XML

XML stands for **E**xtensible **M**arkup **L**anguage. It is a text-based markup language derived from Standard Generalized Markup Language (SGML).

XML tags identify the data and are used to store and organize the data, rather than specifying how to display it like HTML tags, which are used to display the data. XML is not going to replace HTML in the near future, but it introduces new possibilities by adopting many successful features of HTML.

There are three important characteristics of XML that make it useful in a variety of systems and solutions −

- **XML is extensible** − XML allows you to create your own self-descriptive tags, or language, that suits your application.
- **XML carries the data, does not present it** − XML allows you to store the data irrespective of how it will be presented.
- **XML is a public standard** − XML was developed by an organization called the World Wide Web Consortium (W3C) and is available as an open standard.

XML Usage

A short list of XML usage says it all −

- XML can work behind the scene to simplify the creation of HTML documents for large web sites.
- XML can be used to exchange the information between organizations and systems.
- XML can be used for offloading and reloading of databases.
- XML can be used to store and arrange the data, which can customize your data handling needs.
- XML can easily be merged with style sheets to create almost any desired output.
- Virtually, any type of data can be expressed as an XML document.

What is Markup?

XML is a markup language that defines set of rules for encoding documents in a format that is both human-readable and machine-readable. So *what exactly is a markup language?* Markup is information added to a document that enhances its meaning in certain ways, in that it identifies the parts and how they relate to each other. More specifically, a markup language is a set of symbols that can be placed in the text of a document to demarcate and label the parts of that document.

Following example shows how XML markup looks, when embedded in a piece of text −

```
<message>
   <text>Hello, world!</text>
</message>
```

This snippet includes the markup symbols, or the tags such as <message>...</message> and <text>... </text>. The tags <message> and </message> mark the start and the end of the XML code fragment. The tags <text> and </text> surround the text Hello, world!.

Is XML a Programming Language?

A programming language consists of grammar rules and its own vocabulary which is used to create computer programs. These programs instruct the computer to perform specific tasks. XML does not qualify to be a programming language as it does not perform any computation or algorithms. It is usually stored in a simple text file and is processed by special software that is capable of interpreting XML.

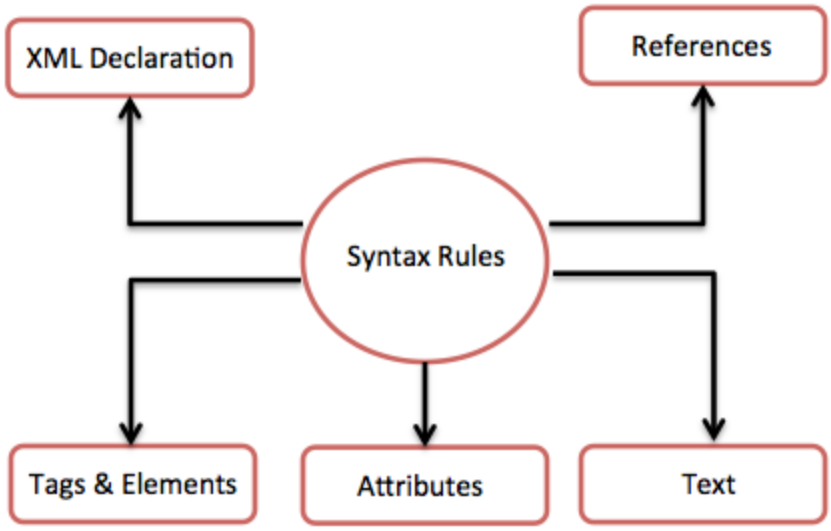
Following is a complete XML document −

```
<?xml version = "1.0"?>
<contact-info>
   <name>Tanmay Patil</name>
   <company>TutorialsPoint</company>
   <phone>(011) 123-4567</phone>
</contact-info>
```

You can notice there are two kinds of information in the above example −

- Markup, like <contact-info>
- The text, or the character data, *Tutorials Point* and *(040) 123-4567*.

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



Let us see each component of the above diagram in detail.

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.

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>

Syntax Rules for Tags and Elements

Element Syntax − Each XML-element needs to be closed either with start or with end elements as shown below −

<element>....</element>

or in simple-cases, just this way −

<element/>

Nesting of Elements − An XML-element can contain multiple XML-elements as its children, but the children elements must not overlap. i.e., an end tag of an element must have the same name as that of the most recent unmatched start tag.

The Following example shows incorrect nested tags −

```
<?xml version = "1.0"?>
<contact-info>
<company>TutorialsPoint
<contact-info>
</company>
```

The Following example shows correct nested tags −

```
<?xml version = "1.0"?>
<contact-info>
   <company>TutorialsPoint</company>
<contact-info>
```

Root Element − An XML document can have only one root element. For example, following is not a correct XML document, because both the **x** and **y** elements occur at the top level without a root element −

```
<x>...</x>
<y>...</y>
```

The Following example shows a correctly formed XML document −

```
<root>
   <x>...</x>
   <y>...</y>
</root>
```

Case Sensitivity − The names of XML-elements are case-sensitive. That means the name of the start and the end elements need to be exactly in the same case.

For example, **<contact-info>** is different from **<Contact-Info>**

XML Attributes

An **attribute** specifies a single property for the element, using a name/value pair. An XML-element can have one or more attributes. For example −

```
<a href = "http://www.tutorialspoint.com/">Tutorialspoint!</a>
```

Here **href** is the attribute name and **http://www.tutorialspoint.com/** is attribute value.

Syntax Rules for XML Attributes

- Attribute names in XML (unlike HTML) are case sensitive. That is, *HREF* and *href* are considered two different XML attributes.
- Same attribute cannot have two values in a syntax. The following example shows incorrect syntax because the attribute *b* is specified twice

```
<a b = "x" c = "y" b = "z">....</a>
```

- Attribute names are defined without quotation marks, whereas attribute values must always appear in quotation marks. Following example demonstrates incorrect xml syntax

```
<a b = x>....</a>
```

In the above syntax, the attribute value is not defined in quotation marks.

An XML *document* is a basic unit of XML information composed of elements and other markup in an orderly package. An XML *document* can contains wide variety of data. For example, database of numbers, numbers representing molecular structure or a mathematical equation.

XML Document Example

A simple document is shown in the following example −

```
<?xml version = "1.0"?>
<contact-info>
   <name>Tanmay Patil</name>
   <company>TutorialsPoint</company>
   <phone>(011) 123-4567</phone>
</contact-info>
```

The following image depicts the parts of XML document.

Document Prolog Section

Document Prolog comes at the top of the document, before the root element. This section contains −

- XML declaration
- Document type declaration

Document Elements Section

Document Elements are the building blocks of XML. These divide the document into a hierarchy of sections, each serving a specific purpose. You can separate a document into multiple sections so that they can be rendered differently, or used by a search engine. The elements can be containers, with a combination of text and other elements.

Syntax

Following syntax shows XML declaration −

```
<?xml
   version = "version_number"
   encoding = "encoding_declaration"
   standalone = "standalone_status"
?>
```

Each parameter consists of a parameter name, an equals sign (=), and parameter value inside a quote. Following table shows the above syntax in detail −

Parameter	**Parameter_value**	**Parameter_description**
Version	1.0	Specifies the version of the XML standard used.
Encoding	UTF-8, UTF-16, ISO-10646-UCS-2, ISO-10646-UCS-4, ISO-8859-1 to ISO-8859-9, ISO-2022-JP, Shift_JIS, EUC-JP	It defines the character encoding used in the document. UTF-8 is the default encoding used.
Standalone	yes or no	It informs the parser whether the document relies on the information from an external source, such as external document type definition (DTD), for its content. The default value is set to *no*. Setting it to *yes* tells the processor there are no external declarations required for parsing the document.

XML Declaration Examples

Following are few examples of XML declarations −

XML declaration with no parameters −

<?xml >

XML declaration with version definition −

<?xml version = "1.0">

XML declaration with all parameters defined −

<?xml version = "1.0" encoding = "UTF-8" standalone = "no" ?>

XML declaration with all parameters defined in single quotes −

<?xml version = '1.0' encoding = 'iso-8859-1' standalone = 'no' ?>

XML - Tags

Start Tag

The beginning of every non-empty XML element is marked by a start-tag. Following is an example of start-tag −

<address>

End Tag

Every element that has a start tag should end with an end-tag. Following is an example of end-tag −

</address>

Note, that the end tags include a solidus ("/") before the name of an element.

Empty Tag

The text that appears between start-tag and end-tag is called content. An element which has no content is termed as empty. An empty element can be represented in two ways as follows −

A start-tag immediately followed by an end-tag as shown below −

<hr></hr>

A complete empty-element tag is as shown below −

<hr />

Empty-element tags may be used for any element which has no content.

XML Tags Rules

Following are the rules that need to be followed to use XML tags −

Rule 1

XML tags are case-sensitive. Following line of code is an example of wrong syntax </Address>, because of the case difference in two tags, which is treated as erroneous syntax in XML.

<address>This is wrong syntax</Address>

Following code shows a correct way, where we use the same case to name the start and the end tag.

<address>This is correct syntax</address>

Rule 2

XML tags must be closed in an appropriate order, i.e., an XML tag opened inside another element must be closed before the outer element is closed. For example −

```
<outer_element>
  <internal_element>
    This tag is closed before the outer_element
  </internal_element>
</outer_element>
```

XML - Elements

XML elements can be defined as building blocks of an XML. Elements can behave as containers to hold text, elements, attributes, media objects or all of these.

Each XML document contains one or more elements, the scope of which are either delimited by start and end tags, or for empty elements, by an empty-element tag.

Syntax

Following is the syntax to write an XML element −

```
<element-name attribute1 attribute2>
....content
</element-name>
```

where,

- **element-name** is the name of the element. The *name* its case in the start and end tags must match.
- **attribute1, attribute2** are attributes of the element separated by white spaces. An attribute defines a property of the element. It associates a name with a value, which is a string of characters. An attribute is written as −

```
name = "value"
```

name is followed by an = sign and a string *value* inside double(" ") or single(' ') quotes.

Empty Element

An empty element (element with no content) has following syntax −

```
<name attribute1 attribute2.../>
```

Following is an example of an XML document using various XML element −

```
<?xml version = "1.0"?>
<contact-info>
   <address category = "residence">
      <name>Tanmay Patil</name>
      <company>TutorialsPoint</company>
      <phone>(011) 123-4567</phone>
   </address>
</contact-info>
```

XML Elements Rules

Following rules are required to be followed for XML elements −

- An element *name* can contain any alphanumeric characters. The only punctuation mark allowed in names are the hyphen (-), under-score (_) and period (.).
- Names are case sensitive. For example, Address, address, and ADDRESS are different names.
- Start and end tags of an element must be identical.
- An element, which is a container, can contain text or elements as seen in the above example.

XML - Attributes

Attributes are part of XML elements. An element can have multiple unique attributes. Attribute gives more information about XML elements. To be more precise, they define properties of elements. An XML attribute is always a name-value pair.

Syntax

An XML attribute has the following syntax −

<element-name attribute1 attribute2 >
....content..
< /element-name>

where *attribute1* and *attribute2* has the following form −

name = "value"

value has to be in double (" ") or single (' ') quotes. Here, *attribute1* and *attribute2* are unique attribute labels.

Attributes are used to add a unique label to an element, place the label in a category, add a Boolean flag, or otherwise associate it with some string of data. Following example demonstrates the use of attributes −

```
<?xml version = "1.0" encoding = "UTF-8"?>
<!DOCTYPE garden [
  <!ELEMENT garden (plants)*>
  <!ELEMENT plants (#PCDATA)>
```

```
  <!ATTLIST plants category CDATA #REQUIRED>
]>

<garden>
  <plants category = "flowers" />
  <plants category = "shrubs">
  </plants>
</garden>
```

Attributes are used to distinguish among elements of the same name, when you do not want to create a new element for every situation. Hence, the use of an attribute can add a little more detail in differentiating two or more similar elements.

In the above example, we have categorized the plants by including attribute category and assigning different values to each of the elements. Hence, we have two categories of *plants*, one *flowers* and other *color*. Thus, we have two plant elements with different attributes.

XML - Comments

Comments can be used to include related links, information, and terms. They are visible only in the source code; not in the XML code. Comments may appear anywhere in XML code.

Syntax

XML comment has the following syntax −

```
<!--Your comment-->
```

A comment starts with **<!--** and ends with **-->**. You can add textual notes as comments between the characters. You must not nest one comment inside the other.

Example

Following example demonstrates the use of comments in XML document −

```
<?xml version = "1.0" encoding = "UTF-8" ?>
<!--Students grades are uploaded by months-->
<class_list>
  <student>
    <name>Tanmay</name>
```

```
    <grade>A</grade>
  </student>
</class_list>
```

Any text between **<!--** and **-->** characters is considered as a comment.

XML Comments Rules

Following rules should be followed for XML comments −

- Comments cannot appear before XML declaration.
- Comments may appear anywhere in a document.
- Comments must not appear within attribute values.
- Comments cannot be nested inside the other comments.

XML - DTDs

The XML Document Type Declaration, commonly known as DTD, is a way to describe XML language precisely. DTDs check vocabulary and validity of the structure of XML documents against grammatical rules of appropriate XML language.

An XML DTD can be either specified inside the document, or it can be kept in a separate document and then liked separately.

Syntax

Basic syntax of a DTD is as follows −

```
<!DOCTYPE element DTD identifier
[
  declaration1
  declaration2
  ........
]>
```

In the above syntax,

- The **DTD** starts with <!DOCTYPE delimiter.
- An **element** tells the parser to parse the document from the specified root element.

- **DTD identifier** is an identifier for the document type definition, which may be the path to a file on the system or URL to a file on the internet. If the DTD is pointing to external path, it is called **External Subset.**
- **The square brackets [ ]** enclose an optional list of entity declarations called *Internal Subset*.

Internal DTD

A DTD is referred to as an internal DTD if elements are declared within the XML files. To refer it as internal DTD, *standalone* attribute in XML declaration must be set to **yes**. This means, the declaration works independent of an external source.

Syntax

Following is the syntax of internal DTD −

```
<!DOCTYPE root-element [element-declarations]>
```

where *root-element* is the name of root element and *element-declarations* is where you declare the elements.

Example

Following is a simple example of internal DTD −

```
<?xml version = "1.0" encoding = "UTF-8" standalone = "yes" ?>
<!DOCTYPE address [
   <!ELEMENT address (name,company,phone)>
   <!ELEMENT name (#PCDATA)>
   <!ELEMENT company (#PCDATA)>
   <!ELEMENT phone (#PCDATA)>
]>

<address>
   <name>Tanmay Patil</name>
   <company>TutorialsPoint</company>
   <phone>(011) 123-4567</phone>
</address>
```

Let us go through the above code −

Start Declaration − Begin the XML declaration with the following statement.

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

DTD − Immediately after the XML header, the *document type declaration* follows, commonly referred to as the DOCTYPE −

<!DOCTYPE address [

The DOCTYPE declaration has an exclamation mark (!) at the start of the element name. The DOCTYPE informs the parser that a DTD is associated with this XML document.

DTD Body − The DOCTYPE declaration is followed by body of the DTD, where you declare elements, attributes, entities, and notations.

<!ELEMENT address (name,company,phone)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT company (#PCDATA)>
<!ELEMENT phone_no (#PCDATA)>

Several elements are declared here that make up the vocabulary of the <name> document. <!ELEMENT name (#PCDATA)> defines the element *name* to be of type "#PCDATA". Here #PCDATA means parse-able text data.

End Declaration − Finally, the declaration section of the DTD is closed using a closing bracket and a closing angle bracket (**]>**). This effectively ends the definition, and thereafter, the XML document follows immediately.

Rules

- The document type declaration must appear at the start of the document (preceded only by the XML header) − it is not permitted anywhere else within the document.
- Similar to the DOCTYPE declaration, the element declarations must start with an exclamation mark.
- The Name in the document type declaration must match the element type of the root element.

External DTD

In external DTD elements are declared outside the XML file. They are accessed by specifying the system attributes which may be either the legal *.dtd* file or a valid URL. To refer it as external DTD, *standalone* attribute in the XML declaration must be set as **no**. This means, declaration includes information from the external source.

Syntax

Following is the syntax for external DTD −

```
<!DOCTYPE root-element SYSTEM "file-name">
```

where *file-name* is the file with *.dtd* extension.

Example

The following example shows external DTD usage −

```
<?xml version = "1.0" encoding = "UTF-8" standalone = "no" ?>
<!DOCTYPE address SYSTEM "address.dtd">
<address>
   <name>Tanmay Patil</name>
   <company>TutorialsPoint</company>
   <phone>(011) 123-4567</phone>
</address>
```

The content of the DTD file **address.dtd** is as shown −

```
<!ELEMENT address (name,company,phone)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT company (#PCDATA)>
<!ELEMENT phone (#PCDATA)>
```

Types

You can refer to an external DTD by using either **system identifiers** or **public identifiers**.

System Identifiers

A system identifier enables you to specify the location of an external file containing DTD declarations. Syntax is as follows −

```
<!DOCTYPE name SYSTEM "address.dtd" [...]>
```

As you can see, it contains keyword SYSTEM and a URI reference pointing to the location of the document.

Public Identifiers

Public identifiers provide a mechanism to locate DTD resources and is written as follows −

```
<!DOCTYPE name PUBLIC "-//Beginning XML//DTD Address Example//EN">
```

As you can see, it begins with keyword PUBLIC, followed by a specialized identifier. Public identifiers are used to identify an entry in a catalog. Public identifiers can follow any format, however, a commonly used format is called **Formal Public Identifiers, or FPIs**.

XML - Schemas

XML Schema is commonly known as **XML Schema Definition (XSD)**. It is used to describe and validate the structure and the content of XML data. XML schema defines the elements, attributes and data types. Schema element supports Namespaces. It is similar to a database schema that describes the data in a database.

Syntax

You need to declare a schema in your XML document as follows −

Example

The following example shows how to use schema −

```
<?xml version = "1.0" encoding = "UTF-8"?>
<xs:schema xmlns:xs = "http://www.w3.org/2001/XMLSchema">
  <xs:element name = "contact">
    <xs:complexType>
      <xs:sequence>
        <xs:element name = "name" type = "xs:string" />
        <xs:element name = "company" type = "xs:string" />
        <xs:element name = "phone" type = "xs:int" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

The basic idea behind XML Schemas is that they describe the legitimate format that an XML document can take.

Elements

As we saw in the XML - Elements chapter, elements are the building blocks of XML document. An element can be defined within an XSD as follows −

```
<xs:element name = "x" type = "y"/>
```

Definition Types

You can define XML schema elements in the following ways −

Simple Type

Simple type element is used only in the context of the text. Some of the predefined simple types are: xs:integer, xs:boolean, xs:string, xs:date. For example −

```
<xs:element name = "phone_number" type = "xs:int" />
```

Complex Type

A complex type is a container for other element definitions. This allows you to specify which child elements an element can contain and to provide some structure within your XML documents. For example −

```
<xs:element name = "Address">
  <xs:complexType>
    <xs:sequence>
      <xs:element name = "name" type = "xs:string" />
      <xs:element name = "company" type = "xs:string" />
      <xs:element name = "phone" type = "xs:int" />
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

In the above example, *Address* element consists of child elements. This is a container for other **<xs:element>** definitions, that allows to build a simple hierarchy of elements in the XML document.

Global Types

With the global type, you can define a single type in your document, which can be used by all other references. For example, suppose you want to generalize the *person* and

company for different addresses of the company. In such case, you can define a general type as follows −

```
<xs:element name = "AddressType">
  <xs:complexType>
    <xs:sequence>
      <xs:element name = "name" type = "xs:string" />
      <xs:element name = "company" type = "xs:string" />
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Now let us use this type in our example as follows −

```
<xs:element name = "Address1">
  <xs:complexType>
    <xs:sequence>
      <xs:element name = "address" type = "AddressType" />
      <xs:element name = "phone1" type = "xs:int" />
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name = "Address2">
  <xs:complexType>
    <xs:sequence>
      <xs:element name = "address" type = "AddressType" />
      <xs:element name = "phone2" type = "xs:int" />
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Instead of having to define the name and the company twice (once for *Address1* and once for *Address2*), we now have a single definition. This makes maintenance simpler, i.e., if you decide to add "Postcode" elements to the address, you need to add them at just one place.

Attributes

Attributes in XSD provide extra information within an element. Attributes have *name* and *type* property as shown below −

```
<xs:attribute name = "x" type = "y"/>
```

XML - Tree Structure

An XML document is always descriptive. The tree structure is often referred to as **XML Tree** and plays an important role to describe any XML document easily.

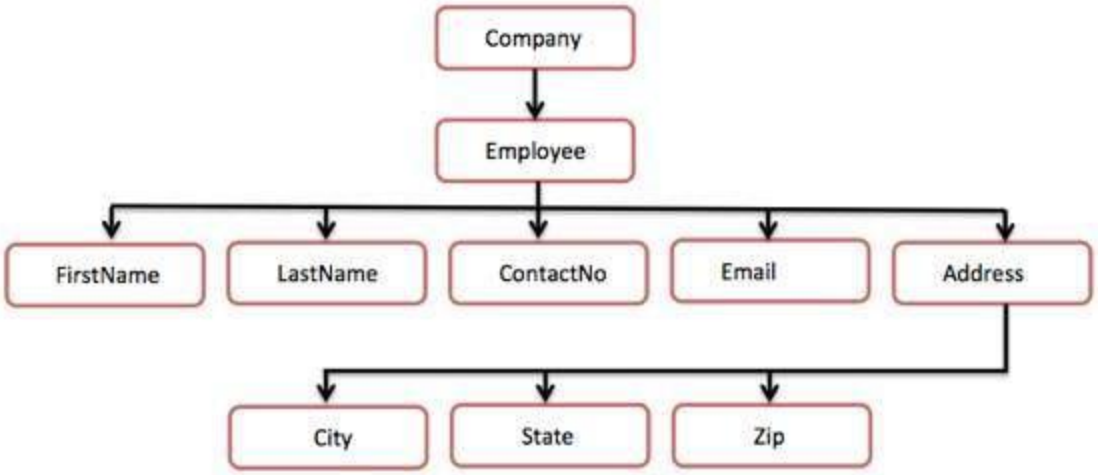
The tree structure contains root (parent) elements, child elements and so on. By using tree structure, you can get to know all succeeding branches and sub-branches starting from the root. The parsing starts at the root, then moves down the first branch to an element, take the first branch from there, and so on to the leaf nodes.

Example

Following example demonstrates simple XML tree structure −

```
<?xml version = "1.0"?>
<Company>
  <Employee>
    <FirstName>Tanmay</FirstName>
    <LastName>Patil</LastName>
    <ContactNo>1234567890</ContactNo>
    <Email>tanmaypatil@xyz.com</Email>
    <Address>
      <City>Bangalore</City>
      <State>Karnataka</State>
      <Zip>560212</Zip>
    </Address>
  </Employee>
</Company>
```

Following tree structure represents the above XML document −



In the above diagram, there is a root element named as <company>. Inside that, there is one more element <Employee>. Inside the employee element, there are five branches named <FirstName>, <LastName>, <ContactNo>, <Email>, and <Address>. Inside the <Address> element, there are three sub-branches, named <City> <State> and <Zip>.

XML - DOM

The **Document Object Model (DOM)** is the foundation of XML. XML documents have a hierarchy of informational units called *nodes*; DOM is a way of describing those nodes and the relationships between them.

A DOM document is a collection of nodes or pieces of information organized in a hierarchy. This hierarchy allows a developer to navigate through the tree looking for specific information. Because it is based on a hierarchy of information, the DOM is said to be *tree based*.

The XML DOM, on the other hand, also provides an API that allows a developer to add, edit, move, or remove nodes in the tree at any point in order to create an application.

Example

The following example (sample.htm) parses an XML document ("address.xml") into an XML DOM object and then extracts some information from it with JavaScript −

```
<!DOCTYPE html>
```

```
<html>
  <body>
    <h1>TutorialsPoint DOM example </h1>
    <div>
      <b>Name:</b> <span id = "name"></span><br>
      <b>Company:</b> <span id = "company"></span><br>
      <b>Phone:</b> <span id = "phone"></span>
    </div>
    <script>
      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","/xml/address.xml",false);
      xmlhttp.send();
      xmlDoc = xmlhttp.responseXML;

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

Contents of **address.xml** are as follows −

```
<?xml version = "1.0"?>
<contact-info>
  <name>Tanmay Patil</name>
  <company>TutorialsPoint</company>
  <phone>(011) 123-4567</phone>
</contact-info>
```

Now let us keep these two files **sample.htm** and **address.xml** in the same directory **/xml** and execute the **sample.htm** file by opening it in any browser. This should produce the following output.

TutorialsPoint DOM example

Name: Tanmay Patil
Company: TutorialsPoint
Phone: (011) 123-4567

Here, you can see how each of the child nodes is extracted to display their values.

XML - Namespaces

A **Namespace** is a set of unique names. Namespace is a mechanisms by which element and attribute name can be assigned to a group. The Namespace is identified by URI(Uniform Resource Identifiers).

Namespace Declaration

A Namespace is declared using reserved attributes. Such an attribute name must either be **xmlns** or begin with **xmlns:** shown as below −

```
<element xmlns:name = "URL">
```

Syntax

- The Namespace starts with the keyword **xmlns**.
- The word **name** is the Namespace prefix.
- The **URL** is the Namespace identifier.

Example

Namespace affects only a limited area in the document. An element containing the declaration and all of its descendants are in the scope of the Namespace. Following is a simple example of XML Namespace −

```
<?xml version = "1.0" encoding = "UTF-8"?>
<cont:contact xmlns:cont = "www.tutorialspoint.com/profile">
   <cont:name>Tanmay Patil</cont:name>
   <cont:company>TutorialsPoint</cont:company>
   <cont:phone>(011) 123-4567</cont:phone>
</cont:contact>
```

Here, the Namespace prefix is **cont**, and the Namespace identifier (URI) as *www.tutorialspoint.com/profile*. This means, the element names and attribute names with the **cont** prefix (including the contact element), all belong to the *www.tutorialspoint.com/profile* namespace.

XML - Databases

XML Database is used to store huge amount of information in the XML format. As the use of XML is increasing in every field, it is required to have a secured place to store the XML documents. The data stored in the database can be queried using **XQuery**, serialized, and exported into a desired format.

XML Database Types

There are two major types of XML databases −

- XML- enabled
- Native XML (NXD)

XML - Enabled Database

XML enabled database is nothing but the extension provided for the conversion of XML document. This is a relational database, where data is stored in tables consisting of rows and columns. The tables contain set of records, which in turn consist of fields.

Native XML Database

Native XML database is based on the container rather than table format. It can store large amount of XML document and data. Native XML database is queried by the **XPath**-expressions.

Native XML database has an advantage over the XML-enabled database. It is highly capable to store, query and maintain the XML document than XML-enabled database.

Example

Following example demonstrates XML database −

```
<?xml version = "1.0"?>
<contact-info>
   <contact1>
      <name>Tanmay Patil</name>
      <company>TutorialsPoint</company>
```

```
    <phone>(011) 123-4567</phone>
  </contact1>

  <contact2>
    <name>Manisha Patil</name>
    <company>TutorialsPoint</company>
    <phone>(011) 789-4567</phone>
  </contact2>
</contact-info>
```

Here, a table of contacts is created that holds the records of contacts (contact1 and contact2), which in turn consists of three entities − *name, company* and *phone*.