters like éòá are perfectly legal in XML, but watch out for problems if your software vendor doesn't support them.

XML Elements are Extensible:

XML elements can be extended to carry more information. look at the following XML example:

```
<note>
    <to>Chythu</to>
    <from>Ravi</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: Chythu
from: Ravi
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>Chythu</to>
    <from>Ravi</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.

ii) XML Attributes:

XML elements can have attributes, just like html. Attributes provide additional information about an element. In html, attributes provide additional information about elements:

```
<img src="computer.gif">
<a href="demo.asp">
```

Attributes often provide information that is not a part of the data. In the example below, the file type is irrelevant to the data, but can be important to the software that wants to manipulate the element:

```
<file type="gif">computer.gif</file>
```

XML attributes must be quoted: Attribute values must always be quoted. Either single or double quotes can be used. For a person's sex, the person element can be written like this:

```
<person sex="female">
```

or like this:

```
<person sex='female'>
```

If the attribute value itself contains double quotes you can use single quotes, like in this example:

```
<gangster name='george "shotgun" ziegler'>
```

or you can use character entities:

<gangster name="george "shotgun" ziegler">

XML Elements v/s Attributes:

Take a look at these examples:

- <person gender="male">
  <firstname>Ravi</firstname>
  <lastname>Chythanya</lastname>
  </person>
- <person>
  <gender>male</gender>
  <firstname>Ravi</firstname>
  <lastname>Chythanya</lastname>
  </person>

In the first example gender is an attribute. In the next, gender is an element. Both examples provide the same information. There are no rules about when to use attributes or when to use elements. Attributes are handy in html. In XML my advice is to avoid them. Use elements instead.

My favorite way:

The following three XML documents contain exactly the same information:

1) A date attribute is used in the first example:

```
<note date="10/01/2008">
    <to>tove</to>
    <from>jani</from>
    <heading>reminder</heading>
    <body>don't forget me this weekend!</body>
</note>
```

2) A date element is used in the second example:

```
<note>
    <date>10/01/2008</date>
    <to>tove</to>
    <from>jani</from>
    <heading>reminder</heading>
    <body>don't forget me this weekend!</body>
</note>
```

3) An expanded date element is used in the third: (this is my favorite):

```
<note>
    <date>
        <day>10</day>
        <month>01</month>
        <year>2008</year>
    </date>
    <to>tove</to>
    <from>jani</from>
    <heading>reminder</heading>
    <body>don't forget me this weekend!</body>
</note>
```

Avoid XML attributes?

Some of the problems with using attributes are:

- Attributes cannot contain multiple values (elements can)
- Attributes cannot contain tree structures (elements can)
- Attributes are not easily expandable (for future changes)
- Attributes are difficult to read and maintain. use elements for data. use attributes for information that is not relevant to the data.
- Don't end up like this:
  <note day="10" month="01" year="2008" to="tove" from="jani" heading="reminder" body="don't forget me this weekend!"> </note>

XML attributes for metadata:

Sometimes ID references are assigned to elements. These IDs can be used to identify XML elements in much the same way as the id attribute in html. This example demonstrates this:

```
<messages>
    <note id="501">
        <to>tove</to>
        <from>jani</from>
        <heading>reminder</heading>
        <body>don't forget me this weekend!</body>
    </note>
    <note id="502">
        <to>jani</to>
        <from>tove</from>
        <heading>re: reminder</heading>
        <body>i will not</body>
    </note>
</messages>
```

The ID attributes above are for identifying the different notes. It is not a part of the note itself. What I'm trying to say here is that metadata (data about data) should be stored as attributes, and the data itself should be stored as elements.

iii) XML Entities:

An entity is the symbolic representation of some information. It references the data that look like an abbreviation or can be found at an external location. Entities reduce the redundant information and also allow for easier editing. The entities can be internal or external. For example, the entity **'&'** will replace as **'&'**, where ever it is found in the XML document.

iv) PCDATA:

PCDATA stands for Parsed Character Data. The text enclosed between starting and ending tag is known as Character Data. The parser parses the PCDATA and identifies the entities as well as markup. Markups include the tags inside the text and entities are expanded. In the case of Parsed Character Data, the symbols '&', '<', '>' are represented using entities.

v) CDATA:

CDATA stands for Character Data and it is not parsed by the parser. The tags inside the text are not considered as markup and the entities are not expanded.

Validation of XML:
There are two levels of correctness of an XML document:

1) XML with correct syntax is **Well Formed XML**.
2) XML validated against a DTD is **Valid XML**.

A **Well Formed XML** document is a document that conforms to the XML syntax rules like:

- XML documents must have a root element.
- XML elements must have a closing tag.
- XML tags are case sensitive.
- XML elements must be properly nested.
- XML attribute values must be quoted

Example:

```
<?XML version="1.0" encoding="iso-8859-1"?>
<note>
        <to>tove</to>
        <from>jani</from>
        <heading>reminder</heading>
        <body>don't forget me this weekend!</body>
</note>
```

A **Valid XML** document is a "well formed" XML document, which conforms to the rules of a document type definition (DTD).

Document Type Definition (DTD):

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

DTD Syntax:

```
<!DOCTYPE DOCUMENT [
<!ELEMENT ELEMENT_NAME1 (Attribute Names) Appearance of attributes>
<!ELEMENT ELEMENT_NAME2>
<!ELEMENT ELEMENT_NAME3>
.
.
.
<!ELEMENT ELEMENT_NAMEn>
<!ATTLIST Element_name Attribute_name Attribute_type Default_value>
]>
```

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 DOCTYPE definition as in the following example:

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

```
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#CDATA)>
]>
<note>
        <to>Ravi</to>
        <from>Chythanya</from>
        <heading>Message</heading>
        <body>Welcome to the XML with DTD</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 that to element to be of the type #PCDATA.
- !ELEMENT from defines that from element to be of the type #PCDATA.
- !ELEMENT heading defines that heading element to be of the type #PCDATA.
- !ELEMENT body defines that body element to be of the type #CDATA.

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 DOCUMENT SYSTEM "File_Name.dtd">
```

Example: emailDTD.xml:

```
<?xml version="1.0"?>
<!DOCTYPE note SYSTEM "emailDTD.dtd">
<note>
        <to>Ravi</to>
        <from>Chythanya</from>
        <heading>Message</heading>
        <body>Welcome to the XML with DTD</body>
</note>
```

emailDTD.dtd:

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

XML Schema:

W3C supports an XML-based alternative to DTD, called XML schema:

1) The XML schema is used to represent the structure of XML document. The goal or purpose of XML schema is to define the building blocks of an XML document. These can be used as an alternative to XML DTD. The XML schema language is called as XML Schema Definition (XSD) language. The XML schema became the World Wide Web Consortium (W3C) recommendation in 2001.

2) XML schema defines elements, attributes, elements having child elements, order of child elements. It also defines fixed and default values of elements and attributes.
3) XML schema also allows the developers to use data types.
4) File extension of XML schema is ".xsd" i.e., Filename.xsd

Ex:

StudentSchema.xsd:

```
<?XML version="1.0"?>
<xs:schema XMLns:xs="http://www.w3.org/2001/XMLschema">
        <xs:element name="student">
                <xs:complextype>
                <xs:sequence>
                        <xs:element name="name" type="xs:string"/>
                        <xs:element name="address" type="xs:string"/>
                        <xs:element name="std" type="xs:string"/>
                        <xs:element name="marks" type="xs:string"/>
                </xs:sequence>
                </xs:complextype>
        </xs:element>
</xs:schema>
```

XML Document (myschema.XML):

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<student XMLns:xsi = "http://www.w3.org/2001/XMLschema-instance"
xsi:nonamespaceschemalocation = "studentschema.xsd">
        <name>anand</name>
        <address>knr</address>
        <std>second</std>
        <marks>70percent</marks>
</student>
```

Data Types in XML:

Various data types are that can be used to specify the data types of an elements are:

- String
- Date
- Time
- Numeric
- Boolean

1. String Data Type:

The string data type can be used to define the elements containing the characters lines, tabs, white spaces, or special characters.

Example:

XML Schema [stringType.xsd]:

```
<?xml version= "1.0" ?>
<xs:schema xmlns:xs=" http://www.w3.org/2001/XMLschema">
        <xs:element name= "StudentName" type="xs:string">
</xs:schema>
```

XML Document[stringTypeDemo.xml]:

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<StudentName XMLns:xsi = "http://www.w3.org/2001/XMLschema-instance"
xsi:nonamespaceschemalocation = "stringType.xsd">
        Ravi Chythanya
</StudentName>
```

2. Date Data Type:

For specifying the date we use date data type. The format of date is yyyy-mm-dd, where yyyy denotes the Year, mm denotes the month and dd specifies the day.

Example:

XML Schema [dateType.xsd]:

```
<?xml version= "1.0" ?>
<xs:schema xmlns:xs=" http://www.w3.org/2001/XMLschema">
        <xs:element name= "DOB" type="xs:date">
</xs:schema>
```

XML Document[dateTypeDemo.xml]:

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<DOB XMLns:xsi = "http://www.w3.org/2001/XMLschema-instance"
xsi:nonamespaceschemalocation = "dateType.xsd">
        1988-03-01
</DOB>
```

3. Time Data Type:

For specifying the time we use time data type. The format of date is hh:mm:ss, where hh denotes the Hour, mm denote the minutes and ss specify the seconds.

Example:

XML Schema [timeType.xsd]:

```
<?xml version= "1.0" ?>
<xs:schema xmlns:xs=" http://www.w3.org/2001/XMLschema">
        <xs:element name= "MyTime" type="xs:time">
</xs:schema>
```

XML Document[timeTypeDemo.xml]:

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<MyTime XMLns:xsi = "http://www.w3.org/2001/XMLschema-instance"
xsi:nonamespaceschemalocation = "timeType.xsd">
        01:57:03
</MyTime>
```

4. Numeric Data Type:

If we want to use numeric value for some element then we can use the data types as either **"decimal"** or **"integer".**

Example:

XML Schema [decimalType.xsd]:

```
<?xml version= "1.0" ?>
```

```
<xs:schema xmlns:xs=“ http://www.w3.org/2001/XMLschema”>
        <xs:element name= “Percentage” type=“xs:decimal”>
</xs:schema>
```

XML Document[decimalTypeDemo.xml]:

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<Percentage XMLns:xsi = "http://www.w3.org/2001/XMLschema-instance"
xsi:nonamespaceschemalocation = "decimalType.xsd">
        99.99
</Percentage>
```

XML Schema [integerType.xsd]:

```
<?xml version= “1.0” ?>
<xs:schema xmlns:xs=“ http://www.w3.org/2001/XMLschema”>
        <xs:element name= “Age” type=“xs:integer”>
</xs:schema>
```

XML Document[integerTypeDemo.xml]:

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<Age XMLns:xsi = "http://www.w3.org/2001/XMLschema-instance"
xsi:nonamespaceschemalocation = "integerType.xsd">
        30
</Age>
```

5. Boolean Data Type:

For specifying either **true** or **false** values we must use the Boolean data type.

Example:

XML Schema [booleanType.xsd]:

```
<?xml version= “1.0” ?>
<xs:schema xmlns:xs=“ http://www.w3.org/2001/XMLschema”>
        <xs:element name= “Flag” type=“xs:boolean”>
</xs:schema>
```

XML Document[booleanTypeDemo.xml]:

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<Flag XMLns:xsi = "http://www.w3.org/2001/XMLschema-instance"
xsi:nonamespaceschemalocation = "booleanType.xsd">
        true
</Flag>
```

Simple Types:

XML defines the simple type which contains only text and does not contain any other element or attributes. But the **text** appears for element comes along with some **type**. The syntax is:

<**xs:element name**= “element_name” **type**= “data_type”>

Here the type can be any built in data type:

- xs:string
- xs:date

- xs:time
- xs:integer
- xs:decimal
- xs:Boolean

Restrictions on Simple Types:

For an XML document, it is possible to put certain restrictions on its contents. These restrictions are called as **facets.**

XML Schema [Month.xsd]:

```
<?xml version= "1.0" ?>
<xs:schema xmlns:xs=" http://www.w3.org/2001/XMLschema">
        <xs:element name= "Month">
                <xs:simpleType>
                        <xs:restriction base= "xs:integer">
                                <xs:minInclusive value= "1">
                                <xs:maxInclusive value= "12">
                        </xs:restriction>
                </xs:simpleType>
        </xs:element>
</xs:schema>
```

XML Document[MonthDemo.xml]:

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<Age xmlns:xsi = "http://www.w3.org/2001/XMLschema-instance"
xsi:nonamespaceschemalocation = "Month.xsd">
        12
</Month>
```

Various facets that can be applied are:

- length – Specifies exact number of characters.
- minLength – Specifies the minimum number of characters allowed.
- maxLength – Specifies the minimum number of characters allowed.
- minInclusive – Specifies lower bound for numeric values.
- minExclusive – Specifies lower bound for numeric values.
- maxInclusive – Specifies upper bound for numeric values.
- maxExclusive – Specifies upper bound for numeric values.
- enumeration – A list of acceptable values can be defined.
- pattern – specifies exact sequence of characters that are acceptable.
- totalDigits – Specifies the exact number of digits.
- fractionDigits – Specifies maximum number of decimal places allowed.
- whiteSpace – Specifies how to handle white spaces. The white spaces can be spaces, tags, line feed, or carriage return.

Restrictions that can be applied on the data types are:

string	date	numeric	boolean
enumeration	enumeration	enumeration	enumeration
Length	maxExclusive	fractionDigits	length
minLength	maxInclusive	maxExclusive	maxLength
maxLength	minExclusive	maxInclusive	minLength
Pattern	minInclusive	minExclusive	pattern
whiteSpaces	pattern	minInclusive	whiteSpaces
	whiteSpaces	totalDigits	
		whiteSpaces	

Complex Types:

1) **Elements:**

Complex elements are the elements that contain some other elements or attributes.
The complex types of elements are:

i) Empty Elements
ii) Elements that contain text
iii) Elements that contain other elements
iv) Elements that contain other text as well other elements

i) Empty Elements:

Let us define the empty elements by the complexType.

Example [Student.xsd]:

```
<?xml version= "1.0" ?>
<xs:schema xmlns:xs=" http://www.w3.org/2001/XMLschema">
    <xs:element name= "Student">
            <xs:complexType>
                    <xs:attribute name= "HTNo" type= "xs:integer">
            </xs:complexType>
    </xs:element>
</xs:schema>
```

XML Document[Student.xml]:

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<Student xmlns:xsi = "http://www.w3.org/2001/XMLschema-instance"
xsi:nonamespaceschemalocation = "Student.xsd" HTNo="1343">
</Student>
```

ii) Elements that contain text only:

We have the elements that contain simple contents such as text and attributes. Hence we must add the type as **simpleContent** in the schema definition. This simpleContent element must have either extension or restriction.

Example [Student.xsd]:

```
<?xml version= "1.0" ?>
<xs:schema xmlns:xs=" http://www.w3.org/2001/XMLschema">
    <xs:element name= "Student">
            <xs:complexType>
```

```
                    <xs:simpleContent>
                        <xs:extension base= “xs:integer”>
                            <xs:attribute name= “HTNo” type= “xs:integer”>
                        </xs:extension>
                    </xs:simpleContent>
                </xs:complexType>
            </xs:element>
        </xs:schema>
```

XML Document[Student.xml]:

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<Student xmlns:xsi = "http://www.w3.org/2001/XMLschema-instance"
xsi:nonamespaceschemalocation = "Student.xsd" HTNo="1343">
    100
</Student>
```

iii) Elements that contain other elements only:

Example [Student.xsd]:

```
<?xml version= “1.0” ?>
<xs:schema xmlns:xs=“ http://www.w3.org/2001/XMLschema”>
    <xs:element name= “Student”>
        <xs:complexType>
            <xs:sequence>
                <xs:element name= “firstName” type= “xs:string”>
                <xs:element name= “lastName” type= “xs:string”>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

XML Document[Student.xml]:

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<Student xmlns:xsi = "http://www.w3.org/2001/XMLschema-instance"
xsi:nonamespaceschemalocation = "Student.xsd">
    <firstName>Ravi</firstName>
    <lastName>Chythanya</lastName>
</Student>
```

iv) Elements that contain text as well as other elements:

Example [Student.xsd]:

```
<?xml version= “1.0” ?>
<xs:schema xmlns:xs=“ http://www.w3.org/2001/XMLschema”>
    <xs:element name= “Student”>
        <xs:complexType mixed= “true”>
            <xs:sequence>
                <xs:element name= “Name” type= “xs:string”>
```

```
                    <xs:element name=  "HTNo" type= "xs:integer">
                </xs:sequence>
            </xs:complexType>
        </xs:element>
    </xs:schema>
```

XML Document[Student.xml]:

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<Student xmlns:xsi = "http://www.w3.org/2001/XMLschema-instance"
xsi:nonamespaceschemalocation = "Student.xsd">
    My Dear Student <Name>Ravi</Name>
    With Hall Ticket Number <HTNo>0536</HTNo>
</Student>
```

2) Indicators:

Indicators allow us to define the elements in the manner we want. Following are some indicators that are available in XML Schema.

i) all
ii) choice
iii) sequence
iv) maxOccurs
v) minOccurs

Document Object Model (DOM):

A DOM (Document Object Model) defines a standard way for accessing and manipulating XML documents.

> DOM is a set of platform independent and language neutral Application Programming Interface (API) which describes how to access and manipulate the information stored in XML or in HTML documents.

Thus XML DOM is for

- **Loading** the XML document.
- **Accessing** the elements of XML document
- **Deleting** the elements of XML document
- **Changing** the elements of XML document

1) Loading the XML document:

We will write a simple script by which any desired XML file can be loaded. Note these scripts are dependent upon the browser in which these are getting opened.

Ex: loadXML.html:

```
<html>
    <!—simple DOM example for loading some XML file-->
    <body>
        <script type= "text/javascript">
            try
```

```
                {
                        xmlDoc=new ActiveXObject("Microsoft.XMLDOM");
                }
                catch(Exception e)
                {
                        try
                        {
                           xmlDoc=document.implementation.createDocument("","",null);
                        }
                        catch(Exception e)
                        {
                           alert(e.message);
                        }
                }
                try
                {
                        xmlDoc.async=false;
                        xmlDoc.load("student.xml");
                        document.write("XML Document student.xml file is loaded");
                }
                catch(Exception e)
                {
                        alert(e.message);
                }
            </script>
        </body>
</html>
```

student.xml:

```
<?xml version= "1.0" ?>
<student>
        <HTNo>1234</HTNo>
        <PersonalInfo>
                <Name>Chythu</Name>
                <City>WGL</City>
        </PersonalInfo>
</student>
```

Properties of XML DOM:

Consider a node 'z'. Few distinct DOM properties are as follows:

- nodeName – represents the name of the node. For example, z.nodeName.
- nodeValue – used to know the value of a the node. For example, z.nodeValue.
- parentNode – used to find the parent node of the given node. For example, z.parentNode.
- childNodes – represents all the children of the node. For example, z.childNodes.
- attributes – represents the attributes of the given node for example, z.attributes.

Methods in XML DOM:

Following are the XML DOM methods:

- z.getElementByTagName(tag_name) – used to get all the elements with the given tag name.
- z.appendChild(node) – inserts a child node to node z.
- z.removeChild(node) – from node z, a child node is removed.

Parsing XML Data:

XML parser is the software which acts as an interpreter for unique xml. The primary goal of any XML processor is to parse the given XML document. Java has a rich source of in-built APIs for parsing the given XML document.

XML parser can be classified into two types:

1) DOM Parser
2) SAX Parser

Difference between DOM and SAX:

DOM	SAX
DOM is a tree based parsing method used to parse the given XML document.	SAX is an event based parsing method used to parse the given XML document.
In this method the entire XML document is stored in the memory before actual processing. Hence it requires more memory	In this method, parsing is done by generating the sequence of events or it calls handler functions.
The DOM approach is useful for smaller applications because it is simpler to use but it is certainly not used for larger XML documents because it will then require larger amount of memory.	Although SAX development is much more challenging .it is useful for parsing the large XML document because the approach is event based, XML gets parsed node by node and does not require large amount of memory.
We can insert or delete a node	We can insert or delete a node
Traversing is done in any direction in DOM approach.	Top to bottom traversing is done in SAX approach.

DOM Parser:

To access and modify the XML documents, the DOM (Document Object Model) provides a standard method. The DOM parser verifies the entire XML document. Then the data in XML document is arranged in a hierarchical manner and is loaded into temporary memory. XML DOM views an XML document as a tree like structure. All elements can be accessed through the DOM tree. Their content (text and attributes) can be modified or deleted, and new elements can be created. The elements, their text, and their attributes are all known as nodes.

Example:

Load an XML String - Cross-browser Example: The following example parses an XML string into an XML DOM object and then extracts some info from it with a JavaScript:

```
<html>
<body>
<h1>W3Schools Internal Note</h1>
<p><b>To:</b> <span id="to"></span><br />
```

```
<b>From:</b> <span id="from"></span><br />
<b>Message:</b> <span id="message"></span>
<script type="text/javascript">
if (window.XMLHttpRequest)
{// code for IE7+, Firefox, Chrome, Opera, Safari
  XMLhttp=new XMLHttpRequest();
}
else
{// code for IE6, IE5
  XMLhttp=new ActiveXObject("Microsoft.XMLHTTP");
}
XMLhttp.open("GET","note.XML",false);
XMLhttp.send();
XMLDoc=XMLhttp.responseXML;

document.getElementById("to").innerHTML=
XMLDoc.getElementsByTagName("to")[0].childNodes[0].nodeValue;

document.getElementById("from").innerHTML=
XMLDoc.getElementsByTagName("from")[0].childNodes[0].nodeValue;

document.getElementById("message").innerHTML=
XMLDoc.getElementsByTagName("body")[0].childNodes[0].nodeValue;
</script>
</body>
</html>
```

DOMParser.java:

```
import java.io.File;
import javax.xml.parsers.*;
import org.w3c.dom.*;
import java.util.Scanner;
public class DOMParser
{
public static void main(String[] args)throws Exception
{
DocumentBuilderFactory fac=DocumentBuilderFactory.newInstance();
DocumentBuilder b=fac.newDocumentBuilder();
Document doc=b.parse(new File("UsersList.xml"));
doc.getDocumentElement().normalize();
Element root=doc.getDocumentElement();
NodeList nl=doc.getElementsByTagName("user");
System.out.println(root.getNodeName());
System.out.println("=====================================");
for(int i=0;i<nl.getLength();i++)
{
Node node=nl.item(i);
```

```
if(node.getNodeType()==Node.ELEMENT_NODE)
{
Element e=(Element)node;
System.out.println("\nHTNo:\t"+ e.getElementsByTagName("htno").item(0).getTextContent());
System.out.println("Name:\t"+ e.getElementsByTagName("name").item(0).getTextContent());
System.out.println("Addr:\t"+ e.getElementsByTagName("address").item(0).getTextContent());
System.out.println("Branch:\t"+ e.getElementsByTagName("branch").item(0).getTextContent());
}
}
}
}
```

UsersList.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<users-details>
   <user gender="Male">
      <htno>501</htno>
      <name>Ravi</name>
      <branch>CSE</branch>
      <address>Warangal</address>
   </user>
   <user gender="Male">
      <htno>401</htno>
      <name>Chythaya</name>
      <branch>ECE</branch>
      <address>Karimnagar</address>
   </user>
   <user gender="Male">
      <htno>301</htno>
      <name>Madan</name>
      <branch>MECH</branch>
      <address>Hyderabad</address>
   </user>
   <user gender="Female">
      <htno>201</htno>
      <name>Raji</name>
      <branch>EEE</branch>
      <address>Warangal</address>
   </user>
   <user gender="Female">
      <htno>101</htno>
      <name>Rani</name>
      <branch>CIVIL</branch>
      <address>Hyderabad</address>
   </user>
   <user gender="Male">
      <htno>1201</htno>
```

```
    <name>Srinu</name>
    <branch>IT</branch>
    <address>Nalgonda</address>
  </user>
  <user gender="Male">
    <htno>502</htno>
    <name>Praveen</name>
    <branch>CSE</branch>
    <address>Medak</address>
  </user>
  <user gender="Male">
    <htno>402</htno>
    <name>Naveen</name>
    <branch>ECE</branch>
    <address>Siricilla</address>
  </user>
  <user gender="Male">
    <htno>302</htno>
    <name>Ramesh</name>
    <branch>Mech</branch>
    <address>Kothagattu</address>
  </user>
  <user gender="Male">
    <htno>202</htno>
    <name>Rahul</name>
    <branch>EEE</branch>
    <address>Siddipet</address>
  </user>
</users-details>
```

SAX Parser:

SAX is the abbreviation for Simple API for XML. Based on the event occurred, the SAX parser feeds the client application with document data. Therefore SAX is known as event based parser. SAX events can be stated as methods that are attached to specific Java interfaces. Some of these registers and methods are implemented as event-handlers by an application. SAX events can be grouped into few interfaces:

- Events related to documents are defined by Document Handler.
- Events related to DTDs are defined by DTD Handler.
- Events related to loading entities are defined by Entity Resolver.
- Error events are defined by Error Handler.

The events related to documents are defined by **Document Handler** which are as follows:

- startDocument()/endDocument()
- characters()/ignorableWhiteSpace()
- eventProcessingInstruction()
- setDocumentLocator()

Example:

```
import java.io.*;
import java.util.*;
import javax.xml.parsers.*;
import org.xml.sax.*;
import org.xml.sax.helpers.*;
public class SAXParser
{
	public static void main(String[] args) throws IOException
	{
		System.out.println("Enter the name of the XML document:");
		Scanner s=new Scanner(System.in);
		String fname=s.next();
		File fp=new File(fname);
		if(fp.exists())
		{
			try
			{
				SAXParserFactory parserFact=SAXParserFactory.newInstance();
				SAXParser parser=parserFact.newSAXParser();
				System.out.println("XML elements from the file "+fname+" are");
				DefaultHandler handler=new DefaultHandler()
				{
					public void startElement(String uri, String localName,
					String  elements, Attributes attributes) throws
					SAXException
					{
						System.out.println("\t"+elements);
					}
				};
				parser.parse(fname,handler);
			}
			catch(Exception e)
			{
				e.printStackTrace();
			}
		}
		else
			System.out.println("File not exist");
	}
}
```