Introduction to XML

XML is a software- and hardware-independent tool for storing and transporting data.

What is XML?

- XML stands for eXtensible Markup Language
- XML is a markup language much like HTML
- XML was designed to store and transport data
- XML was designed to be self-descriptive
- XML is a W3C Recommendation on 10th Feb,1998
- XML is not a replacement for HTML.
- XML is designed to carry data, not to display data.

> *Note: Self-describing data is the data that describes both its content and structure.*

XML Does Not DO Anything

Maybe it is a little hard to understand, but XML does not DO anything.

This note is a note to Tove from Jani, stored as XML:

```
<note>
    <to>Ramu</to>
    <from>Anil</from>
    <heading>Reminder</heading>
    <body2>Don't forget me this weekend!</body2>
</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

But still, the XML above does not DO anything. XML is just information wrapped in tags.

Someone must write a piece of software to send, receive, store, or display it:

The Difference between XML and HTML

XML and HTML were designed with different goals:

- XML was designed to carry data - with focus on what data is
- HTML was designed to display data - with focus on how data looks
- XML tags are not predefined i.e. userdefined. HTML tags are predefined.

XML Does Not Use Predefined Tags

The XML language has no predefined 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.

HTML works with predefined tags like <p>, <h1>, <table>, etc.

With XML, the author must define both the tags and the document structure.

XML Simplifies Things

- XML simplifies data sharing
- XML simplifies data transport
- XML simplifies platform changes
- XML simplifies data availability

Many computer systems contain data in incompatible formats. Exchanging data between incompatible systems (or upgraded systems) is a time-consuming task for web developers. Large amounts of data must be converted, and incompatible data is often lost.

XML stores data in plain text format. This provides a software- and hardware-independent way of storing, transporting, and sharing data.

XML also makes it easier to expand or upgrade to new operating systems, new applications, or new browsers, without losing data.

With XML, data can be available to all kinds of "reading machines" like people, computers.

How Can XML be used?

XML is used in many aspects of web development.
XML is often used to separate data from presentation.

XML Separates Data from Presentation

XML does not carry any information about how to be displayed.
The same XML data can be used in many different presentation scenarios.
Because of this, with XML, there is a full separation between data and presentation.

XML is Often a Complement to HTML

In many HTML applications, XML is used to store or transport data, while HTML is used to format and display the same data.

XML Separates Data from HTML

When displaying data in HTML, you should not have to edit the HTML file when the data changes.
With XML, the data can be stored in separate XML files.

XML Tree Structure

XML documents are formed as **element trees**.
An XML tree starts at a **root element** and branches from the root to **child elements**.
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.

Parents have children. Children have parents. Siblings are children on the same level (brothers and sisters).

Self-Describing Syntax

XML uses a much self-describing syntax.

A prolog defines the XML version and the character encoding:

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

XML Syntax Rules

The syntax rules of XML are very simple and logical. The rules are easy to learn, and easy to use.

XML Documents Must Have a Root Element

XML documents must contain one **root** element that is the parent of all other elements:

```
<?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>
```

XML Tags are Case Sensitive

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 correct</message>

"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

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 Attribute Values Must Always be Quoted

XML elements can have attributes in name/value pairs just like in HTML.

In XML, the attribute values must always be quoted:

```
<note date="12/11/2007">
  <to>ramu</to>
  <from>raju</from>
</note>
```

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>salary < 1000</message>

To avoid this error, replace the "<" character with an **entity reference**:

<message>salary < 1000</message>

There are 5 pre-defined entity references in XML:

<	<	less than
>	>	greater than
&	&	ampersand
'	'	apostrophe
"	"	quotation mark

Only < and > are strictly illegal in XML, but it is a good habit to replace > with > as well.

Comments in XML

The syntax for writing comments in XML is similar to that of HTML:

<!-- This is a comment -->

Two dashes in the middle of a comment are not allowed:

<!-- This is an invalid -- comment -->

White-space is preserved in XML

XML does not truncate multiple white-spaces (HTML truncates multiple white-spaces to one single white-space):

XML:	Hello	Raju
HTML:	Hello Raju	

Well Formed XML

XML documents that conform to the syntax rules above are said to be "Well Formed" XML documents.

XML Elements

An XML document contains XML Elements.

What is an XML Element?

An XML element is everything from (including) the element's start tag to (including) the element's end tag.

<price>29.99</price>

An element can contain:

- text
- attributes
- other elements

- or a mix of the above

```
<bookstore>  //root element
    <book category="children">   //book is a child element of bookstore
       <title>Harry Potter</title> //title,author,year,price are sub elements of book tag
       <author>J K. Rowling</author>
        <year>2005</year>
       <price>29.99</price>
    </book>
   <book category="web">  //category is an attribute wrt book tag.. 2 parts … name and value
       <title>Learning XML</title>
       <author>Erik T. Ray</author>
        <year>2003</year>
       <price>39.95</price>
    </book>
</bookstore>
```

In the example above:

- <title>, <author>, <year>, and <price> have **text content** because they contain text (like 29.99).
- <bookstore> and <book> have **element contents**, because they contain elements.
- <book> has an **attribute** (category="children").

Empty XML Elements

An element with no content is said to be empty.

In XML, you can indicate an empty element like this:

<element></element>

You can also use a so called self-closing tag:

<element />

The two forms produce identical results in XML software (Readers, Parsers, Browsers).

Empty elements can have attributes.

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).

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)

Tip! Choose your naming style, and be consistent about it!

XML documents often have a corresponding database. A common practice is to use the naming rules of the database for the XML elements.

XML Attributes

XML elements can have attributes, just like HTML.

Attributes are designed to contain data related to a specific element.

XML Attributes Must be Quoted

Attribute values must always be quoted. Either single or double quotes can be used.

For a person's gender, the <person> element can be written like this:

<person gender="female">

or like this: <person gender='female'>

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

<gangster name='George "Shotgun" James'>

or you can use character entities:

<gangster name="George "Shotgun" James">

XML Elements vs Attributes

Take a look at these two examples:

```
<person gender="female">
    <firstname>Anna</firstname>
    <lastname>Smith</lastname>
</person>
```

```
<person>
    <gender>female</gender>
    <firstname>Anna</firstname>
    <lastname>Smith</lastname>
</person>
```

In the first example, gender is an attribute. In the last example, 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.

```
<note>
    <date>
      <year>2008</year>
       <month>01</month>
       <day>10</day>
    </date>
  <to>Ramu</to>
   <from>Anil</from>
   <heading>Reminder</heading>
  <body>Don't forget me this weekend</body>
</note>
```

Avoid XML Attributes?

Some things to consider when 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)

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.

XML Namespaces (XMLNS)

XML Namespaces provide a method to avoid element name conflicts.

Name Conflicts

In XML, element names are defined by the developer. This often results in a conflict when trying to mix XML documents from different XML applications.

This XML carries HTML table information:

```
<table>
  <tr>
    <td>Apples</td>
    <td>Bananas</td>
  </tr>
</table>
```

This XML carries information about a table (a piece of furniture):

```
<table>
  <name>African Coffee Table</name>
  <width>80</width>
  <length>120</length>
</table>
```

If these XML fragments were added together, there would be a name conflict. Both contain a <table> element, but the elements have different content and meaning.

A user or an XML application will not know how to handle these differences.

Solving the Name Conflict Using a Prefix

Name conflicts in XML can easily be avoided using a name prefix.

This XML carries information about an HTML table, and a piece of furniture:

```
<h:table>
  <h:tr>
    <h:td>Apples</h:td>
    <h:td>Bananas</h:td>
  </h:tr>
</h:table>

<f:table>
  <f:name>African Coffee Table</f:name>
  <f:width>80</f:width>
  <f:length>120</f:length>
</f:table>
```

In the example above, there will be no conflict because the two <table> elements have different names.

XML Namespaces - The xmlns Attribute

When using prefixes in XML, a **namespace** for the prefix must be defined.

The namespace can be defined by an **xmlns** attribute in the start tag of an element.

The namespace declaration has the following syntax. xmlns:*prefix*="*URI*".

```
<root>

<h:table xmlns:h="http://www.w3.org/TR/html4/">
  <h:tr>
    <h:td>Apples</h:td>
    <h:td>Bananas</h:td>
  </h:tr>
</h:table>

<f:table xmlns:f="https://www.w3schools.com/furniture">
  <f:name>African Coffee Table</f:name>
  <f:width>80</f:width>
  <f:length>120</f:length>
</f:table>

</root>
```

In the example above:

The xmlns attribute in the first <table> element gives the h: prefix a qualified namespace.

The xmlns attribute in the second <table> element gives the f: prefix a qualified namespace.

When a namespace is defined for an element, all child elements with the same prefix are associated with the same namespace.

Namespaces can also be declared in the XML root element:

```
<root xmlns:h="http://www.w3.org/TR/html4/"
xmlns:f="https://www.w3schools.com/furniture">

<h:table>
  <h:tr>
    <h:td>Apples</h:td>
    <h:td>Bananas</h:td>
  </h:tr>
</h:table>

<f:table>
  <f:name>African Coffee Table</f:name>
  <f:width>80</f:width>
  <f:length>120</f:length>
</f:table>

</root>
```

Note: The namespace URI is not used by the parser to look up information.

The purpose of using an URI is to give the namespace a unique name.

However, companies often use the namespace as a pointer to a web page containing namespace information.

Uniform Resource Identifier (URI)

A **Uniform Resource Identifier** (URI) is a string of characters which identifies an Internet Resource.

The most common URI is the **Uniform Resource Locator** (URL) which identifies an Internet domain address. Another, not so common type of URI is the **Uniform Resource Name** (URN).

Default Namespaces

Defining a default namespace for an element saves us from using prefixes in all the child elements. It has the following syntax:

xmlns="*namespaceURI*"

This XML carries HTML table information:

```
<table xmlns="http://www.w3.org/TR/html4/">
  <tr>
    <td>Apples</td>
    <td>Bananas</td>
  </tr>
</table>
```

This XML carries information about a piece of furniture:

```
<table xmlns="https://www.w3schools.com/furniture">
  <name>African Coffee Table</name>
  <width>80</width>
  <length>120</length>
</table>
```

XML Tree Structure

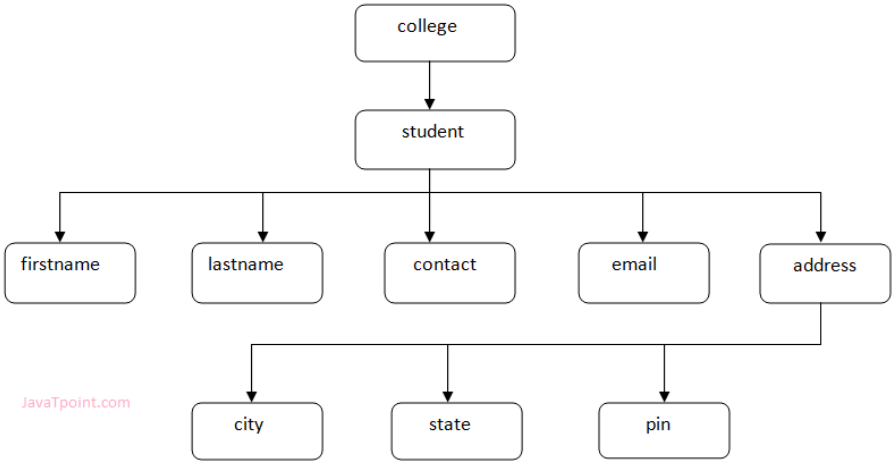
An XML document has a self descriptive structure. It forms a tree structure which is referred as an XML tree. The tree structure makes easy to describe an XML document.

A tree structure contains root element (as parent), child element and so on. It is very easy to traverse all succeeding branches and sub-branches and leaf nodes starting from the root.

Example of an XML document

```
<?xml version="1.0"?>
<college>
  <student>
    <firstname>Raj Kumar</firstname>
    <lastname>Dasari</lastname>
    <contact>9990449935</contact>
    <email> rajkumar.dasari @gmail.com        </email>
    <address>
      <city>Nizamabad</city>
      <state>Telangana</state>
      <pin>503000</pin>
    </address>
  </student>
</college>
```

Let's see the tree-structure representation of the above example.



In the above example, first line is the XML declaration. It defines the XML version 1.0. Next line shows the root element (college) of the document. Inside that there is one more element (student). Student element contains five branches named <firstname>, <lastname>, <contact>, <Email> and <address>. <address> branch contains 3 sub-branches named <city>, <state> and <pin>.

Note: DOM parser represents the XML document in Tree structure

XML Tree Rules

These rules are used to figure out the relationship of the elements. It shows if an element is a child or a parent of the other element.

Descendants: If element A is contained by element B, then A is known as descendant of B. In the above example "College" is the root element and all the other elements are the descendants of "College".

Ancestors: The containing element which contains other elements is called "Ancestor" of other element. In the above example Root element (College) is ancestor of all other elements.

XML Validation

A well formed XML document can be validated against DTD or Schema.

A well-formed XML document is an XML document with correct syntax. It is very necessary to know about valid XML document before knowing XML validation.

Valid XML document

It must be well formed (satisfy all the basic syntax condition)

It should be behave according to predefined DTD or XML schema

Rules for well formed XML

- It must begin with the XML declaration.
- It must have one unique root element.
- All start tags of XML documents must match end tags.
- XML tags are case sensitive.
- All elements must be closed.
- All elements must be properly nested.
- All attributes values must be quoted.

- XML entities must be used for special characters.

XML DTD(Document Type Definition)

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.

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.

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.

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>  //root element is address  and child elements are name company and phone
  <name>Raj Kumar</name>
  <company>TCS </company>
  <phone>(040) 123-4567</phone>
</address>
```

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

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

UTF stands for UCS Transformation Format, and UCS itself means Universal Character Set. The number 8 or 16 refers to the number of bits used to represent a character. They are either 8(1 to 4 bytes) or 16(2 or 4 bytes). For the documents without encoding information, UTF-8 is set by default. The abbreviation of "UTF-8" stands for **"8-Bit Universal Character Set Transformation Format."**

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 – externaldtd.xml

```
<?xml version = "1.0" encoding = "UTF-8" standalone = "no" ?>
<!DOCTYPE address SYSTEM "address.dtd">
<address>
  <name>Raj Kumar</name>
  <company>Capgemini</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)>
```

CDATA vs PCDATA

CDATA

CDATA: (Unparsed Character data): CDATA contains the text which is not parsed further in an XML document. Tags inside the CDATA text are not treated as markup and entities will not be expanded.

Let's take an example for CDATA:

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

In the above CDATA example, CDATA is used just after the element employee to make the data/text unparsed, so it will give the value of employee:

```
<firstname>vimal</firstname><lastname>jaiswal</lastname><email>vimal@javatpoint.com</email>
```

PCDATA

PCDATA: (Parsed Character Data): XML parsers are used to parse all the text in an XML document. PCDATA stands for Parsed Character data. PCDATA is the text that will be parsed by a parser. Tags inside the PCDATA will be treated as markup and entities will be expanded.

In other words you can say that a parsed character data means the XML parser examine the data and ensure that it doesn't content entity if it contains that will be replaced.

Let's take an example:

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

```
<!DOCTYPE employee SYSTEM "employee.dtd">
<employee>
  <firstname>vimal</firstname>
  <lastname>jaiswal</lastname>
  <email>vimal@gmail.com</email>
</employee>
```

In the above example, the employee element contains 3 more elements 'firstname', 'lastname', and 'email', so it parses further to get the data/text of firstname, lastname and email to give the value of employee as:

```
vimal jaiswal vimal@gmail.com
```

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.

DTD - Elements

In a DTD, elements are declared with an ELEMENT declaration.

Declaring Elements

In a DTD, XML elements are declared with the following syntax:

<!ELEMENT element-name category>
or
<!ELEMENT element-name (element-content)>

Ex: <!ELEMENT address (name,company,phone)>

Empty Elements

Empty elements are declared with the category keyword EMPTY:

<!ELEMENT element-name EMPTY>

DTD Example: <!ELEMENT br EMPTY>

XML example:

Elements with Parsed Character Data

Elements with only parsed character data are declared with #PCDATA inside parentheses:

<!ELEMENT element-name (#PCDATA)>

Example:

<!ELEMENT from (#PCDATA)>

Elements with any Contents

Elements declared with the category keyword ANY, can contain any combination of parsable data:

<!ELEMENT element-name ANY>

Example:

<!ELEMENT note ANY>

Elements with Children (sequences)

Elements with one or more children are declared with the name of the children elements inside parentheses:

<!ELEMENT element-name (child1)>
or
<!ELEMENT element-name (child1,child2,...)>

Example: <!ELEMENT note (to,from,heading,body)>

When children are declared in a sequence separated by commas, the children must appear in the same sequence in the document. In a full declaration, the children must also be declared, and the children can also have children. The full declaration of the "note" element is:

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

Declaring Only One Occurrence of an Element

<!ELEMENT element-name (child-name)>

Example: <!ELEMENT note (message)>

The example above declares that the child element "message" must occur once, and only once inside the "note" element.

Declaring Minimum One Occurrence of an Element

<!ELEMENT element-name (child-name+)>

Example:

<!ELEMENT note (message+)>

The + sign in the example above declares that the child element "message" must occur one or more times inside the "note" element.

Declaring Zero or More Occurrences of an Element

<!ELEMENT element-name (child-name*)>

Example:

<!ELEMENT note (message*)>

The * sign in the example above declares that the child element "message" can occur zero or more times inside the "note" element.

Declaring Zero or One Occurrences of an Element

<!ELEMENT element-name (child-name?)>

Example:

<!ELEMENT note (message?)>

The ? sign in the example above declares that the child element "message" can occur zero or one time inside the "note" element.

Declaring either/or Content

<!ELEMENT note (to,from,header,(message|body))>

The example above declares that the "note" element must contain a "to" element, a "from" element, a "header" element, and either a "message" or a "body" element.

Declaring Mixed Content

<!ELEMENT note (#PCDATA|to|from|header|message)*>

The example above declares that the "note" element can contain zero or more occurrences of parsed character data, "to", "from", "header", or "message" elements.

DTD - Attributes

In a DTD, attributes are declared with an ATTLIST declaration.

Declaring Attributes

An attribute declaration has the following syntax:

<!ATTLIST element-name attribute-name attribute-type attribute-value>

DTD example: <!ATTLIST payment type CDATA "check">

XML example: <payment type="check" />

The **attribute-type** can be one of the following:

Type	Description
CDATA	The value is character data
(*en1*|*en2*|..)	The value must be one from an enumerated list

ID	The value is a unique id
IDREF	The value is the id of another element
IDREFS	The value is a list of other ids
NMTOKEN	The value is a valid XML name
NMTOKENS	The value is a list of valid XML names
ENTITY	The value is an entity
ENTITIES	The value is a list of entities
NOTATION	The value is a name of a notation
xml:	The value is a predefined xml value

The **attribute-value** can be one of the following:

Value	**Explanation**
value	The default value of the attribute
#REQUIRED	The attribute is required
#IMPLIED	The attribute is optional
#FIXED *value*	The attribute value is fixed

A Default Attribute Value

DTD:
<!ELEMENT square EMPTY>
<!ATTLIST square width CDATA "0">

Valid XML:
<square width="100" />

In the example above, the "square" element is defined to be an empty element with a "width" attribute of type CDATA. If no width is specified, it has a default value of 0.

#REQUIRED

Syntax

<!ATTLIST element-name attribute-name attribute-type #REQUIRED>

Example

DTD:
<!ATTLIST person number CDATA #REQUIRED>

Valid XML:
<person number="5677" />

Invalid XML:
<person />

Use the #REQUIRED keyword if you don't have an option for a default value, but still want to force the attribute to be present.

#IMPLIED

Syntax

<!ATTLIST element-name attribute-name attribute-type #IMPLIED>

Example

DTD:
<!ATTLIST contact fax CDATA #IMPLIED>

Valid XML:
<contact fax="555-667788" />

Valid XML:
<contact />

Use the #IMPLIED keyword if you don't want to force the author to include an attribute, and you don't have an option for a default value.

#FIXED

Syntax

<!ATTLIST element-name attribute-name attribute-type #FIXED "value">

Example

DTD:
<!ATTLIST sender company CDATA #FIXED "Microsoft">

Valid XML:
<sender company="Microsoft" />

Invalid XML:
<sender company="W3Schools" />

Use the #FIXED keyword when you want an attribute to have a fixed value without allowing the author to change it. If an author includes another value, the XML parser will return an error.

XML Schema

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.

XML Schema Example1:

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

<xs:element name="note">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="to" type="xs:string"/>
      <xs:element name="from" type="xs:string"/>
      <xs:element name="heading" type="xs:string"/>
      <xs:element name="body" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

</xs:schema>
```

The purpose of an XML Schema is to define the legal building blocks of an XML document:

- the elements and attributes that can appear in a document
- the number of (and order of) child elements
- data types for elements and attributes
- default and fixed values for elements and attributes

Why Learn XML Schema?

In the XML world, hundreds of standardized XML formats are in daily use.

Many of these XML standards are defined by XML Schemas.

XML Schema is an XML-based (and more powerful) alternative to DTD

XML Schemas Support Data Types

One of the greatest strength of XML Schemas is the support for data types.

- It is easier to describe allowable document content
- It is easier to validate the correctness of data
- It is easier to define data facets (restrictions on data)
- It is easier to define data patterns (data formats)
- It is easier to convert data between different data types

XML Schemas use XML Syntax

Another great strength about XML Schemas is that they are written in XML.

- You don't have to learn a new language
- You can use your XML editor to edit your Schema files
- You can use your XML parser to parse your Schema files
- You can manipulate your Schema with the XML DOM
- You can transform your Schema with XSLT

XML Schemas are extensible, because they are written in XML.

With an extensible Schema definition you can:

- Reuse your Schema in other Schemas
- Create your own data types derived from the standard types
- Reference multiple schemas in the same document

A Simple XML Document

XML documents can have a reference to a DTD or to an XML Schema.

Look at this simple XML document called "note.xml":

```
<?xml version="1.0"?>
<note>
  <to>Ramu</to>
  <from>Rani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

The <schema> Element

The <schema> element is the root element of every XML Schema:

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

<xs:schema>
...
...
</xs:schema>
```

The <schema> element may contain some attributes. A schema declaration often looks something like this:

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

<xs:schema
xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="https://www.w3schools.com"
xmlns="https://www.w3schools.com"
elementFormDefault="qualified    >
...
...
</xs:schema>
```

The following fragment:

xmlns:xs="http://www.w3.org/2001/XMLSchema"

indicates that the elements and data types used in the schema come from the "http://www.w3.org/2001/XMLSchema" namespace. It also specifies that the elements and data types that come from the "http://www.w3.org/2001/XMLSchema" namespace should be prefixed with **xs:**

This fragment:

targetNamespace=https://www.w3schools.com

indicates that the elements defined by this schema (note, to, from, heading, body.) come from the "https://www.w3schools.com" namespace.

This fragment:

xmlns=https://www.w3schools.com

indicates that the default namespace is "https://www.w3schools.com".

This fragment:

elementFormDefault="qualified"

indicates that any elements used by the XML instance document which were declared in this schema must be namespace qualified.

Referencing a Schema in an XML Document

This XML document has a reference to an XML Schema:

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

<note xmlns="https://www.w3schools.com"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="https://www.w3schools.com/note.xsd">

  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>      //root element
```

The following fragment:

xmlns="https://www.w3schools.com"

specifies the default namespace declaration. This declaration tells the schema-validator that all the elements used in this XML document are declared in the "https://www.w3schools.com" namespace.

Once you have the XML Schema Instance namespace available:

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

you can use the schemaLocation attribute. This attribute has two values, separated by a space. The first value is the namespace to use. The second value is the location of the XML schema to use for that namespace:

xsi:schemaLocation="https://www.w3schools.com/note.xsd"

The xsi:schemaLocation attribute **locates schemas for elements and attributes that are in a specified namespace**. Its value is a namespace URI followed by a relative or absolute URL where the schema for that namespace can be found. It is most commonly attached to the root element but can appear further down the tree.

XSD Simple Elements

XML Schemas define the elements of your XML files.

A simple element is an XML element that contains only text.

It cannot contain any other elements or attributes.

What is a Simple Element?

A simple element is an XML element that can contain only text. It cannot contain any other elements or attributes.

However, the "only text" restriction is quite misleading. The text can be of many different types. It can be one of the types included in the XML Schema definition (boolean, string, date, etc.), or it can be a custom type that you can define yourself.

You can also add restrictions (facets) to a data type in order to limit its content, or you can require the data to match a specific pattern.

Defining a Simple Element

The syntax for defining a simple element is:

<xs:element name="abcd" type="xyz"/>

where abcd is the name of the element and xyz is the data type of the element.

Ex: <xs:element name="lastname" type="xs:string"/>

XML Schema has a lot of built-in data types. The most common types are:

- xs:string
- xs:decimal
- xs:integer
- xs:boolean
- xs:date
- xs:time

Example

Here are some XML elements: XML Data

```
<sname> Raju </sname>
<age>36</age>
<dob>1970-03-27</dob>
```

And here are the corresponding simple element definitions: XML Schema

```
<xs:element name="sname" type="xs:string"/>
<xs:element name="age" type="xs:integer"/>
<xs:element name="dob" type="xs:date"/>
```

Default and Fixed Values for Simple Elements

Simple elements may have a default value OR a fixed value specified.

A default value is automatically assigned to the element when no other value is specified.

In the following example the default value is "red":

<xs:element name="color" type="xs:string" default="red"/>

A fixed value is also automatically assigned to the element, and you cannot specify another value.

In the following example the fixed value is "red":

<xs:element name="color" type="xs:string" fixed="red"/>

XSD Attributes

All attributes are declared as simple types.

What is an Attribute?

Simple elements cannot have attributes. If an element has attributes, it is considered to be of a complex type. But the attribute itself is always declared as a simple type.

How to Define an Attribute?

The syntax for defining an attribute is:

<xs:attribute name="abcd" type="xyz"/>

where abcd is the name of the attribute and xyz specifies the data type of the attribute.

XML Schema has a lot of built-in data types. The most common types are:

- xs:string
- xs:decimal
- xs:integer
- xs:boolean
- xs:date
- xs:time

Example

Here is an XML element with an attribute:

<lastname lang="EN">Smith</lastname>

And here is the corresponding attribute definition:

<xs:attribute name="lang" type="xs:string"/>

Default and Fixed Values for Attributes

Attributes may have a default value OR a fixed value specified.

A default value is automatically assigned to the attribute when no other value is specified.

In the following example the default value is "EN":

<xs:attribute name="lang" type="xs:string" default="EN"/>

A fixed value is also automatically assigned to the attribute, and you cannot specify another value.

In the following example the fixed value is "EN":

<xs:attribute name="lang" type="xs:string" fixed="EN"/>

Optional and Required Attributes

Attributes are optional by default. To specify that the attribute is required, use the "use" attribute:

<xs:attribute name="lang" type="xs:string" use="required"/>

Restrictions on Content

When an XML element or attribute has a data type defined, it puts restrictions on the element's or attribute's content.

If an XML element is of type "xs:date" and contains a string like "Hello World", the element will not validate.

With XML Schemas, you can also add your own restrictions to your XML elements and attributes. These restrictions are called facets.

XSD Restrictions/Facets

Restrictions are used to define acceptable values for XML elements or attributes. Restrictions on XML elements are called facets.

Restrictions on Values

The following example defines an element called "age" with a restriction. The value of age cannot be lower than 0 or greater than 100:

```
<xs:element name="age">
  <xs:simpleType>
    <xs:restriction base="xs:integer">
      <xs:minInclusive value="0"/>
      <xs:maxInclusive value="100"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

Other Restrictions on a Series of Values

The example below defines an element called "letter" with a restriction.

The acceptable value is zero or more occurrences of lowercase letters from a to z:

```
<xs:element name="letter">
  <xs:simpleType>
    <xs:restriction base="xs:string">
```

```
      <xs:pattern value="([a-z])*"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

Restrictions for Datatypes

Constraint	Description
enumeration	Defines a list of acceptable values
fractionDigits	Specifies the maximum number of decimal places allowed. Must be equal to or greater than zero
length	Specifies the exact number of characters or list items allowed. Must be equal to or greater than zero
maxExclusive	Specifies the upper bounds for numeric values (the value must be less than this value)
maxInclusive	Specifies the upper bounds for numeric values (the value must be less than or equal to this value)
maxLength	Specifies the maximum number of characters or list items allowed. Must be equal to or greater than zero
minExclusive	Specifies the lower bounds for numeric values (the value must be greater than this value)
minInclusive	Specifies the lower bounds for numeric values (the value must be greater than or equal to this value)
minLength	Specifies the minimum number of characters or list items allowed. Must be equal to or greater than zero
pattern	Defines the exact sequence of characters that are acceptable
totalDigits	Specifies the exact number of digits allowed. Must be greater than zero
whiteSpace	Specifies how white space (line feeds, tabs, spaces, and carriage returns) is handled

XSD Complex Elements

A complex element is an XML element that contains other elements and/or attributes.

There are four kinds of complex elements:

- empty elements
- elements that contain only other elements
- elements that contain only text
- elements that contain both other elements and text

Note: Each of these elements may contain attributes as well!

Examples of Complex Elements

A complex XML element, "product", which is empty:

```
<product pid="1345"/>
```

1. A complex XML element, "employee", which contains only other elements:

```
<employee>
  <firstname>John</firstname>
  <lastname>Smith</lastname>
</employee>
```

2. A complex XML element, "food", which contains only text:

```
<food type="dessert">Ice cream</food>
```

3. A complex XML element, "description", which contains both elements and text:

```
<description>
It happened on <date lang="norwegian">03.03.99</date> ....
</description>
```

How to Define a Complex Element

Look at this complex XML element, "employee", which contains only other elements:

```
<employee>
  <firstname>John</firstname>
  <lastname>Smith</lastname>
</employee>
```

Define a complex element in an XML Schema:

The "employee" element can be declared directly by naming the element, like this:

```
<xs:element name="employee">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="firstname" type="xs:string"/>
      <xs:element name="lastname" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

XSD Indicators

We can control HOW elements are to be used in documents with indicators.

Order indicators:

- All
- Choice

- Sequence

Occurrence indicators:

- maxOccurs
- minOccurs

Order Indicators

Order indicators are used to define the order of the elements.

All Indicator

The <all> indicator specifies that the child elements can appear in any order, and that each child element must occur only once:

```
<xs:element name="person">
  <xs:complexType>
    <xs:all>
      <xs:element name="firstname" type="xs:string"/>
      <xs:element name="lastname" type="xs:string"/>
    </xs:all>
  </xs:complexType>
</xs:element>
```

Note: When using the <all> indicator you can set the <minOccurs> indicator to 0 or 1 and the <maxOccurs> indicator can only be set to 1

Choice Indicator

The <choice> indicator specifies that either one child element or another can occur:

```
<xs:element name="person">
  <xs:complexType>
    <xs:choice>
      <xs:element name="employee" type="employee"/>
      <xs:element name="member" type="member"/>
    </xs:choice>
  </xs:complexType>
</xs:element>
```

Sequence Indicator

The <sequence> indicator specifies that the child elements must appear in a specific order:

```
<xs:element name="person">
   <xs:complexType>
```

```
    <xs:sequence>
      <xs:element name="firstname" type="xs:string"/>
      <xs:element name="lastname" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Occurrence Indicators

Occurrence indicators are used to define how often an element can occur.

Note: For all "Order" and "Group" indicators (any, all, choice, sequence, group name, and group reference) the default value for maxOccurs and minOccurs is 1.

maxOccurs Indicator

The <maxOccurs> indicator specifies the maximum number of times an element can occur:

```
<xs:element name="person">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="full_name" type="xs:string"/>
      <xs:element name="child_name" type="xs:string" maxOccurs="10"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

The example above indicates that the "child_name" element can occur a minimum of one time (the default value for minOccurs is 1) and a maximum of ten times in the "person" element.

minOccurs Indicator

The <minOccurs> indicator specifies the minimum number of times an element can occur:

```
<xs:element name="person">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="full_name" type="xs:string"/>
      <xs:element name="child_name" type="xs:string" maxOccurs="10" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

The example above indicates that the "child_name" element can occur a minimum of zero times and a maximum of ten times in the "person" element.

Tip: To allow an element to appear an unlimited number of times, use the maxOccurs="unbounded" statement:

A working example:

An XML file called "Myfamily.xml":

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

<persons xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="family.xsd">

<person>
  <full_name>Hege Refsnes</full_name>
  <child_name>Cecilie</child_name>
</person>

<person>
  <full_name>Tove Refsnes</full_name>
  <child_name>Hege</child_name>
  <child_name>Stale</child_name>
  <child_name>Jim</child_name>
  <child_name>Borge</child_name>
</person>

<person>
  <full_name>Stale Refsnes</full_name>
</person>

</persons>
```

The XML file above contains a root element named "persons". Inside this root element we have defined three "person" elements. Each "person" element must contain a "full_name" element and it can contain up to five "child_name" elements.

Here is the schema file "family.xsd":

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified">

<xs:element name="persons">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="person" maxOccurs="unbounded">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="full_name" type="xs:string"/>
            <xs:element name="child_name" type="xs:string"
            minOccurs="0" maxOccurs="5"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
</xs:schema>
```

DTD vs XSD

There are many differences between DTD (Document Type Definition) and XSD (XML Schema Definition). In short, DTD provides less control on XML structure whereas XSD (XML schema) provides more control. The important differences are given below:

No.	DTD	XSD
1)	DTD stands for **Document Type Definition**.	XSD stands for XML Schema Definition.
2)	DTDs are derived from **SGML** syntax.	XSDs are written in XML.
3)	DTD **doesn't support datatypes**.	XSD **supports datatypes** for elements and attributes.
4)	DTD **doesn't support namespace**.	XSD **supports namespace**.
5)	DTD **doesn't define order** for child elements.	XSD **defines order** for child elements.
6)	DTD is **not extensible**.	XSD is **extensible**.
7)	DTD is **not simple to learn**.	XSD is **simple to learn** because you don't need to learn new language.
8)	DTD provides **less control** on XML structure.	XSD provides **more control** on XML structure.

What is XHTML

XHTML stands for **EXtensible HyperText Markup Language.** It is a cross between HTML and XML language.

XHTML is almost identical to HTML but it is stricter than HTML.

XHTML is HTML defined as an XML application. It is supported by all major browsers.

Although XHTML is almost the same as HTML but It is more important to create your code correctly, because XHTML is stricter than HTML in syntax and case sensitivity.

XHTML documents are well-formed and parsed using standard XML parsers, unlike HTML, which requires a lenient HTML-specific parser.

Why use XHTML

XHTML was developed to make HTML more extensible and increase interoperability with other data formats. There are two main reasons behind the creation of XHTML:

- It creates a stricter standard for making web pages, reducing incompatibilities between browsers. So it is compatible for all major browsers.
- It creates a standard that can be used on a variety of different devices without changes.

Let's take an example to understand it.

HTML is mainly used to create web pages but we can see that many pages on the internet contain "bad" HTML (not follow the HTML rule).

This HTML code works fine in most browsers (even if it does not follow the HTML rules).

For example:

```
<html>
<head>
  <title>This is an example of bad HTML</title>
<body>
  <h1>Bad HTML
  <p>This is a paragraph
</body>
```

The above HTML code doesn't follow the HTML rule although it runs.

Now a day, there are different browser technologies.

Some browsers run on computers, and some browsers run on mobile phones or other small devices.

The main issue with the bad HTML is that it can't be interpreted by smaller devices.

So, XHTML is introduced to combine the strengths of HTML and XML.

XHTML is HTML redesigned as XML. It helps you to create better formatted code on your site.

XHTML doesn't facilitate you to make badly formed code to be XHTML compatible. Unlike with HTML (where simple errors (like missing out a closing tag) are ignored by the browser), XHTML code must be exactly how it is specified to be.

S.No.	HTML	XHTML
1.	Hypertext mark-up language - - > HTML	Extensible Hypertext Mark-up Language - - > XHTML.
2.	Tim Berners created in 1991	World wide web consortium or W3C created in 2000
4.	It is an extension of standard generalized markup language or SGML	It is a combination of extensible markup language XML and hypertext markup language HTML
5.	It stored in a document file format	It stored as a markup language format
6.	It is not case sensitive as there is no mandatory rule to write the entire mark up in uppercase or	It is case-sensitive, and every tag and attribute used inside must be in lowercase.

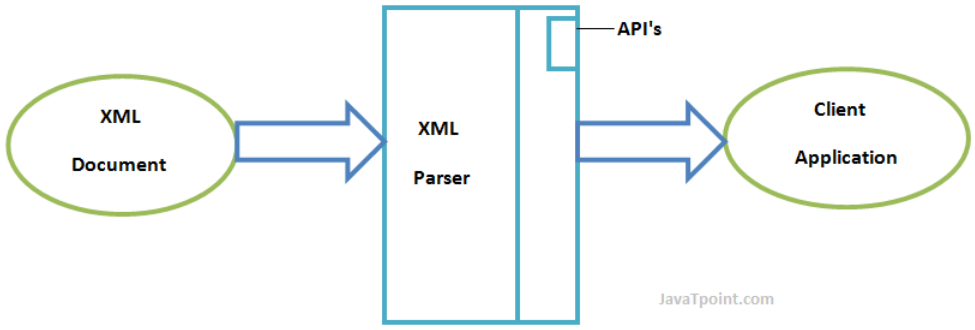
	lower case. It can also be a combination of both.	
7.	It is not mandatory to add document label < DOCTYPE >at the top of every page. We can even skip it.	It is mandatory to add a document label < DOCTYPE > at the beginning of the page.
8.	We can close any tag anytime and anywhere as per our needs	It is mandatory to close all the tags in strict residing order as they were declared.
9.	We can add attributes without any quotes.	It is mandatory to add quotes on every attribute we declare
10.	.html and .htm are the extensions used by HTML	.xhtml, .xml and .xht are the file extensions used by XHTML

XML Parsers

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.

Let's understand the working of XML parser by the figure given below:



Types of XML Parsers

These are the two main types of XML Parsers:

1. DOM
2. SAX

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 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.

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.

XML DOM

DOM is an acronym stands for Document Object Model. It defines a standard way to access and manipulate documents. The Document Object Model (DOM) is a programming API for

HTML and XML documents. It defines the logical structure of documents and the way a document is accessed and manipulated.

As a W3C specification, one important objective for the Document Object Model is to provide a standard programming interface that can be used in a wide variety of environments and applications. The Document Object Model can be used with any programming language.

XML DOM defines a standard way to access and manipulate XML documents.

What does XML DOM

The XML DOM makes a tree-structure view for an XML document.

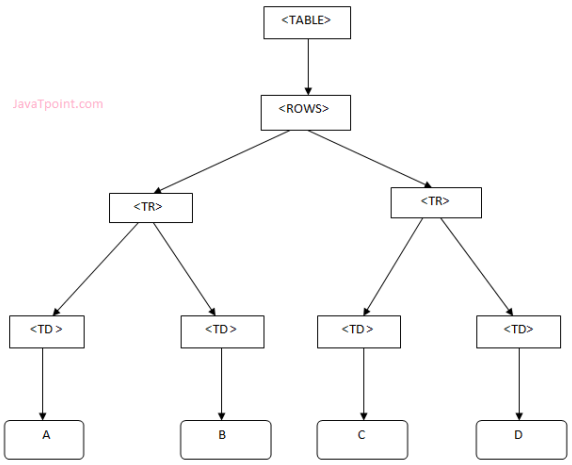
We can access all elements through the DOM tree.

We can modify or delete their content and also create new elements. The elements, their content (text and attributes) are all known as nodes.

For example, consider this table, taken from an HTML document:

```
<TABLE>
<ROWS>
<TR>
<TD>A</TD>
<TD>B</TD>
</TR>
<TR>
<TD>C</TD>
<TD>D</TD>
</TR>
</ROWS>
</TABLE>
```

The Document Object Model represents this table like this:



XML DOM Example : Load XML File

Let's take an example to show how an XML document ("note.xml") is parsed into an XML DOM object.

This example parses an XML document (note.xml) into an XML DOM object and extracts information from it with JavaScript.

Let's see the XML file that contains message.

note.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<note>
  <to>sonoojaiswal@javatpoint.com</to>
  <from>vimal@javatpoint.com</from>
  <body>Hello XML DOM</body>
</note>
```

Let's see the HTML file that extracts the data of XML document using DOM.

xmldom.html

```
<!DOCTYPE html>
<html>
<body>
<h1>Important Note</h1>
<div>
<b>To:</b> <span id="to"></span><br>
<b>From:</b> <span id="from"></span><br>
<b>Message:</b> <span id="message"></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","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>
```

Output:

Important Note

To: sonoojaiswal@javatpoint.com
From: vimal@javatpoint.com
Message: Hello XML DOM

XML DOM Example : Load XML String

This example parses an XML string into an XM DOM object and then extracts some information from it with a JavaScript.

Let's see the HTML file that extracts the data of XML string using DOM.

xmldom.html

```
<!DOCTYPE html>
<html>
<body>
<h1>Important Note2</h1>
<div>
<b>To:</b> <span id="to"></span><br>
<b>From:</b> <span id="from"></span><br>
<b>Message:</b> <span id="message"></span>
</div>
<script>
txt1="<note>";
txt2="<to>Sania Mirza</to>";
txt3="<from>Serena William</from>";
txt4="<body>Don't forget me this weekend!</body>";
txt5="</note>";
txt=txt1+txt2+txt3+txt4+txt5;

if (window.DOMParser)
  {
  parser=new DOMParser();
  xmlDoc=parser.parseFromString(txt,"text/xml");
  }
else // Internet Explorer
  {
  xmlDoc=new ActiveXObject("Microsoft.XMLDOM");
  xmlDoc.async=false;
  xmlDoc.loadXML(txt);
  }
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>
```

Output:

Important Note2

To: Sania Mirza
From: Serena William
Message: Don't forget me this weekend!

Difference of DOM and SAX

The objective of **DOM** (Document Object Model) parser and **SAX** (Simple API for XML) parser are same but implementation are different. Both the parser work in different way internally, but intent of both are same. Internal implementation of DOM Vs SAX are different. It means, with same intent philosophy of the implementation are different.

You should know the key differences about DOM and SAX Parser.

- **DOM** stands for Document Object Model while **SAX** stands for Simple API for XML parsing.
- DOM parser load full XML file in-memory and creates a tree representation of XML document, while SAX is an event based XML parser and doesn't load whole XML document into memory.
- If you know you have sufficient amount of memory in your server you can choose DOM as this is faster because load entire XML in-memory and works as tree structure which is faster to access.
- As a thumb rule, for small and medium sized XML documents, DOM is much faster than SAX because of in memory agnostic.
- As a thumb rule, for larger XML and for frequent parsing, SAX XML parser is better because it consume less memory.

DOM Vs SAX Parser

	DOM (Document Object Model)	SAX (Simple API for XML) Parser
Abbreviation	**DOM** stands for Document Object Model,	**SAX** stands for Simple API for XML Parsing
type	Load entire memory and keep in tree structure	event based parse
size of Document	good for smaller size	good to choose for larger size of file.
Load	Load entire document in memory	does not load entire document.
suitable	better suitable for smaller and efficient memory	SAX is suitable for larger XML do