XML

Extensible Markup Language (XML) is a markup language that defines a set of rules for encoding documents in a format that is both human-readable and machine-readable. The design goals of XML focus on simplicity, generality, and usability across the Internet. It is a textual data format with strong support via Unicode for different human languages. Although the design of XML focuses on documents, the language is widely used for the representation of arbitrary data structures such as those used in web services.

- XML stands for extensible Markup Language
- XML is a markup language like HTML
- XML is designed to store and transport data
- XML is designed to be self-descriptive

What is mark-up language?

A **mark up language** is a modern system for highlight or underline a documen

Why xml?

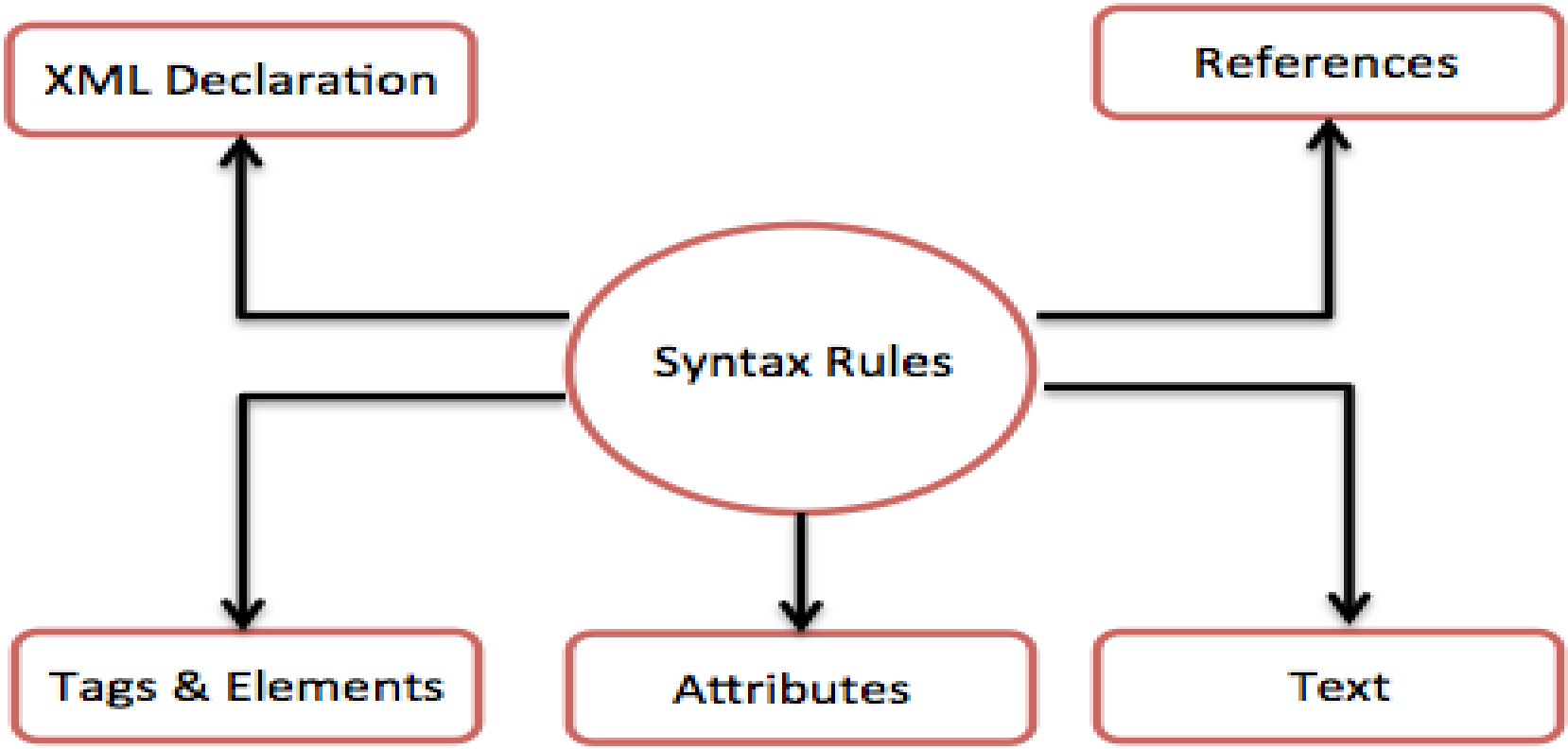
Platform Independent and Language Independent: The main benefit of xml is that you can use it to take data from a program like Microsoft SQL, convert it into XML then share that XML with other programs and platforms. You can communicate between two platforms which are generally very difficult. The main thing which makes XML truly powerful is its international acceptance. Many corporation use XML interfaces for databases, programming, office application mobile phones and more. It is due to its platform independent feature.

What is xml?

- **Xml** (eXtensible Markup Language) is a mark up language.
- XML is designed to store and transport data.
- Xml was released in late 90's. it was created to provide an easy to use and store self describing data.
- XML became a W3C Recommendation on February 10, 1998.
- XML is not a replacement for HTML.
- XML is designed to be self-descriptive.
- XML is designed to carry data, not to display data.
- XML tags are not predefined. You must define your own tags.
- XML is platform independent and language independent.

XML - Syntax

The following diagram depicts the syntax rules to write different types of markup and text in an XML document



XML Declaration

The XML document can optionally have an XML declaration. It is written as follows –

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

Where *version* is the XML version and *encoding* specifies the character encoding used in the document.

UTF-Unicode Transformation Format

Syntax Rules for XML Declaration

- The XML declaration is case sensitive and must begin with "**<?xml>**" where "**xml**" is written in lower-case.
- If document contains XML declaration, then it strictly needs to be the first statement of the XML document.
- The XML declaration strictly needs be the first statement in the XML document.
- An HTTP protocol can override the value of *encoding* that you put in the XML declaration.

Tags and Elements

An XML file is structured by several XML-elements, also called XML-nodes or XML-tags. The names of XML-elements are enclosed in triangular brackets < > as shown below –

```
<element>
```

```
<?xml version = "1.0" encoding = "UTF-8"?>
<note>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

The XML above is quite self-descriptive:

- It has sender information
- It has receiver information
- It has a heading
- It has a message body

Features and Advantages of XML

1) 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 focus on using HTML/CSS for display and layout, and be sure that changes in the underlying data will not require any changes to the HTML.

2) XML simplifies data sharing

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.

3) XML simplifies data transport

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.

4) XML simplifies Platform change

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.

5) XML increases data availability

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.

6) XML can be used to create new internet languages

A lot of new Internet languages are created with XML.

Here are some examples:

XHTML

WSDL (**Web Services for Devices)** for describing available web services

WAP (**wireless application protocol)** and **WML** (**Wireless Markup Language)**as markup languages for handheld devices

RSS (really simple syndication) languages for news feeds

RDF(Resource Description Framework) and **OWL (Web Ontology Language)** for describing resources and ontology

SMIL (**Synchronized Multimedia Integration Language)**for describing multimedia for the web

XML Naming Rules

XML elements must follow these naming rules:

- Element names are case-sensitive
- Element names must start with a letter or underscore
- Element names cannot start with the letters xml (or XML, or Xml, etc)
- Element names can contain letters, digits, hyphens, underscores, and periods
- Element names cannot contain spaces

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

Best Naming Practices

Create descriptive names, like this: <person>, <firstname>, <lastname>.

Create short and simple names, like this: <book_title> not like this: <the_title_of_the_book>.

Avoid "-". If you name something "first-name", some software may think you want to subtract "name" from "first".

Avoid ".". If you name something "first.name", some software may think that "name" is a property of the object "first".

Avoid ":". Colons are reserved for namespaces (more later).

Non-English letters like éòá are perfectly legal in XML, but watch out for problems if your software doesn't support them!

Naming Conventions

Some commonly used naming conventions for XML elements:

Style	Example	Description
Lower case	<firstname>	All letters lower case
Upper case	<FIRSTNAME>	All letters upper case
Snake case	<first_name>	Underscore separates words (commonly used in SQL databases)
Pascal case	<FirstName>	Uppercase first letter in each word (commonly used by C programmers)
Camel case	<firstName>	Uppercase first letter in each word except the first (commonly used in JavaScript)

The Building Blocks of XML Documents

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

- Elements
- Attributes
- Entities
- PCDATA
- CDATA

- **Elements**

Elements are the **main building blocks** of both XML and HTML documents.

Examples of HTML elements are "body" and "table". Examples of XML elements could be "note" and "message". Elements can contain text, other elements, or be empty. Examples of empty HTML elements are "hr", "br" and "img".

Examples:

<body>some text</body>

<message>some text</message>

•**Attributes**

Attributes provide **extra information about elements**.

Attributes are always placed inside the opening tag of an element. Attributes always come in name/value pairs. The following "img" element has additional information about a source file:

The name of the element is "img". The name of the attribute is "src". The value of the attribute is "computer.gif". Since the element itself is empty it is closed by a " /".

•**Entities**

Some characters have a special meaning in XML, like the less than sign (<) that defines the start of an XML tag.

Most of you know the HTML entity: " ". This "no-breaking-space" entity is used in HTML to insert an extra space in a document. Entities are expanded when a document is parsed by an XML parser.

The entities are predefined in XML are:

Entity References	Character
<	<
>	>
&	&
"	"
'	'

•**PCDATA**

PCDATA means parsed character data.

Think of character data as the text found between the start tag and the end tag of an XML element.

PCDATA is text that WILL be parsed by a parser. **The text will be examined by the parser for entities and markup**.

Tags inside the text will be treated as markup and entities will be expanded. However, parsed character data should not contain any &, <, or > characters; these need to be represented by the & < and > entities, respectively.

•**CDATA**

CDATA means character data.

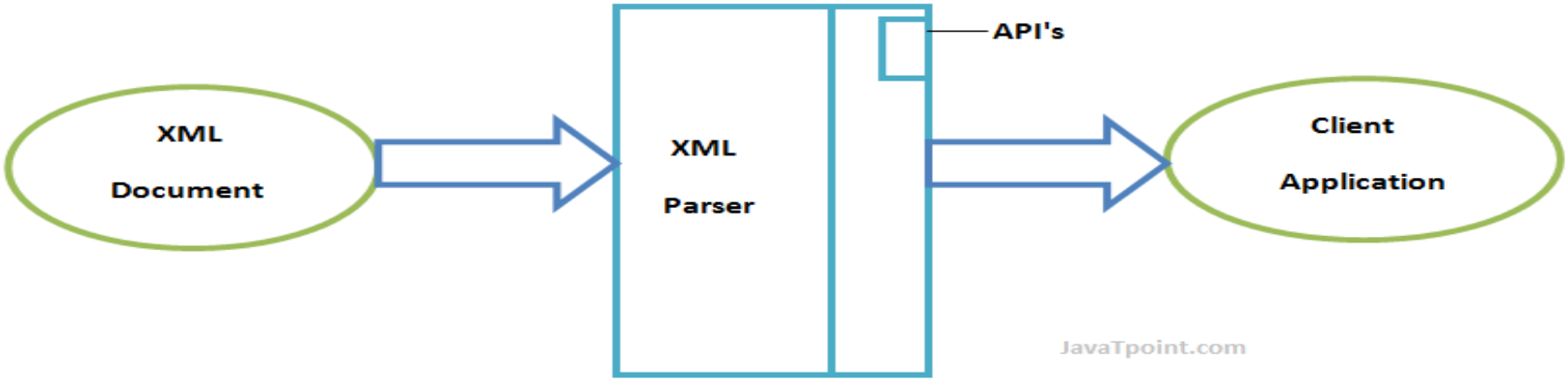
CDATA is text that will NOT be parsed by a parser. Tags inside the text will NOT be treated as markup and entities will not be expanded.

HTML vs XML

No.	HTML	XML
1)	HTML is used **to display data** and focuses on how data looks.	XML is a software and hardware independent tool used **to transport and store data**. It focuses on what data is.
2)	HTML is a **markup language** itself.	XML provides a **framework to define markup languages**.
3)	HTML is **not case sensitive**.	XML is **case sensitive**.
4)	HTML is a presentation language.	XML is neither a presentation language nor a programming language.
5)	HTML **has its own predefined tags**.	You **can define tags according to your need**.
6)	In HTML, it is **not necessary to use a closing tag**.	XML **makes it mandatory to use a closing tag**.
7)	HTML is **static** because it is used to display data.	XML is **dynamic** because it is used to transport data.
8)	HTML **does not preserve whitespaces**.	XML **preserve whitespaces**.

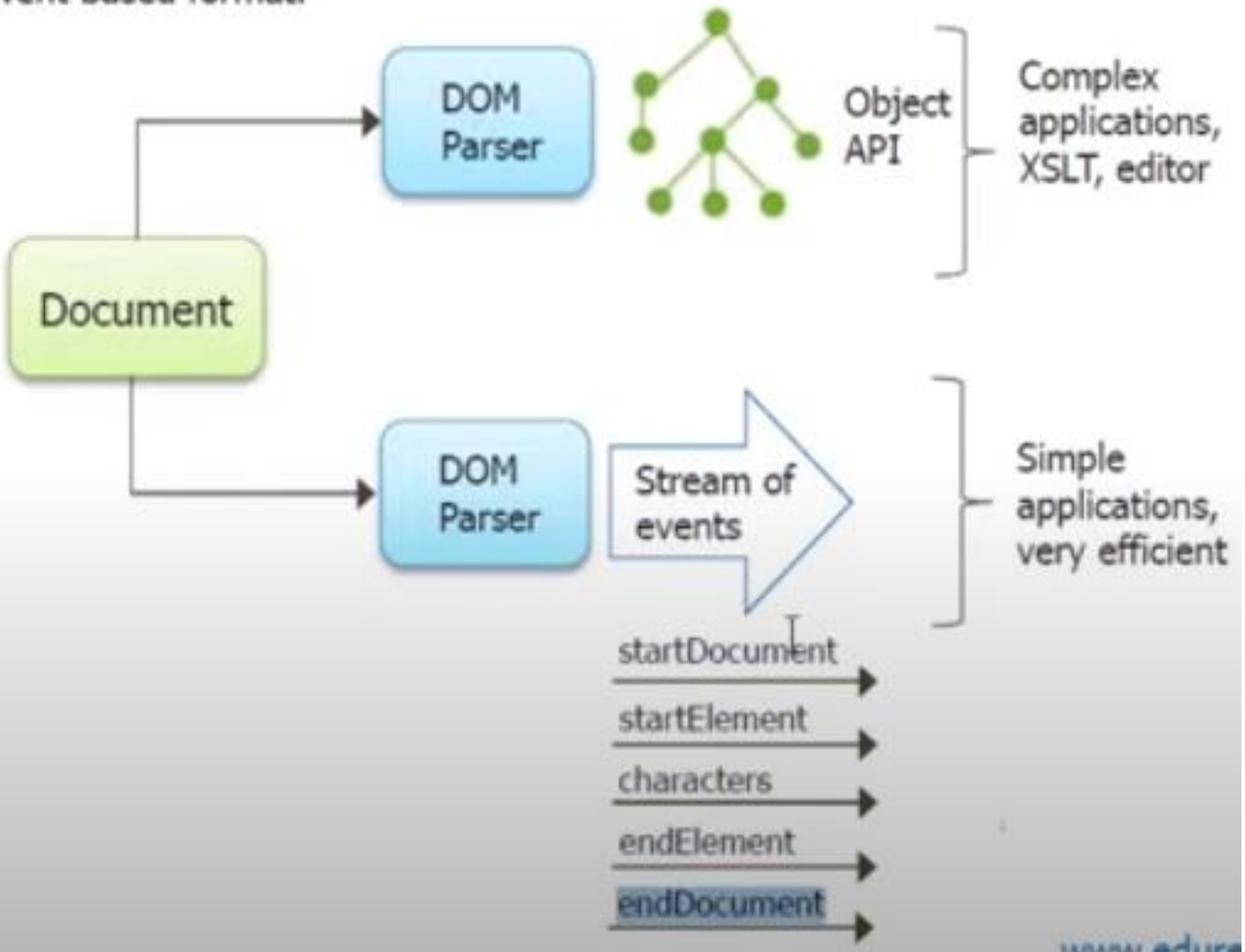
XML Parser (Reading XML file is called Parsing)

All major browsers have a built-in XML parser to access and manipulate XML.
XML Parser. An XML parser is a software library or package that provides interfaces for client applications to work with an XML document. The XML Parser is designed to read the XML and create a way for programs to use XML.
XML parser validates the document and check that the document is well formatted.
The XML DOM (Document Object Model) defines the properties and methods for accessing and editing XML.
However, before an XML document can be accessed, it must be loaded into an XML DOM object.
All modern browsers have a built-in XML parser that can convert text into an XML DOM object.
The working of XML parser by the figure given below:



→ DOM reads the XML file and stores in tree format.

→ SAX reads in event based format.



Types of XML Parsers

These are the two main types of XML Parsers:

1. DOM (Tree Based)
2. SAX (Event based)

DOM (Document Object Model)

A DOM document is an object which contains all the information of an XML document. It is composed like a tree structure. The DOM Parser implements a DOM API. This API(**application programming interface)** is very simple to use.

Features of DOM Parser

A DOM Parser creates an internal structure in memory which is a DOM document object and the client applications get information of the original XML document by invoking methods on this document object.

DOM Parser has a tree based structure.

Advantages

1) It supports both read and write operations and the API is very simple to use.

2) It is preferred when random access to widely separated parts of a document is required.

Disadvantages

1) It is memory inefficient. (consumes more memory because the whole XML document needs to loaded into memory).

2) It is comparatively slower than other parsers.

XML-DOM:- DOM is Document object Model.

↳ DOM document is a collection of nodes or pieces of info" organized in a hierarchy.

→ Tree-Based.

→ Defines a standard way to access and manipulate documents.

→ Programmer can modify / delete their content and can also create new elements.

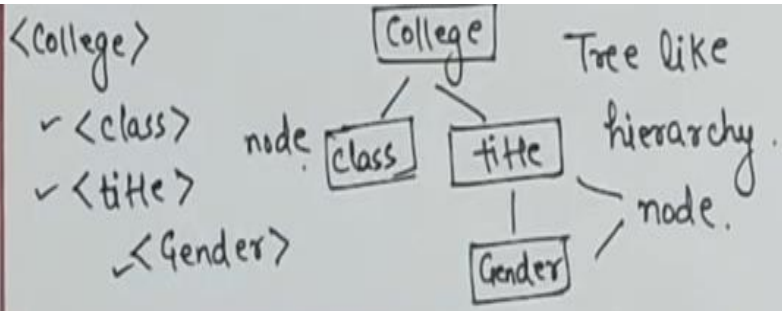
→ Elements, their content (text & Attributes) are all known as Nodes.

Example: `<h2 id="demo"></h2>`

HTML DOM.

```
<button type="button" onclick="
document.getElementById('demo').
innerHTML="Hello"> click
</button>
```

```
<College>
 ✓ <class>
 ✓ <title>
   ✓ <Gender>
</College>
```



Tree like hierarchy.

node.

node.

Example of XML DOM:-

```
//College.xml
<?xml version="1.0" encoding="UTF-8"?>
<College>
  <Branch Category="IT">
    <Students>60</Students>
    <HOD> ABC </HOD>
  </Branch>
  <Branch Category="ECE">
    <Students>100</Students>
    <HOD> XYZ </HOD>
  </Branch>
</College>
```

0 1

o/p:- 100

Get Value of XML element

```
t = xmlDoc.getElementsBy
    TagName("Students")
    [0].childNodes[0].
    nodeValue;
```

o/p = 60

```
[1].childNodes[1].
nodeValue;
```

SAX (Simple API for XML)

A SAX Parser implements SAX API. This API is an event based API and less intuitive.

Features of SAX Parser

It does not create any internal structure.

Clients does not know what methods to call, they just overrides the methods of the API and place his own code inside method.

It is an event based parser, it works like an event handler in Java.

Advantages

1) It is simple and memory efficient.

2) It is very fast and works for huge documents.

Disadvantages

1) It is event-based so its API is less intuitive.

2) Clients never know the full information because the data is broken into pieces.

DOM Parsers	SAX Parsers
DOM stands for Document Object Model	SAX stands for Simple API for XML
DOM reads the entire document	SAX reads node by node
DOM is useful when reading small to medium size XML files	SAX is used when big XML files needs to be parsed
DOM Is tree based parser	SAX is event based parser
DOM is little slow as compared to SAX	SAX is faster than DOM
DOM can insert and delete nodes	SAX cannot insert and delete nodes

Parsing a Text String

This example parses a text string into an XML DOM object, and extracts the info from it with JavaScript:

```
<!DOCTYPE html>
<html>
<body>
<p id="demo"></p>
<script>
var parser, xmlDoc;
var text = "<bookstore><book>" +
"<title>Web Technology</title>" +
"<author>Balram</author>" +
"<year>2005</year>" +
"</book></bookstore>";
parser = new DOMParser();
xmlDoc = parser.parseFromString(text,"text/xml");

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

Text string is defined

XML DOM parser is created

Parser create anew XML DOM object using the text string

XML DTD

What is DTD

DTD stands for **Document Type Definition**. It defines the legal building blocks of an XML document. It is used to define document structure with a list of legal elements and attributes.

Purpose of DTD

Its main purpose is to define the structure of an XML document. It contains a list of legal elements and define the structure with the help of them.

Checking Validation

Before proceeding with XML DTD, you must check the validation. An XML document is called "well-formed" if it contains the correct syntax.

A well-formed and valid XML document is one which have been validated against DTD.

Valid and well-formed XML document with DTD

Let's take an example of well-formed and valid XML document. It follows all the rules of DTD.

employee.xml

```
<?xml version="1.0"?>
<!DOCTYPE employee SYSTEM "employee.dtd">
<employee>
  <firstname>vimal</firstname>
  <lastname>jaiswal</lastname>
  <email>vimal@javatpoint.com</email>
</employee>
```

In the above example, the DOCTYPE declaration refers to an external DTD file. The content of the file is shown in below paragraph.

employee.dtd

```
<!ELEMENT employee (firstname,lastname,email)>
<!ELEMENT firstname (#PCDATA)>
<!ELEMENT lastname (#PCDATA)>
<!ELEMENT email (#PCDATA)>
```

Description of DTD

<!DOCTYPE employee : It defines that the root element of the document is employee.

<!ELEMENT employee: It defines that the employee element contains 3 elements "firstname, lastname and email".

<!ELEMENT firstname: It defines that the firstname element is #PCDATA typed. (parse-able data type).

<!ELEMENT lastname: It defines that the lastname element is #PCDATA typed. (parse-able data type).

<!ELEMENT email: It defines that the email element is #PCDATA typed. (parse-able data type).

XML DTD with entity declaration

A doctype declaration can also define special strings that can be used in the XML file.

An entity has three parts:

An ampersand (&)

An entity name

A semicolon (;)

Syntax to declare entity:

<!ENTITY entity-name "entity-value">

A code to define the ENTITY in doctype declaration.

author.xml

```
<?xml version="1.0" standalone="yes" ?>
<!DOCTYPE author [
  <!ELEMENT author (#PCDATA)>
  <!ENTITY sj "Sonoo Jaiswal">
]>
<author>&sj;</author>
```

In the above example, sj is an entity that is used inside the author element. In such case, it will print the value of sj entity that is "Sonoo Jaiswal".

XML CSS

Purpose of CSS in XML

CSS (Cascading Style Sheets) can be used to add style and display information to an XML document. It can format the whole XML document.

How to link XML file with CSS

To link XML files with CSS, you should use the following syntax:

<?xml-stylesheet type="text/css" href="cssemployee.css"**?>**

XML CSS Example:

cssemployee.css

```
employee
{
background-color: pink;
}
firstname,lastname,email
{
font-size:25px;
display:block;
color: blue;
margin-left: 50px;
}
```

create the DTD file.

employee.dtd

```
<!ELEMENT employee (firstname,lastname,email)>
<!ELEMENT firstname (#PCDATA)>
<!ELEMENT lastname (#PCDATA)>
<!ELEMENT email (#PCDATA)>
```

the xml file using CSS and DTD.

employee.xml

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/css" href="cssemployee.css"?>
<!DOCTYPE employee SYSTEM "employee.dtd">
<employee>
  <firstname>vimal</firstname>
  <lastname>jaiswal</lastname>
  <email>vimal@javatpoint.com</email>
</employee>
```