tutorialspoint

SIMPLYEASYLEARNING

www.tutorialspoint.com

https://www.facebook.com/tutorialspointindia

https://twitter.com/tutorialspoint

About the Tutorial

The **D**ocument **O**bject **M**odel (DOM) is a W3C standard. It defines a standard for accessing documents like HTML and XML.

This tutorial will teach you the basics of XML DOM. The tutorial is divided into sections such as XML DOM Basics, XML DOM Operations and XML DOM Objects. Each of these sections contain related topics with simple and useful examples.

Audience

This reference has been prepared for the beginners to help them understand the basic-to-advanced concepts related to XML DOM. This tutorial will give you enough understanding on XML DOM from where you can take yourself to a higher level of expertise.

Prerequisites

Before proceeding with this tutorial you should have basic knowledge of XML, HTML and Javascript.

Disclaimer & Copyright

Table of Contents

1. XML DOM – Overview

The **D**ocument **O**bject **M**odel (DOM) is a W3C standard. It defines a standard for accessing documents like HTML and XML.

Definition of DOM as put by the W3C is:

> The Document Object Model (DOM) is an application programming interface (API) for HTML and XML documents. It defines the logical structure of documents and the way a document is accessed and manipulated.

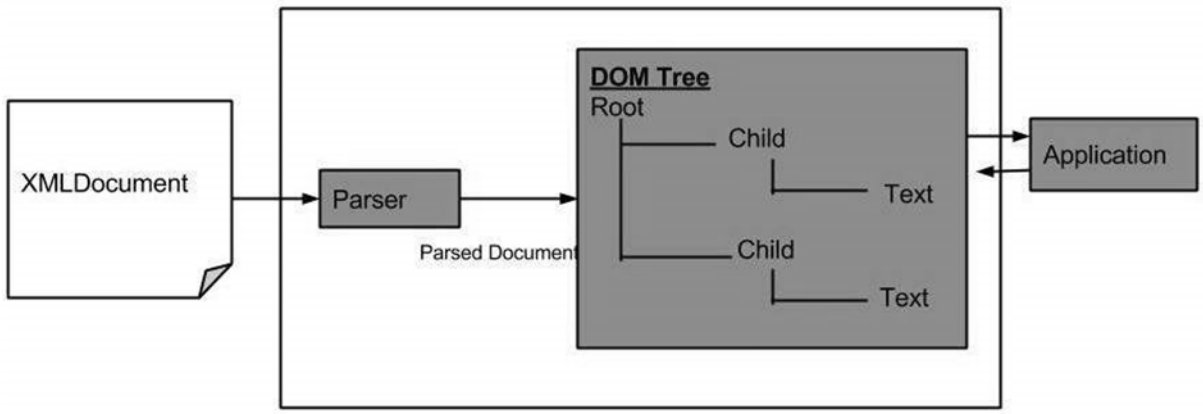
DOM defines the objects and properties and methods (interface) to access all XML elements. It is separated into 3 different parts / levels:

- **Core DOM** - standard model for any structured document
- **XML DOM** - standard model for XML documents
- **HTML DOM** - standard model for HTML documents

XML DOM is a standard object model for XML. XML documents have a hierarchy of informational units called *nodes*; DOM is a standard programming interface of describing those nodes and the relationships between them.

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

Following is the diagram for the DOM structure. The diagram depicts that parser evaluates an XML document as a DOM structure by traversing through each node.



Advantages of XML DOM

The following are the advantages of XML DOM.

- XML DOM is language and platform independent.
- XML DOM is **traversable** - Information in XML DOM is organized in a hierarchy which allows developer to navigate around the hierarchy looking for specific information.

- XML DOM is **modifiable** - It is dynamic in nature providing the developer a scope to add, edit, move or remove nodes at any point on the tree.

Disadvantages of XML DOM

- It consumes more memory (if the XML structure is large) as program written once remains in memory all the time until and unless removed explicitly.

- Due to the extensive usage of memory, its operational speed compared to SAX is slower.

2. XML DOM — Model

Now that we know what DOM means, let's see what a DOM structure is. A DOM document is a collection of *nodes* or pieces of information, organized in a hierarchy. Some types of *nodes* may have *child* nodes of various types and others are leaf nodes that cannot have anything under them in the document structure. Following is a list of the node types, with a list of node types that they may have as children:

- ***Document --*** Element (maximum of one), ProcessingInstruction, Comment, DocumentType (maximum of one)
- ***DocumentFragment --*** Element, ProcessingInstruction, Comment, Text, CDATASection, EntityReference
- ***EntityReference --*** Element, ProcessingInstruction, Comment, Text, CDATASection, EntityReference
- ***Element --*** Element, Text, Comment, ProcessingInstruction, CDATASection, EntityReference
- ***Attr --*** Text, EntityReference
- ***ProcessingInstruction --*** No children
- ***Comment --*** No children
- ***Text --*** No children
- ***CDATASection --*** No children
- ***Entity --*** Element, ProcessingInstruction, Comment, Text, CDATASection, EntityReference
- ***Notation --*** No children

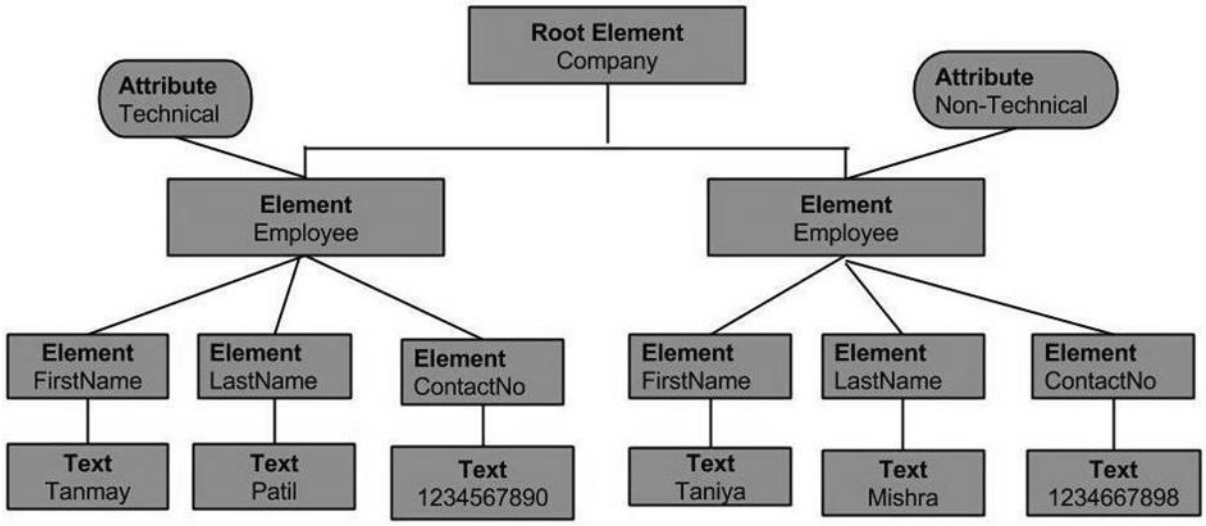
Example

Consider the DOM representation of the following XML document **node.xml**.

```
<?xml version="1.0"?>
<Company>
   <Employee category="technical">
      <FirstName>Tanmay</FirstName>
      <LastName>Patil</LastName>
      <ContactNo>1234567890</ContactNo>
   </Employee>
   <Employee category="non-technical">
      <FirstName>Taniya</FirstName>
      <LastName>Mishra</LastName>
```

```
      <ContactNo>1234667898</ContactNo>
   </Employee>
</Company>
```

The Document Object Model of the above XML document would be as follows:



From the above flowchart, we can infer:

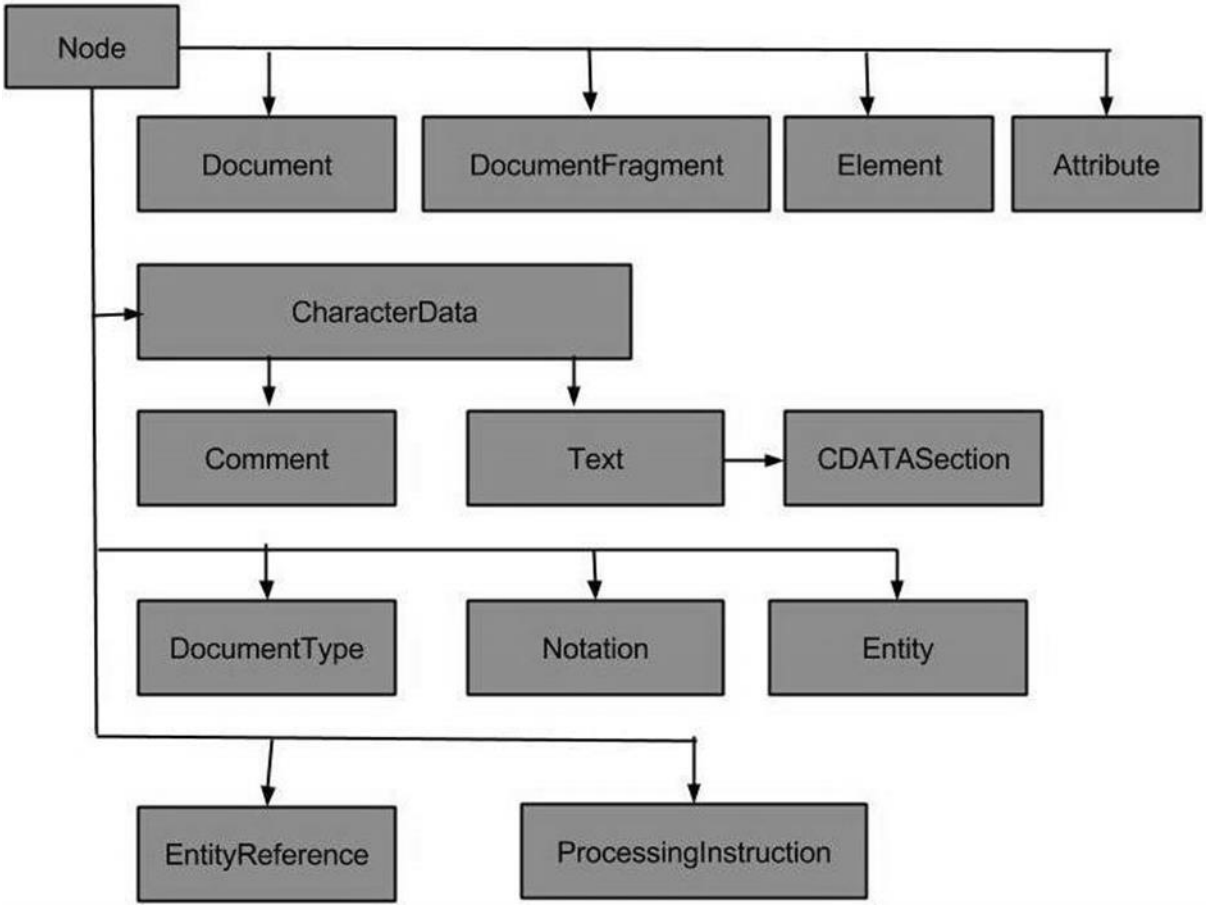
- *Node* object can have only one *parent node* object. This occupies the position above all the nodes. Here it is *Company*.
- The *parent node* can have multiple nodes called the *child* nodes. These *child* nodes can have additional nodes called the *attribute* nodes. In the above example, we have two attribute nodes — *Technical* and *Non-technical*. The *attribute* node is not actually a child of the element node, but is still associated with it.
- These *child* nodes in turn can have multiple child nodes. The text within the nodes is called the *text* node.
- The node objects at the same level are called as siblings.
- The DOM identifies:
    - the objects to represent the interface and manipulate the document.
    - the relationship among the objects and interfaces.

3. XML DOM — Nodes

In this chapter we will study about the XML DOM *Nodes*. Every XML DOM contains the information in hierarchical units called *Nodes* and the DOM describes these nodes and the relationship between them.

Node Types

The following flowchart shows all the node types:



The most common types of nodes in XML are:

- **Document Node:** Complete XML document structure is a *document node*.
- **Element Node:** Every XML element is an *element node*. This is also the only type of node that can have attributes.
- **Attribute Node:** Each attribute is considered an *attribute node*. It contains information about an element node, but is not actually considered to be children of the element.
- **Text Node:** The document texts are considered as *text node*. It can consist of more information or just white space.

Some less common types of nodes are:

- **CData Node:** This node contains information that should not be analyzed by the parser. Instead, it should just be passed on as plain text.
- **Comment Node:** This node includes information about the data, and is usually ignored by the application.
- **Processing Instructions Node:** This node contains information specifically aimed at the application.
- **Document Fragments Node**
- **Entities Node**
- **Entity reference nodes**
- **Notations Node**

4. XML DOM — Node Tree

In this chapter, we will study about the XML DOM *Node Tree.* In an XML document, the information is maintained in hierarchical structure; this hierarchical structure is referred to as the *Node Tree*. This hierarchy allows a developer to navigate around the tree looking for specific information, thus nodes are allowed to access. The content of these nodes can then be updated.

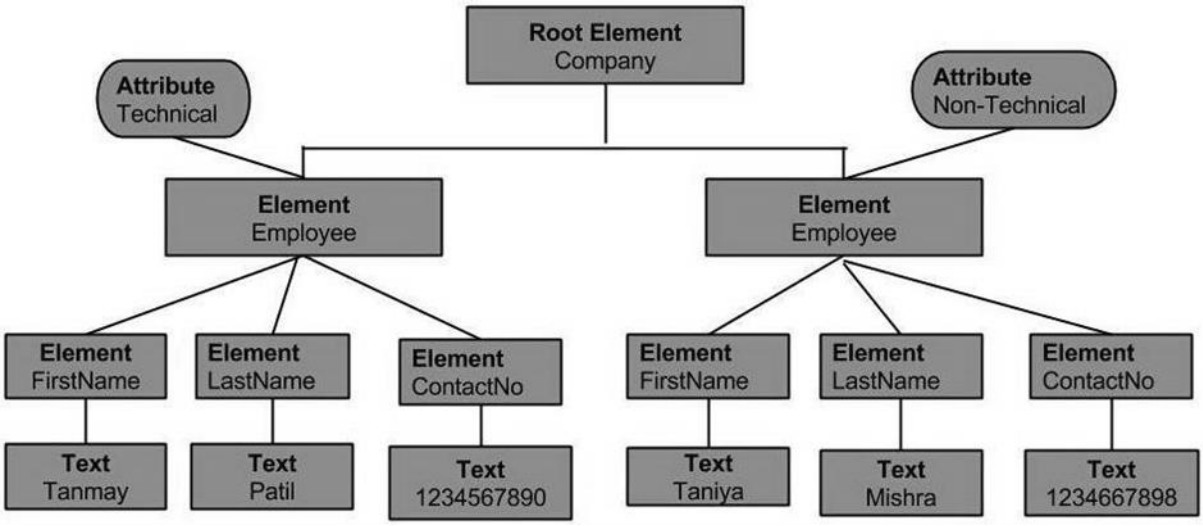
The structure of the node tree begins with the root element and spreads out to the child elements till the lowest level.

Example

Following example demonstrates a simple XML document, whose node tree is structure is shown in the diagram below:

```
<?xml version="1.0"?>
<Company>
   <Employee category="Technical">
      <FirstName>Tanmay</FirstName>
      <LastName>Patil</LastName>
      <ContactNo>1234567890</ContactNo>
   </Employee>
   <Employee category="Non-Technical">
      <FirstName>Taniya</FirstName>
      <LastName>Mishra</LastName>
      <ContactNo>1234667898</ContactNo>
   </Employee>
</Company>
```

As can be seen in the above example whose pictorial representation (of its DOM) is as shown below:



- The topmost node of a tree is called the **root**. The **root** node is <Company> which in turn contains the two nodes of <Employee>. These nodes are referred to as child nodes.
- The child node <Employee> of root node <Company>, in turn consists of its own child node (<FirstName>, <LastName>, <ContactNo>).
- The two child nodes, <Employee> have attribute values Technical and Non-Technical, are referred as *attribute nodes.*
- The text within every node is called the *text node*.

5. XML DOM — Methods

DOM as an API contains interfaces that represent different types of information that can be found in an XML document, such as elements and text. These interfaces include the methods and properties necessary to work with these objects. Properties define the characteristic of the node whereas methods give the way to manipulate the nodes.

Following table lists the DOM classes and interfaces:

Interface	Description
DOMImplementation	It provides a number of methods for performing operations that are independent of any particular instance of the document object model.
DocumentFragment	It is the "lightweight" or "minimal" document object, and it (as the superclass of Document) anchors the XML/HTML tree in a full-fledged document.
Document	It represents the XML document's top-level node, which provides access to all the nodes in the document, including the root element.
Node	It represents XML node.
NodeList	It represents a read-only list of *Node* objects.
NamedNodeMap	It represents collections of nodes that can be accessed by name.
Data	It extends *Node* with a set of attributes and methods for accessing character data in the DOM.
Attribute	It represents an attribute in an Element object.
Element	It represents the element node. Derives from Node.
Text	It represents the text node. Derives from CharacterData.
Comment	It represents the comment node. Derives from CharacterData.
ProcessingInstruction	It represents a "processing instruction". It is used in XML as a way to keep processor-specific information in the text of the document.
CDATA Section	It represents the CDATA Section. Derives from Text.

Entity	It represents an entity. Derives from Node.
EntityReference	This represent an entity reference in the tree. Derives from Node.

We will be discussing methods and properties of each of the above Interfaces in their respective chapters.

6. XML DOM — Loading

In this chapter, we will study about XML *Loading* and *Parsing*.

In order to describe the interfaces provided by the API, the W3C uses an abstract language called the Interface Definition Language (IDL). The advantage of using IDL is that the developer learns how to use the DOM with his or her favorite language and can switch easily to a different language.

The disadvantage is that, since it is abstract, the IDL cannot be used directly by Web developers. Due to the differences between programming languages, they need to have mapping — or binding — between the abstract interfaces and their concrete languages. DOM has been mapped to programming languages such as Javascript, JScript, Java, C, C++, PLSQL, Python, and Perl.

In the following sections and chapters, we will be using Javascript as our programming language to load XML file.

Parser

A *parser* is a software application that is designed to analyze a document, in our case XML document and do something specific with the information. Some of the DOM based parsers are listed in the following table:

Parser	Description
JAXP	Sun Microsystem's Java API for XML Parsing (JAXP)
XML4J	IBM's XML Parser for Java (XML4J)
msxml	Microsoft's XML parser (msxml) version 2.0 is built-into Internet Explorer 5.5
4DOM	4DOM is a parser for the Python programming language
XML::DOM	XML::DOM is a Perl module to manipulate XML documents using Perl
Xerces	Apache's Xerces Java Parser

In a tree-based API like DOM, the parser traverses the XML file and creates the corresponding DOM objects. Then you can traverse the DOM structure back and forth.

Loading and Parsing XML

While loading an XML document, the XML content can come in two forms:

- Directly as XML file
- As XML string

Content as XML file

Following example demonstrates how to load XML (node.xml) data using Ajax and Javascript when the XML content is received as an XML file. Here, the Ajax function gets the content of an xml file and stores it in XML DOM. Once the DOM object is created, it is then parsed.

```
<!DOCTYPE html>
<html>
   <body>
      <div>
         <b>FirstName:</b> <span id="FirstName"></span><br>
         <b>LastName:</b> <span id="LastName"></span><br>
         <b>ContactNo:</b> <span id="ContactNo"></span><br>
         <b>Email:</b> <span id="Email"></span>
      </div>
      <script>
     //if browser supports XMLHttpRequest
         if (window.XMLHttpRequest)
         {// Create an instance of XMLHttpRequest object. code for IE7+,
Firefox, Chrome, Opera, Safari
            xmlhttp = new XMLHttpRequest();
         }
         else
         {// code for IE6, IE5
            xmlhttp = new ActiveXObject("Microsoft.XMLHTTP");
         }
      // sets and sends the request for calling "node.xml"
         xmlhttp.open("GET","/dom/node.xml",false);
         xmlhttp.send();

      // sets and returns the content as XML DOM
         xmlDoc=xmlhttp.responseXML;

      //parsing the DOM object
```

```
         document.getElementById("FirstName").innerHTML=
         xmlDoc.getElementsByTagName("FirstName")[0].childNodes[0].nodeValue;
         document.getElementById("LastName").innerHTML=
         xmlDoc.getElementsByTagName("LastName")[0].childNodes[0].nodeValue;
         document.getElementById("ContactNo").innerHTML=

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

node.xml

```
<Company>
     <Employee category="Technical" id="firstelement">
        <FirstName>Tanmay</FirstName>
        <LastName>Patil</LastName>
        <ContactNo>1234567890</ContactNo>
        <Email>tanmaypatil@xyz.com</Email>
     </Employee>

     <Employee category="Non-Technical">
        <FirstName>Taniya</FirstName>
        <LastName>Mishra</LastName>
        <ContactNo>1234667898</ContactNo>
        <Email>taniyamishra@xyz.com</Email>
     </Employee>

     <Employee category="Management">
        <FirstName>Tanisha</FirstName>
        <LastName>Sharma</LastName>
        <ContactNo>1234562350</ContactNo>
        <Email>tanishasharma@xyz.com</Email>
     </Employee>
</Company>
```

Most of the details of the code are in the script code.

- Internet Explorer uses the *ActiveXObject("Microsoft.XMLHTTP")* to create an instance of XMLHttpRequest object, other browsers use the *XMLHttpRequest()* method.

- The *responseXML* transforms the XML content directly in XML DOM.

- Once the XML content is transformed into JavaScript XML DOM, you can access any XML element by using the JS DOM methods and properties. We have used the DOM properties such as *childNodes*, *nodeValue* and DOM methods such as getElementsById(ID), getElementsByTagName(tags_name).

Execution

Save this file as loadingexample.html and open it in your browser. You will receive the following output:

FirstName: Tanmay
LastName: Patil
ContactNo: 1234567890
Email: tanmaypatil@xyz.com

Content as XML string

Following example demonstrates how to load XML data using Ajax and Javascript when XML content is received as XML file. Here, the Ajax function gets the content of an xml file and stores it in XML DOM. Once the DOM object is created, it is then parsed.

```
<!DOCTYPE html>
<html>
   <head>
      <script>
     // loads the xml string in a dom object
         function loadXMLString(t)
         {
        // for non IE browsers
           if (window.DOMParser)
           {
            // create an instance for xml dom object
               parser=new DOMParser();
               xmlDoc=parser.parseFromString(t,"text/xml");
           }
        // code for IE
```

```
            else
            {
            // create an instance for xml dom object
               xmlDoc=new ActiveXObject("Microsoft.XMLDOM");
               xmlDoc.async=false;

               xmlDoc.loadXML(t);
            }
            return xmlDoc;
         }
      </script>
   </head>
   <body>
      <script>
     // a variable with the string
         var text="<Employee>";
         text=text+"<FirstName>Tanmay</FirstName>";
         text=text+"<LastName>Patil</LastName>";
         text=text+"<ContactNo>1234567890</ContactNo>";
         text=text+"<Email>tanmaypatil@xyz.com</Email>";
         text=text+"</Employee>";

     // calls the loadXMLString() with "text" function and store the xml dom in
a variable
         var xmlDoc=loadXMLString(text);

      //parsing the DOM object
         y=xmlDoc.documentElement.childNodes;
         for (i=0;i<y.length;i++)
         {

            document.write(y[i].childNodes[0].nodeValue);
            document.write("<br>");
         }
      </script>
   </body>
</html>
```

Most of the details of the code are in the script code.

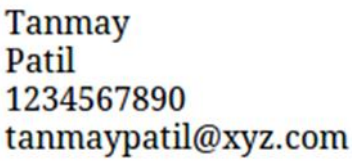
- Internet Explorer uses the *ActiveXObject("Microsoft.XMLDOM")* to load XML data into a DOM object, other browsers use the *DOMParser()* function and *parseFromString(text, 'text/xml')* method.

- The variable *text* shall contain a string with XML content.

- Once the XML content is transformed into JavaScript XML DOM, you can access any XML element by using JS DOM methods and properties. We have used DOM properties such as *childNodes*, *nodeValue*.

Execution

Save this file as loadingexample.html and open it in your browser. You will see the following output:



Now that we saw how the XML content transforms into JavaScript XML DOM, you can now access any XML element by using the XML DOM methods.

7. XML DOM — Traversing

In this chapter, we will discuss XML DOM Traversing. We studied in the previous chapter how to load XML document and parse the thus obtained DOM object. This parsed DOM object can be traversed. Traversing is a process in which looping is done in a systematic manner by going across each and every element step by step in a node tree.

Example

The following example (traverse_example.htm) demonstrates DOM traversing. Here we traverse through each child node of <Employee> element.

```
<!DOCTYPE html>
<html>
<style>
table,th,td
{
border:1px solid black;
border-collapse:collapse
}
</style>
   <body>
     <div id="ajax_xml">
      </div>
      <script>
     //if browser supports XMLHttpRequest
         if (window.XMLHttpRequest)
         {// Create an instance of XMLHttpRequest object. code for IE7+,
Firefox, Chrome, Opera, Safari
            var xmlhttp = new XMLHttpRequest();
         }
         else
         {// code for IE6, IE5
            var xmlhttp = new ActiveXObject("Microsoft.XMLHTTP");
         }

         // sets and sends the request for calling "node.xml"
         xmlhttp.open("GET","/dom/node.xml",false);
         xmlhttp.send();
```

```
      // sets and returns the content as XML DOM
         var xml_dom=xmlhttp.responseXML;
     // this variable stores the code of the html table
     var html_tab = '<table id="id_tabel" align="center"><tr><th>Employee
Category</th><th>FirstName</th><th>LastName</th><th>ContactNo</th><th>Email</th
></tr>';
     var arr_employees = xml_dom.getElementsByTagName("Employee");
     // traverses the "arr_employees" array
     for(var i=0; i<arr_employees.length; i++) {
     var employee_cat = arr_employees[i].getAttribute('category');
// gets the value of 'category' element of current "Element" tag

     // gets the value of first child-node of 'FirstName' element of current
"Employee" tag
     var employee_firstName =
arr_employees[i].getElementsByTagName('FirstName')[0].childNodes[0].nodeValue;

     // gets the value of first child-node of 'LastName' element of current
"Employee" tag
     var employee_lastName =
arr_employees[i].getElementsByTagName('LastName')[0].childNodes[0].nodeValue;

     // gets the value of first child-node of 'ContactNo' element of current
"Employee" tag
     var employee_contactno =
arr_employees[i].getElementsByTagName('ContactNo')[0].childNodes[0].nodeValue;

     // gets the value of first child-node of 'Email' element of current
"Employee" tag
     var employee_email =
arr_employees[i].getElementsByTagName('Email')[0].childNodes[0].nodeValue;

     // adds the values in the html table
     html_tab += '<tr><td>'+ employee_cat+ '</td><td>'+ employee_firstName+
'</td><td>'+ employee_lastName+ '</td><td>'+ employee_contactno+ '</td><td>'+
employee_email+ '</td></tr>';
     }
        html_tab += '</table>';
     // adds the html table in a html tag, with id="ajax_xml"
     document.getElementById('ajax_xml').innerHTML = html_tab;
      </script>

   </body>
```

```
</html>
```

- This code loads node.xml.
- The XML content is transformed into JavaScript XML DOM object.
- The array of elements (with tag Element) using the method getElementsByTagName() is obtained.
- Next, we traverse through this array and display the child node values in a table.

Execution

Save this file as *traverse_example.html* on the server path (this file and node.xml should be on the same path in your server). You will receive the following output:

Employee Category	FirstName	LastName	ContactNo	Email
Technical	Tanmay	Patil	1234567890	tanmaypatil@xyz.com
Non-Technical	Taniya	Mishra	1234667898	taniyamishra@xyz.com
Management	Tanisha	Sharma	1234562350	tanishasharma@xyz.com

End of ebook preview

If you liked what you saw…

Buy it from our store @ **https://store.tutorialspoint.com**