**eXtensible Mark-up Language (XML)**

XML is "Programing Language & Platform Independent Language" which helps to store and transport data

Different Applications which are developed using different technologies can Transfer the Data among themselves with the help of XML

As the name implies it's an extension of HTML & hence XML looks similar to HTML but it’s not a HTML

XML has User-defind Tags. XML tags are also called as "elements"

XML Elements are "Case Sensitive"

XML is "Strictly Typed" Language hence,

- For every element data, “data-type” should be defined,

- every opening element should have corresponding closing element and

- also XML elements must be properly nested/closed

**Ex:**

<employee>

<name>Praveen</name>

</employee>

**Note:-**

In the above example first you should closed </name> & then </employee> but in HTML it’s not mandatory. For example, <B><U><I>My Text</U></I></B> works perfectly fine

Below line is called as "XML prolog", which is optional. If it exists, it must be the First Line of XML

<?xml version="1.0" encoding="UTF-8" ?>

The syntax of XML comment is similar to that of HTML

<!-- This is a comment -->

File extension of XML is ".xml"

MIME type (Content Type) of XML is "application/xml"

**1. XML Structure**

Like HTML, XML follows a Tree Structure

An XML tree starts at a "root element" and branches from "root element" will have "child elements"

XML Consists of "Only One" root element which is parent of all other elements

"child elements" can have "sub elements / child elements"

Structure

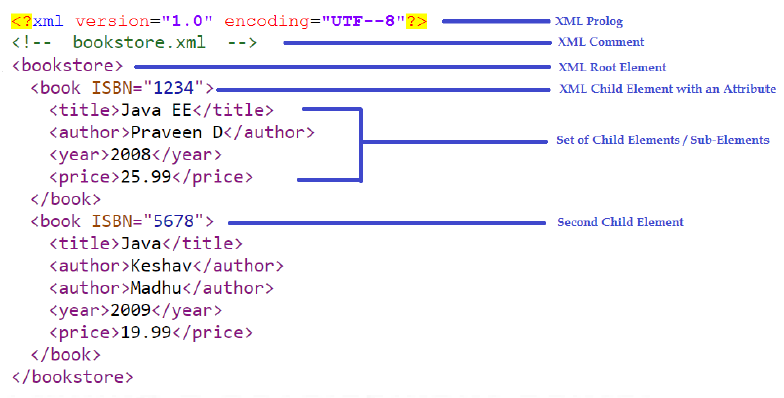
<root>

<child>

<subchild>.....</subchild>

</child>

</root>



**2. 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 it represents the start of a new element

**Ex:** <message>salary<1000</message>

To avoid this error, we can replace the "<" character with an "entity reference" as shown below

<message>salary **&lt;** 1000</message>

There are 5 pre-defined entity references in XML:

&lt; < less than

&gt; > greater than

&amp; & ampersand

&apos; ' apostrophe

&quot; " quotation mark

**3. PCDATA: Parsed Character Data**

Text between start-element and end-element is called as PCDATA which will be examined by the parser

**Example:-**

<employee>Praveen</employee>

The string "Praveen" is considered as PCDATA

**4. CDATA: Character Data**

W.K.T special characters (such as "<", "&") must be referenced through pre-defined entities

If XML data contain many special characters, it is cumbersome to replace all of them. Instead we can use "CDATA (character data) section"

A CDATA section starts with the following sequence:

<![CDATA[

and ends with the next occurrence of the sequence:

]]>

All characters enclosed between these two sequences are interpreted as characters

The XML parsers ignores all the mark-up within the CDATA section.

**Example: -**

<employee>Praveen</employee>

the start and end "employee" elements are interpreted as mark-up. However, if written like this:

<![CDATA[ <employee>Praveen</employee> ]]>

then the parsers interprets the same as if it had been written like this:

**&lt;**employee**&gt;**Praveen**&lt;**/employee**&gt;**

**5. XML Elements**

XML element is everything from (including) the element's start tag to (including) the element's end tag

An element can contain:

1. data

2. Attributes

3. other elements OR

4. All of the above

In the above example

- <title>, <author>, <year>, and <price> have text content

- <bookstore> and <book> have element contents

- <book> has an attribute (ISBN="------")

An element with no content is said to be "empty". In XML, we can indicate an empty element like this

<element></element>

OR

<element />

Empty elements can have attributes <book ISBN="5678" />

If data present between elements consist of white spaces then they are considered in XML. However HTML truncates multiple white-spaces to one single white-space

**6. XML Elements Naming Rules**

they are case-sensitive

they cannot contain spaces

they must start with a letter or underscore

they are cannot start with the letters like xml or XML or Xml etc.,

they can contain letters, digits, hyphens, underscores, and periods

Any name can be used, no words are reserved (except xml)

**Best Naming Practices**

Avoid "." and ":"

Create descriptive names, like

<person>, <firstname>, <lastname>

Create short and simple names, like

<book\_title> not like this: <the\_title\_of\_the\_book>

Non-English letters are perfectly legal in XML but avoid them

**7. XML Attributes**

Like HTML, XML elements can also have attributes

Attributes are designed to contain data related to a specific element

XML Attributes Must be Quoted either single or double quotes can be used

**Ex:**

<person gender="female">

OR

<person gender='female'>

If the attribute value itself contains double quotes then we can use single quotes

**Ex:**

<person name='Praveen "Bangalore" D'>

OR

<person name='Praveen **&quot;**Bangalore**&quot;** D'>

**8. XML Elements v/s Attributes**

**Example 1:-**

<person gender="male">

<name>Praveen</name>

</person>

**Example 2:-**

<person>

<gender>male</gender>

<name>Praveen</name>

</person>

**Note:**

- In Example 1 gender is an attribute &

- In Example 2 gender is an element

- Both examples provide the same information

- There are no rules about when to use attributes or when to use elements in XML

**When to avoid XML Attributes?**

Attributes cannot contain multiple values but Elements can

Attributes cannot contain tree structures but Elements can

Attributes are not easily expandable for future changes but Elements can

**9. XML Schema's**

W.K.T XML helps us to store & transfer the data

When sending data from one application to an another, it is essential that both applications have the same "expectations / agreement" about the content/data

for example, A date like "03-11-2004"

- in some countries, be interpreted as 3rd November and

- in other countries as 11th March

With XML Schemas, the sender application can describe the data in a way that the receiver application will understand

Schema is nothing but a "Structure". It is a formal description of structure of an XML.

- i.e., which elements are allowed,

- which elements must be present,

- which elements are optional,

- the sequence and relationship of the elements, etc.,

For example,

- abc@gmail.com is a Valid Email ID. However

- abc#gmail is Invalid because there is "NO @ and ."

- hence email schema looks something like some-name@domain-name.com

Schema "does not validate the data" instead "it validates the structure"

There are two ways to define a Schema for XML

1. Document Type Definition (DTD)

2. XML Schema Definition (XSD)

**JavaScript Object Notation [JSON]**

**1. Examples of Data Formats:**

**String**

String str1 = "123praveen200.12";

String str2 = "EmpID=123|EmpNM=praveen|EmpSal=200.12";

**XML**

<employee>

<emp-id>123</emp-id>

<emp-name>Praveen</emp-name>

<emp-salary>200.12</emp-salary>

</employee>

**JSON**

{"EmpID":123, "EmpNM":"Praveen", "EmpSal":200.12}

Like XML, JSON also is a "Programing Language & Platform Independent Language" which helps to store and transport data

However compared to XML, it’s a lightweight, easy for applications to parse and generate by avoiding complicated parsing and translations

JSON is a "text format" but uses conventions that are familiar to programmers of the C-family of languages (C, C++, C#, Java, JavaScript, Perl, Python, etc.,). Hence JSON is an "ideal data interchange language"

JSON, as the name implies, which consists of data similar to "Object Notation of JavaScript". It's an extension of JavaScript scripting language and this format was specified by "Douglas Crockford in 2006"

Hence if we receive data from a server in JSON format, we can directly use it like any other JavaScript object

The filename extension of JSON is “.json"

MIME type (Content Type) of JSON is "application/json"

**2. JSON Syntax**

JSON syntax is derived from JavaScript object notation syntax:

- Data is in "name:value" pairs

- Data is separated by "commas"

- "Curly braces" hold objects

- "Square brackets" hold arrays

**i. JSON Data**

JSON data is written as name/value pairs. A name/value pair consists of

- a field name (**Should be** in double quotes)

- followed by a colon

- followed by a value

**Ex:** "employee-name" : "Praveen D"

**ii. JSON Values**

In JSON, values must be one of the following data types

1. String

2. Number

3. Boolean

4. NULL

5. an Object (JSON object)

6. an Array

In JSON,

- String values must be written with double quotes

- Numbers must be an integer/decimal values

- Boolean values must be true/false

- JSON NULL values must be null

**Ex:-**

{ "name":"Praveen D",

"age":33,

"isEmployed":true,

"girlFriend":null

}

**iii. JSON Objects**

Values in JSON can be objects

JSON Objects are

- surrounded by curly braces {}

- JSON object data is written in "key:value" pairs

- Each "key:value" pair is separated by a comma

- Keys must be String and Values must be a valid

- JSON data type (String, Number, Object, Array, Boolean or null)

**Ex:-**

{

"employee":{ "name":"Praveen D",

"age":33,

"isEmployed":true,

"girlFriend":null

}

}

Values in a JSON object can be another JSON object

{

"employee": {

"name":"Praveen D",

"age":33,

"isEmployed":true,

"girlFriend":null,

"cars": {

"car1":"GM",

"car2":"BMW",

"car3":"Audi"

}

}

}

**iv. JSON Arrays**

Values in JSON can be arrays

JSON Arrays are

- surrounded by "Square Brackets []"

- JSON Arrays values is separated by a comma

- Array values must be a valid JSON data type

- (String, Number, Object, Array, Boolean or null)

Example 1:-

{

"employees":[ "Praveen", "Rekha", "Malleshwar" ]

}

Example 2:-

{

"name":"Praveen",

"age":33,

"cars":[ "GM", "BMW", "Audi" ]

}

Values in an array can also be another array, or even another JSON object:

{

"name":"Praveen",

"age":33,

"cars": [

{ "name":"GM",

"models":[ "Aveo", "Beat", "Cruze" ]

},

{ "name":"Audi",

"models":[ "A3", "A7" ]

}

]

}

**JSON v/s XML**

Both JSON and XML can be used to store & get the data from a web server

However,

- XML has to be parsed with an XML parser

- JSON can be parsed by a standard JavaScript function

Hence, XML is much more difficult to parse than JSON

JSON is a ready-to-use JavaScript object

XML Data cannot consist of Arrays

JSON Data can consist of Arrays

In General,

- Web Applications interact with each other by exchanging data using XML

- Mobile Apps interact with Web Applications by exchanging data using JSON