XML DOM - Quick Guide

Previous Page Next Page

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

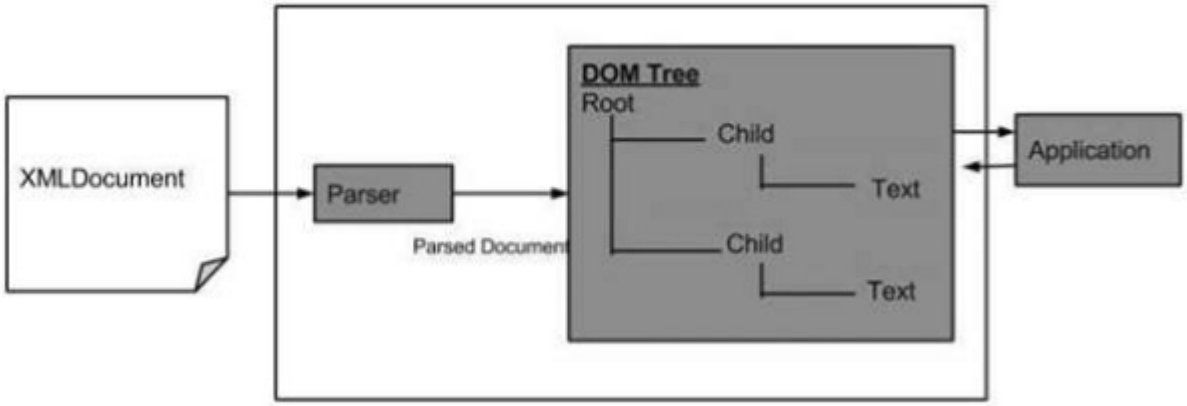
XML DOM – standard model for XML documents

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.

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

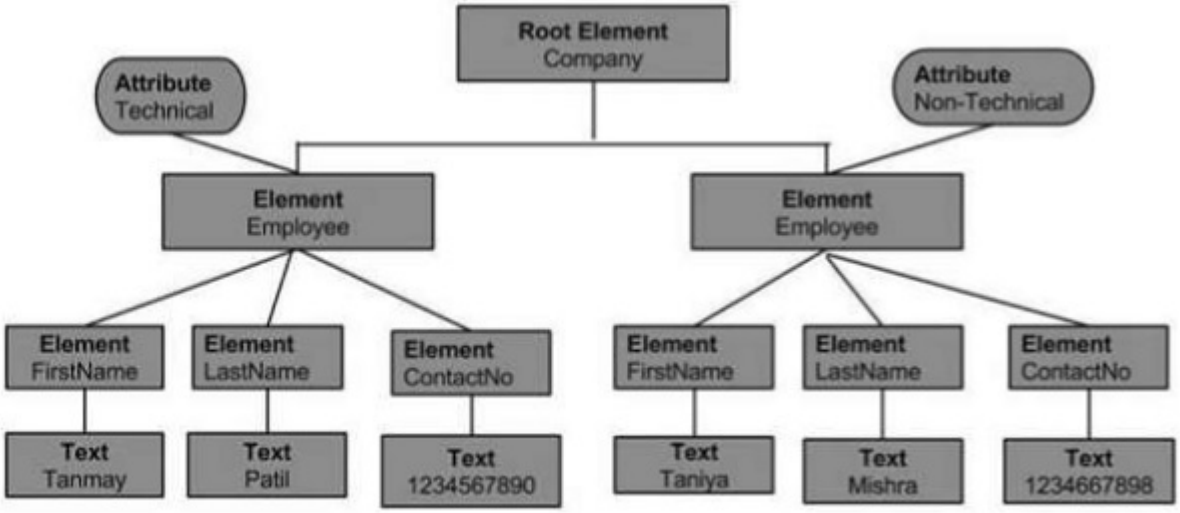
Example

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.

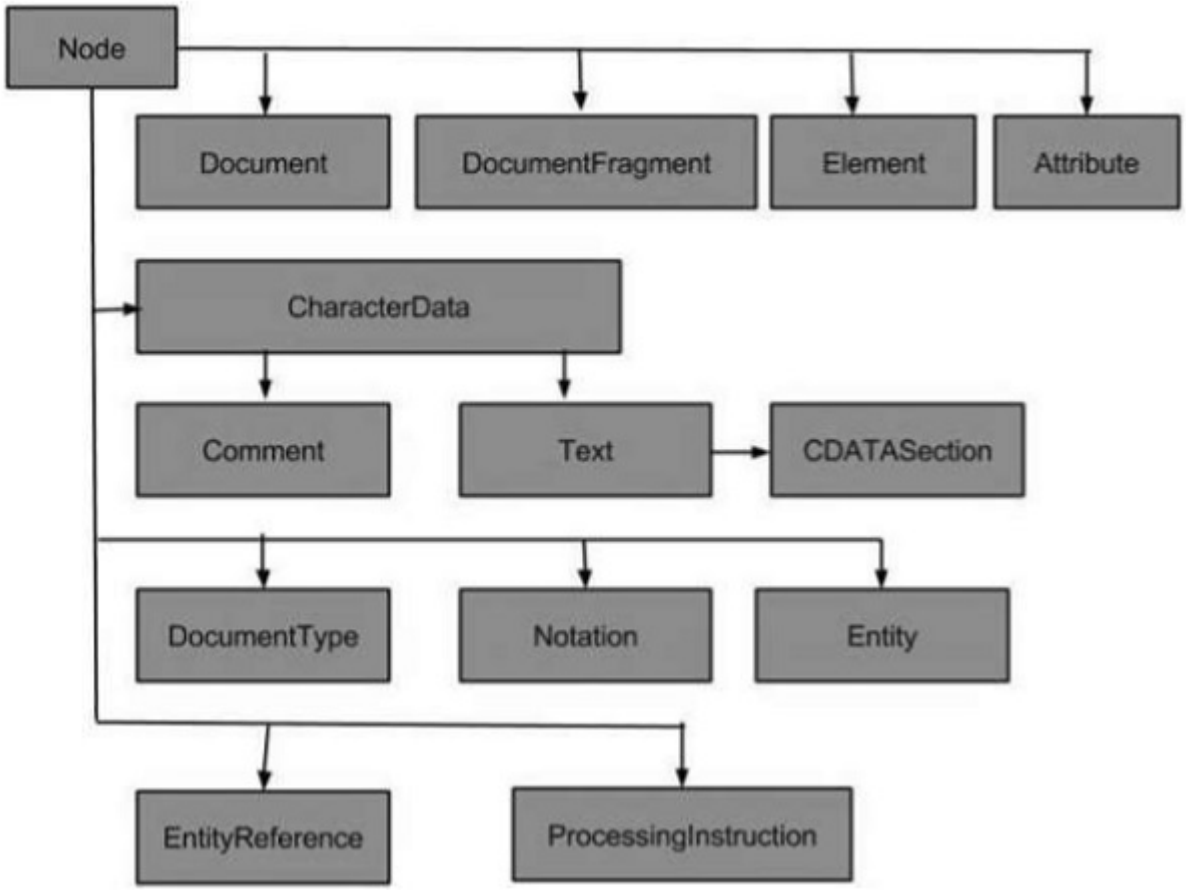
the relationship among the objects and interfaces.

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

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.

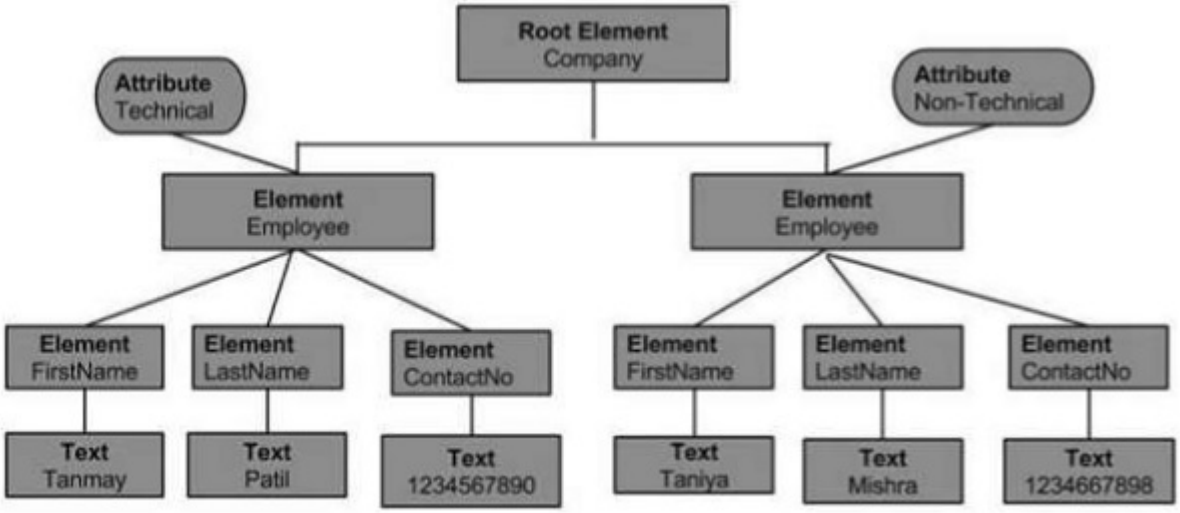
The structure of the node tree begins with the root element and spreads out to the child elements till the lowest level.

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*.

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 –

S.No.	Interface & Description
1	**DOMImplementation** It provides a number of methods for performing operations that are independent of any particular instance of the document object model.
2	**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.
3	

	Document It represents the XML document's top-level node, which provides access to all the nodes in the document, including the root element.
4	**Node** It represents XML node.
5	**NodeList** It represents a read-only list of *Node* objects.
6	**NamedNodeMap** It represents collections of nodes that can be accessed by name.
7	**Data** It extends *Node* with a set of attributes and methods for accessing character data in the DOM.
8	**Attribute** It represents an attribute in an Element object.
9	**Element** It represents the element node. Derives from Node.
10	**Text** It represents the text node. Derives from CharacterData.
11	**Comment** It represents the comment node. Derives from CharacterData.
12	**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.
13	**CDATA Section** It represents the CDATA Section. Derives from Text.

14	**Entity** It represents an entity. Derives from Node.
15	**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.

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 –

S.No	Parser & Description
1	**JAXP** Sun Microsystem’s Java API for XML Parsing (JAXP)
2	**XML4J** IBM’s XML Parser for Java (XML4J)

3	**msxml** Microsoft's XML parser (msxml) version 2.0 is built-into Internet Explorer 5.5
4	**4DOM** 4DOM is a parser for the Python programming language
5	**XML::DOM** XML::DOM is a Perl module to manipulate XML documents using Perl
6	**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 –

Tanmay
Patil
1234567890
tanmaypatil@xyz.com

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.

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].nodeValu

            // 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].nodeValu

            // 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

XML DOM - Navigation

Until now we studied DOM structure, how to load and parse XML DOM object and traverse through the DOM objects. Here we will see how we can navigate between nodes in a DOM object. The XML DOM consist of various properties of the nodes which help us navigate through the nodes, such as –

parentNode

childNodes

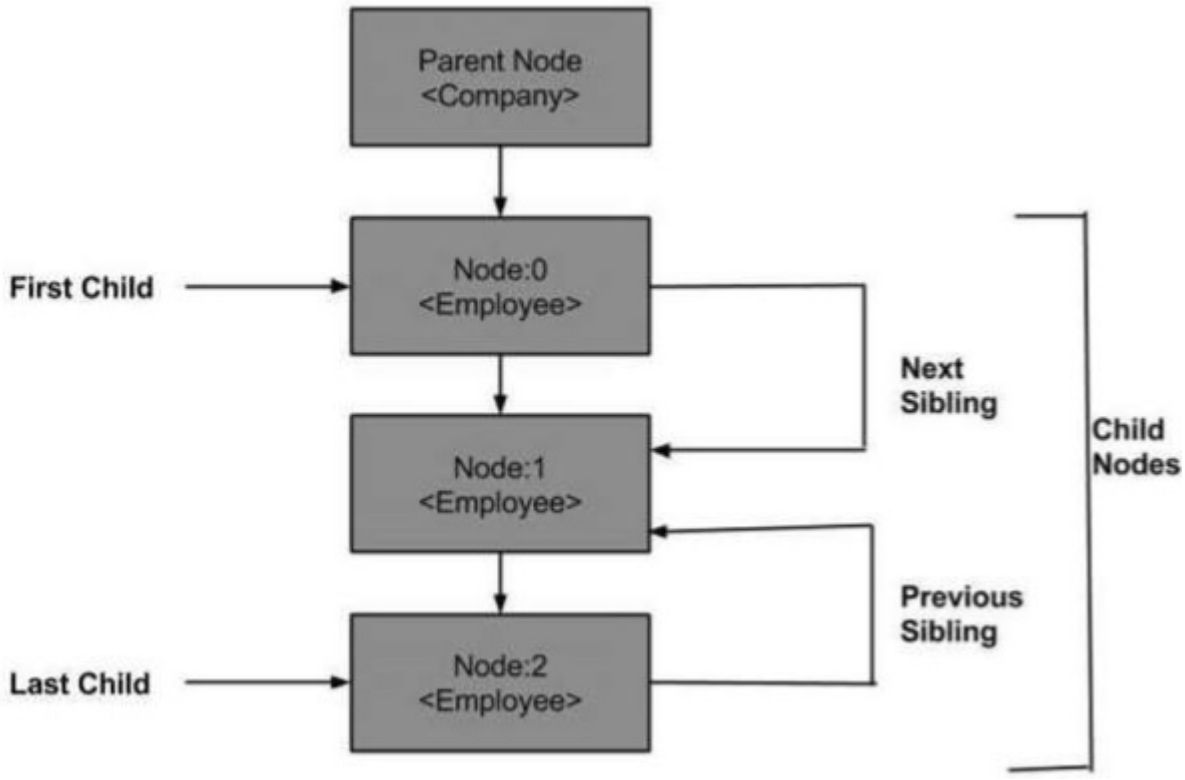
firstChild

lastChild

nextSibling

previousSibling

Following is a diagram of a node tree showing its relationship with the other nodes.



DOM - Parent Node

This property specifies the parent node as a node object.

Example

The following example (navigate_example.htm) parses an XML document (node.xml) into an XML DOM object. Then the DOM object is navigated to the parent node through the child node –

```
<!DOCTYPE html>
<html>
   <body>
      <script>
         if (window.XMLHttpRequest) {
            xmlhttp = new XMLHttpRequest();
         } else {
            xmlhttp = new ActiveXObject("Microsoft.XMLHTTP");
         }
         xmlhttp.open("GET","/dom/node.xml",false);
         xmlhttp.send();
         xmlDoc = xmlhttp.responseXML;

         var y = xmlDoc.getElementsByTagName("Employee")[0];
         document.write(y.parentNode.nodeName);
      </script>
   </body>
</html>
```

As you can see in the above example, the child node *Employee* navigates to its parent node.

Execution

Save this file as *navigate_example.html* on the server path (this file and node.xml should be on the same path in your server). In the output, we get the parent node of *Employee*, i.e, *Company*.

First Child

This property is of type *Node* and represents the first child name present in the NodeList.

Example

The following example (first_node_example.htm) parses an XML document (node.xml) into an XML DOM object, then navigates to the first child node present in the DOM object.

```
<!DOCTYPE html>
<html>
   <body>
      <script>
         if (window.XMLHttpRequest) {
            xmlhttp = new XMLHttpRequest();
         } else {
            xmlhttp = new ActiveXObject("Microsoft.XMLHTTP");
         }
         xmlhttp.open("GET","/dom/node.xml",false);
         xmlhttp.send();
         xmlDoc = xmlhttp.responseXML;

         function get_firstChild(p) {
            a = p.firstChild;

            while (a.nodeType != 1) {
               a = a.nextSibling;
            }
            return a;
         }
         var firstchild = get_firstChild(xmlDoc.getElementsByTagName("Employee")[0]);
         document.write(firstchild.nodeName);
      </script>
   </body>
</html>
```

Function *get_firstChild(p)* is used to avoid the empty nodes. It helps to get the firstChild element from the node list.

x = get_firstChild(xmlDoc.getElementsByTagName("Employee")[0]) fetches the first child node for the tag name *Employee*.

Execution

Save this file as *first_node_example.htm* on the server path (this file and node.xml should be on the same path in your server). In the output, we get the first child node of *Employee* i.e. *FirstName*.

Last Child

This property is of type *Node* and represents the last child name present in the NodeList.

Example

The following example (last_node_example.htm) parses an XML document (node.xml) into an XML DOM object, then navigates to the last child node present in the xml DOM object.

```
<!DOCTYPE html>
  <body>
      <script>
         if (window.XMLHttpRequest) {
            xmlhttp = new XMLHttpRequest();
         } else {
            xmlhttp = new ActiveXObject("Microsoft.XMLHTTP");
         }
         xmlhttp.open("GET","/dom/node.xml",false);
         xmlhttp.send();
         xmlDoc = xmlhttp.responseXML;

         function get_lastChild(p) {
            a = p.lastChild;

            while (a.nodeType != 1){
               a = a.previousSibling;
            }
            return a;
         }
         var lastchild = get_lastChild(xmlDoc.getElementsByTagName("Employee")[0]);
         document.write(lastchild.nodeName);
      </script>
   </body>
</html>
```

Execution

Save this file as *last_node_example.htm* on the server path (this file and node.xml should be on the same path in your server). In the output, we get the last child node of *Employee,* i.e, *Email*.

Next Sibling

This property is of type *Node* and represents the next child, i.e, the next sibling of the specified child element present in the NodeList.

Example

The following example (nextSibling_example.htm) parses an XML document (node.xml) into an XML DOM object which navigates immediately to the next node present in the xml document.

```
<!DOCTYPE html>
   <body>
      <script>
         if (window.XMLHttpRequest) {
            xmlhttp = new XMLHttpRequest();
         }
         else {
            xmlhttp = new ActiveXObject("Microsoft.XMLHTTP");
         }
         xmlhttp.open("GET","/dom/node.xml",false);
         xmlhttp.send();
         xmlDoc = xmlhttp.responseXML;

         function get_nextSibling(p) {
            a = p.nextSibling;

            while (a.nodeType != 1) {
               a = a.nextSibling;
            }
            return a;
         }
         var nextsibling = get_nextSibling(xmlDoc.getElementsByTagName("FirstName")[0]);
         document.write(nextsibling.nodeName);
      </script>
   </body>
</html>
```

Execution

Save this file as *nextSibling_example.htm* on the server path (this file and node.xml should be on the same path in your server). In the output, we get the next sibling node of *FirstName,* i.e, *LastName*.

Previous Sibling

This property is of type *Node* and represents the previous child, i.e, the previous sibling of the specified child element present in the NodeList.

Example

The following example (previoussibling_example.htm) parses an XML document (node.xml) into an XML DOM object, then navigates the before node of the last child node present in the xml document.

```
<!DOCTYPE html>
   <body>
      <script>
         if (window.XMLHttpRequest)
         {
            xmlhttp = new XMLHttpRequest();
```

```
         } else {
            xmlhttp = new ActiveXObject("Microsoft.XMLHTTP");
         }
         xmlhttp.open("GET","/dom/node.xml",false);
         xmlhttp.send();
         xmlDoc = xmlhttp.responseXML;

         function get_previousSibling(p) {
            a = p.previousSibling;

            while (a.nodeType != 1) {
               a = a.previousSibling;
            }
            return a;
         }

         prevsibling = get_previousSibling(xmlDoc.getElementsByTagName("Email")[0]);
         document.write(prevsibling.nodeName);
      </script>
   </body>
</html>
```

Execution

Save this file as *previoussibling_example.htm* on the server path (this file and node.xml should be on the same path in your server). In the output, we get the previous sibling node of *Email,* i.e, *ContactNo*.

XML DOM - Accessing

In this chapter, we will study about how to access the XML DOM nodes which are considered as the informational units of the XML document. The node structure of the XML DOM allows the developer to navigate around the tree looking for specific information and simultaneously access the information.

Accessing Nodes

Following are the three ways in which you can access the nodes –

- By using the **getElementsByTagName ()** method
- By looping through or traversing through nodes tree
- By navigating the node tree, using the node relationships

getElementsByTagName ()

This method allows accessing the information of a node by specifying the node name. It also allows accessing the information of the Node List and Node List Length.

Syntax

The getElementByTagName() method has the following syntax –

```
node.getElementByTagName("tagname");
```

Where,

node – is the document node.

tagname – holds the name of the node whose value you want to get.

Example

Following is a simple program which illustrates the usage of method getElementByTagName.

```
<!DOCTYPE html>
<html>
   <body>
      <div>
         <b>FirstName:</b> <span id = "FirstName"></span><br>
         <b>LastName:</b> <span id = "LastName"></span><br>
         <b>Category:</b> <span id = "Employee"></span><br>
      </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","/dom/node.xml",false);
         xmlhttp.send();
         xmlDoc = xmlhttp.responseXML;

         document.getElementById("FirstName").innerHTML =
         xmlDoc.getElementsByTagName("FirstName")[0].childNodes[0].nodeValue;
         document.getElementById("LastName").innerHTML =
         xmlDoc.getElementsByTagName("LastName")[0].childNodes[0].nodeValue;
         document.getElementById("Employee").innerHTML =
         xmlDoc.getElementsByTagName("Employee")[0].attributes[0].nodeValue;
      </script>
   </body>
</html>
```

In the above example, we are accessing the information of the nodes *FirstName*, *LastName* and *Employee*.

xmlDoc.getElementsByTagName("FirstName")[0].childNodes[0].nodeValue; This line accesses the value for the child node *FirstName* using the getElementByTagName() method.

xmlDoc.getElementsByTagName("Employee")[0].attributes[0].nodeValue; This line accesses the attribute value of the node *Employee* getElementByTagName() method.

Traversing Through Nodes

This is covered in the chapter DOM Traversing with examples.

Navigating Through Nodes

This is covered in the chapter DOM Navigation with examples.

XML DOM - Get Node

In this chapter, we will study about how to get the *node* value of a XML DOM object. XML documents have a hierarchy of informational units called nodes. Node object has a property *nodeValue*, which returns the value of the element.

In the following sections, we will discuss –

- Getting node value of an element
- Getting attribute value of a node

The *node.xml* used in all the following examples is as below –

```
<Company>
   <Employee category = "Technical">
      <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>
```

Get Node Value

The method *getElementsByTagName()* returns a *NodeList* of all the *Elements* in document order with a given tag name.

Example

The following example (getnode_example.htm) parses an XML document (node.xml) into an XML DOM object and extracts the node value of the child node *Firstname* (index at 0) –

```
<!DOCTYPE html>
<html>
   <body>
      <script>
         if (window.XMLHttpRequest) {
            xmlhttp = new XMLHttpRequest();
         } else{
            xmlhttp = new ActiveXObject("Microsoft.XMLHTTP");
         }
         xmlhttp.open("GET","/dom/node.xml",false);
         xmlhttp.send();
         xmlDoc = xmlhttp.responseXML;

         x = xmlDoc.getElementsByTagName('FirstName')[0]
         y = x.childNodes[0];
         document.write(y.nodeValue);
      </script>
   </body>
</html>
```

Execution

Save this file as *getnode_example.htm* on the server path (this file and node.xml should be on the same path in your server). In the output, we get the node value as *Tanmay*.

Get Attribute Value

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

The *getAttribute()* method retrieves an attribute value by element name.

Example

The following example (get_attribute_example.htm) parses an XML document (node.xml) into an XML DOM object and extracts the attribute value of the category *Employee* (index at 2) –

```
<!DOCTYPE html>
<html>
   <body>
      <script>
         if (window.XMLHttpRequest) {
            xmlhttp = new XMLHttpRequest();
         } else {
            xmlhttp = new ActiveXObject("Microsoft.XMLHTTP");
         }
```

```
         xmlhttp.open("GET","/dom/node.xml",false);
         xmlhttp.send();
         xmlDoc = xmlhttp.responseXML;

         x = xmlDoc.getElementsByTagName('Employee')[2];
         document.write(x.getAttribute('category'));
      </script>
   </body>
</html>
```

Execution

Save this file as *get_attribute_example.htm* on the server path (this file and node.xml should be on the same path in your server). In the output, we get the attribute value as *Management*.

XML DOM - Set Node

In this chapter, we will study about how to change the values of nodes in an XML DOM object. Node value can be changed as follows –

```
var value = node.nodeValue;
```

If *node* is an *Attribute* then the *value* variable will be the value of the attribute; if *node* is a *Text* node it will be the text content; if *node* is an *Element* it will be *null*.

Following sections will demonstrate the node value setting for each node type (attribute, text node and element).

The *node.xml* used in all the following examples is as below –

```
<Company>
   <Employee category = "Technical">
      <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>
```

Change Value of Text Node

When we, say the change value of Node element we mean to edit the text content of an element (which is also called the *text node*). Following example demonstrates how to change the text node of an element.

Example

The following example (set_text_node_example.htm) parses an XML document (node.xml) into an XML DOM object and change the value of an element's text node. In this case, *Email* of each *Employee* to *support@xyz.com* and print the values.

```
<!DOCTYPE html>
<html>
   <head>
      <script>
         function loadXMLDoc(filename) {
            if (window.XMLHttpRequest) {
               xhttp = new XMLHttpRequest();
            } else // code for IE5 and IE6 {
                xhttp = new ActiveXObject("Microsoft.XMLHTTP");
            }
            xhttp.open("GET",filename,false);
            xhttp.send();
            return xhttp.responseXML;
         }
      </script>
   </head>
   <body>
      <script>
         xmlDoc = loadXMLDoc("/dom/node.xml");

         x = xmlDoc.getElementsByTagName("Email");
         for(i = 0;i<x.length;i++) {

            x[i].childNodes[0].nodeValue = "support@xyz.com";
            document.write(i+');
            document.write(x[i].childNodes[0].nodeValue);
            document.write('<br>');
         }

      </script>
   </body>
</html>
```

Execution

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

```
0) support@xyz.com
1) support@xyz.com
2) support@xyz.com
```

Change Value of Attribute Node

The following example demonstrates how to change the attribute node of an element.

Example

The following example (set_attribute_example.htm) parses an XML document (node.xml) into an XML DOM object and changes the value of an element's attribute node. In this case, the *Category* of each *Employee* to *admin-0, admin-1, admin-2* respectively and print the values.

```
<!DOCTYPE html>
<html>
   <head>
      <script>
         function loadXMLDoc(filename) {
            if (window.XMLHttpRequest) {
                xhttp = new XMLHttpRequest();
            } else // code for IE5 and IE6 {
                xhttp = new ActiveXObject("Microsoft.XMLHTTP");
            }
            xhttp.open("GET",filename,false);
            xhttp.send();
            return xhttp.responseXML;
         }
      </script>
   </head>
   <body>
      <script>
         xmlDoc = loadXMLDoc("/dom/node.xml");

         x = xmlDoc.getElementsByTagName("Employee");
         for(i = 0 ;i<x.length;i++){

            newcategory = x[i].getAttributeNode('category');
            newcategory.nodeValue = "admin-"+i;
            document.write(i+');
            document.write(x[i].getAttributeNode('category').nodeValue);
            document.write('<br>');
         }

      </script>
   </body>
</html>
```

Execution

Save this file as *set_node_attribute_example.htm* on the server path (this file and node.xml should be on the same path in your server). The result would be as below –

```
0) admin-0
1) admin-1
2) admin-2
```

XML DOM - Create Node

In this chapter, we will discuss how to create new nodes using a couple of methods of the document object. These methods provide a scope to create new *element node, text node, comment node, CDATA section node and attribute node*. If the newly created node already exists in the element object, it is replaced by the new one. Following sections demonstrate this with examples.

Create new *Element* node

The method *createElement()* creates a new element node. If the newly created element node exists in the element object, it is replaced by the new one.

Syntax

Syntax to use the *createElement()* method is as follows –

```
var_name = xmldoc.createElement("tagname");
```

Where,

var_name – is the user-defined variable name which holds the name of new element.

("tagname") – is the name of new element node to be created.

Example

The following example (createnewelement_example.htm) parses an XML document (node.xml) into an XML DOM object and creates a new element node *PhoneNo* in the XML document.

```
<!DOCTYPE html>
<html>
   <head>
      <script>
         function loadXMLDoc(filename) {
            if (window.XMLHttpRequest) {
               xhttp = new XMLHttpRequest();
            } else // code for IE5 and IE6 {
               xhttp = new ActiveXObject("Microsoft.XMLHTTP");
            }
            xhttp.open("GET",filename,false);
            xhttp.send();
            return xhttp.responseXML;
         }
      </script>
   </head>
   <body>
      <script>
         xmlDoc = loadXMLDoc("/dom/node.xml");

         new_element = xmlDoc.createElement("PhoneNo");
```

```
         x = xmlDoc.getElementsByTagName("FirstName")[0];
         x.appendChild(new_element);

         document.write(x.getElementsByTagName("PhoneNo")[0].nodeName);
      </script>
   </body>
</html>
```

new_element = xmlDoc.createElement("PhoneNo"); creates the new element node <PhoneNo>

x.appendChild(new_element); x holds the name of the specified child node <FirstName> to which the new element node is appended.

Execution

Save this file as *createnewelement_example.htm* on the server path (this file and node.xml should be on the same path in your server). In the output we get the attribute value as *PhoneNo*.

Create new *Text* node

The method *createTextNode()* creates a new text node.

Syntax

Syntax to use *createTextNode()* is as follows −

```
var_name = xmldoc.createTextNode("tagname");
```

Where,

var_name − it is the user-defined variable name which holds the name of new text node.

("tagname") − within the parenthesis is the name of new text node to be created.

Example

The following example (createtextnode_example.htm) parses an XML document (node.xml) into an XML DOM object and creates a new text node *Im new text node* in the XML document.

```
<!DOCTYPE html>
<html>
   <head>
      <script>
         function loadXMLDoc(filename) {
            if (window.XMLHttpRequest) {
               xhttp = new XMLHttpRequest();
            } else // code for IE5 and IE6 {
               xhttp = new ActiveXObject("Microsoft.XMLHTTP");
            }
```

```
            xhttp.open("GET",filename,false);
            xhttp.send();
            return xhttp.responseXML;
         }
      </script>
   </head>
   <body>
      <script>
         xmlDoc = loadXMLDoc("/dom/node.xml");

         create_e = xmlDoc.createElement("PhoneNo");
         create_t = xmlDoc.createTextNode("Im new text node");
         create_e.appendChild(create_t);

         x = xmlDoc.getElementsByTagName("Employee")[0];
         x.appendChild(create_e);


         document.write(" PhoneNO: ");
         document.write(x.getElementsByTagName("PhoneNo")[0].childNodes[0].nodeValue);
      </script>
    </body>
</html>
```

Details of the above code are as below −

create_e = xmlDoc.createElement("PhoneNo"); creates a new element *<PhoneNo>*.

create_t = xmlDoc.createTextNode("Im new text node"); creates a new text node *"Im new text node"*.

x.appendChild(create_e); the text node, *"Im new text node"* is appended to the element, *<PhoneNo>*.

document.write(x.getElementsByTagName("PhoneNo")[0].childNodes[0].nodeValue); writes the new text node value to the element <PhoneNo>.

Execution

Save this file as *createtextnode_example.htm* on the server path (this file and node.xml should be on the same path in your server). In the output, we get the attribute value as i.e. *PhoneNO: Im new text node*.

Create new *Comment* node

The method *createComment()* creates a new comment node. Comment node is included in the program for the easy understanding of the code functionality.

Syntax

Syntax to use *createComment()* is as follows −

```
var_name = xmldoc.createComment("tagname");
```

Where,

var_name – is the user-defined variable name which holds the name of new comment node.

("tagname") – is the name of the new comment node to be created.

Example

The following example (createcommentnode_example.htm) parses an XML document (node.xml) into an XML DOM object and creates a new comment node, *"Company is the parent node"* in the XML document.

```
<!DOCTYPE html>
<html>
   <head>
      <script>
         function loadXMLDoc(filename) {
            if (window.XMLHttpRequest) {
               xhttp = new XMLHttpRequest();
            }
            else // code for IE5 and IE6 {
               xhttp = new ActiveXObject("Microsoft.XMLHTTP");
            }
            xhttp.open("GET",filename,false);
            xhttp.send();
            return xhttp.responseXML;
         }
      </script>
   </head>
   <body>
      <script>
         xmlDoc = loadXMLDoc("/dom/node.xml");

         create_comment = xmlDoc.createComment("Company is the parent node");

         x = xmlDoc.getElementsByTagName("Company")[0];

         x.appendChild(create_comment);

         document.write(x.lastChild.nodeValue);
      </script>
   </body>
</html>
```

In the above example –

create_comment = xmlDoc.createComment("Company is the parent node") **creates a specified comment line**.

x.appendChild(create_comment) In this line, *'x'* holds the name of the element <Company> to which the comment line is appended.

Execution

Save this file as *createcommentnode_example.htm* on the server path (this file and the node.xml should be on the same path in your server). In the output, we get the attribute value as *Company is the parent node* .

Create New *CDATA Section* Node

The method *createCDATASection()* creates a new CDATA section node. If the newly created CDATA section node exists in the element object, it is replaced by the new one.

Syntax

Syntax to use *createCDATASection()* is as follows –

```
var_name = xmldoc.createCDATASection("tagname");
```

Where,

var_name – is the user-defined variable name which holds the name of new the CDATA section node.

("tagname") – is the name of new CDATA section node to be created.

Example

The following example (createcdatanode_example.htm) parses an XML document (node.xml) into an XML DOM object and creates a new CDATA section node, *"Create CDATA Example"* in the XML document.

```
<!DOCTYPE html>
<html>
   <head>
      <script>
         function loadXMLDoc(filename) {
            if (window.XMLHttpRequest) {
               xhttp = new XMLHttpRequest();
            }
            else // code for IE5 and IE6 {
               xhttp = new ActiveXObject("Microsoft.XMLHTTP");
            }
            xhttp.open("GET",filename,false);
            xhttp.send();
            return xhttp.responseXML;
         }
      </script>
   </head>
   <body>
      <script>
         xmlDoc = loadXMLDoc("/dom/node.xml");

         create_CDATA = xmlDoc.createCDATASection("Create CDATA Example");

         x = xmlDoc.getElementsByTagName("Employee")[0];
         x.appendChild(create_CDATA);
```

```
         document.write(x.lastChild.nodeValue);
      </script>
   </body>
</html>
```

In the above example –

create_CDATA = xmlDoc.createCDATASection("Create CDATA Example") creates a new CDATA section node, *"Create CDATA Example"*

x.appendChild(create_CDATA) here, *x* holds the specified element <Employee> indexed at 0 to which the CDATA node value is appended.

Execution

Save this file as *createcdatanode_example.htm* on the server path (this file and node.xml should be on the same path in your server). In the output, we get the attribute value as *Create CDATA Example*.

Create new *Attribute* node

To create a new attribute node, the method *setAttributeNode()* is used. If the newly created attribute node exists in the element object, it is replaced by the new one.

Syntax

Syntax to use the *createElement()* method is as follows –

```
var_name = xmldoc.createAttribute("tagname");
```

Where,

var_name – is the user-defined variable name which holds the name of new attribute node.

("tagname") – is the name of new attribute node to be created.

Example

The following example (createattributenode_example.htm) parses an XML document (node.xml) into an XML DOM object and creates a new attribute node *section* in the XML document.

```
<!DOCTYPE html>
<html>
   <head>
      <script>
         function loadXMLDoc(filename) {
            if (window.XMLHttpRequest) {
               xhttp = new XMLHttpRequest();
            } else // code for IE5 and IE6 {
               xhttp = new ActiveXObject("Microsoft.XMLHTTP");
            }
```

```
            xhttp.open("GET",filename,false);
            xhttp.send();
            return xhttp.responseXML;
         }
      </script>
   </head>
   <body>
      <script>
         xmlDoc = loadXMLDoc("/dom/node.xml");

         create_a = xmlDoc.createAttribute("section");
         create_a.nodeValue = "A";

         x = xmlDoc.getElementsByTagName("Employee");
         x[0].setAttributeNode(create_a);
         document.write("New Attribute: ");
         document.write(x[0].getAttribute("section"));

      </script>
   </body>
</html>
```

In the above example –

create_a=xmlDoc.createAttribute("Category") creates an attribute with the name <section>.

create_a.nodeValue="Management" creates the value *"A"* for the attribute <section>.

x[0].setAttributeNode(create_a) this attribute value is set to the node element <Employee> indexed at 0.

XML DOM - Add Node

In this chapter, we will discuss the nodes to the existing element. It provides a means to –

append new child nodes before or after the existing child nodes

insert data within the text node

add attribute node

Following methods can be used to add/append the nodes to an element in a DOM –

appendChild()

insertBefore()

insertData()

appendChild()

The method appendChild() adds the new child node after the existing child node.

Syntax

Syntax of appendChild() method is as follows –

```
Node appendChild(Node newChild) throws DOMException
```

Where,

newChild – Is the node to add

This method returns the *Node* added.

Example

The following example (appendchildnode_example.htm) parses an XML document (node.xml) into an XML DOM object and appends new child *PhoneNo* to the element <FirstName>.

```
<!DOCTYPE html>
<html>
   <head>
      <script>
         function loadXMLDoc(filename) {
            if (window.XMLHttpRequest) {
               xhttp = new XMLHttpRequest();
            } else // code for IE5 and IE6 {
               xhttp = new ActiveXObject("Microsoft.XMLHTTP");
            }
            xhttp.open("GET",filename,false);
            xhttp.send();
            return xhttp.responseXML;
         }
      </script>
   </head>
   <body>
      <script>
         xmlDoc = loadXMLDoc("/dom/node.xml");

         create_e = xmlDoc.createElement("PhoneNo");

         x = xmlDoc.getElementsByTagName("FirstName")[0];
         x.appendChild(create_e);

         document.write(x.getElementsByTagName("PhoneNo")[0].nodeName);
      </script>
   </body>
</html>
```

In the above example –

using the method createElement(), a new element *PhoneNo* is created.

The new element *PhoneNo* is added to the element *FirstName* using the method appendChild().

Execution

Save this file as *appendchildnode_example.htm* on the server path (this file and node.xml should be on the same path in your server). In the output, we get the attribute value as *PhoneNo*.

insertBefore()

The method *insertBefore()*, inserts the new child nodes before the specified child nodes.

Syntax

Syntax of insertBefore() method is as follows –

```
Node insertBefore(Node newChild, Node refChild) throws DOMException
```

Where,

newChild – Is the node to insert

refChild – Is the reference node, i.e., the node before which the new node must be inserted.

This method returns the *Node* being inserted.

Example

The following example (insertnodebefore_example.htm) parses an XML document (node.xml) into an XML DOM object and inserts new child *Email* before the specified element <Email>.

```
<!DOCTYPE html>
<html>
   <head>
      <script>
         function loadXMLDoc(filename) {
            if (window.XMLHttpRequest) {
               xhttp = new XMLHttpRequest();
            } else // code for IE5 and IE6 {
               xhttp = new ActiveXObject("Microsoft.XMLHTTP");
            }
            xhttp.open("GET",filename,false);
            xhttp.send();
            return xhttp.responseXML;
         }
      </script>
   </head>
   <body>
      <script>
         xmlDoc = loadXMLDoc("/dom/node.xml");

         create_e = xmlDoc.createElement("Email");

         x = xmlDoc.documentElement;
         y = xmlDoc.getElementsByTagName("Email");

         document.write("No of Email elements before inserting was: " + y.length);
```

```
         document.write("<br>");
         x.insertBefore(create_e,y[3]);

         y=xmlDoc.getElementsByTagName("Email");
         document.write("No of Email elements after inserting is: " + y.length);
      </script>
   </body>
</html>
```

In the above example –

using the method createElement(), a new element *Email* is created.

The new element *Email* is added before the element *Email* using the method insertBefore().

y.length gives the total number of elements added before and after the new element.

Execution

Save this file as *insertnodebefore_example.htm* on the server path (this file and node.xml should be on the same path in your server). We will receive the following output –

```
No of Email elements before inserting was: 3
No of Email elements after inserting is: 4
```

insertData()

The method insertData(), inserts a string at the specified 16-bit unit offset.

Syntax

The insertData() has the following syntax –

```
void insertData(int offset, java.lang.String arg) throws DOMException
```

Where,

offset – is the character offset at which to insert.

arg – is the key word to insert the data. It encloses the two parameters offset and string within the parenthesis separated by comma.

Example

The following example (addtext_example.htm) parses an XML document ("node.xml ") into an XML DOM object and inserts new data *MiddleName* at the specified position to the element <FirstName>.

```
<!DOCTYPE html>
<html>
   <head>
```

```
      <script>
         function loadXMLDoc(filename) {
            if (window.XMLHttpRequest) {
               xhttp = new XMLHttpRequest();
            } else // code for IE5 and IE6 {
               xhttp = new ActiveXObject("Microsoft.XMLHTTP");
            }
            xhttp.open("GET",filename,false);
            xhttp.send();
            return xhttp.responseXML;
         }
      </script>
   </head>
   <body>
      <script>
         xmlDoc = loadXMLDoc("/dom/node.xml");

         x = xmlDoc.getElementsByTagName("FirstName")[0].childNodes[0];
         document.write(x.nodeValue);
         x.insertData(6,"MiddleName");
         document.write("<br>");
         document.write(x.nodeValue);

      </script>
    </body>
</html>
```

x.insertData(6,"MiddleName"); – Here, *x* holds the name of the specified child name, i.e, <FirstName>. We then insert to this text node the data *"MiddleName"* starting from position 6.

Execution

Save this file as *addtext_example.htm* on the server path (this file and node.xml should be on the same path in your server). We will receive the following in the output –

```
Tanmay
TanmayMiddleName
```

XML DOM - Replace Node

In this chapter, we will study about the replace node operation in an XML DOM object. As we know everything in the DOM is maintained in a hierarchical informational unit known as node and the replacing node provides another way to update these specified nodes or a text node.

Following are the two methods to replace the nodes.

replaceChild()

replaceData()

replaceChild()

The method *replaceChild()* replaces the specified node with the new node.

Syntax

The insertData() has the following syntax –

```
Node replaceChild(Node newChild, Node oldChild) throws DOMException
```

Where,

newChild – is the new node to put in the child list.

oldChild – is the node being replaced in the list.

This method returns the node replaced.

Example

The following example (replacenode_example.htm) parses an XML document (node.xml) into an XML DOM object and replaces the specified node <FirstName> with the new node <Name>.

```
<!DOCTYPE html>
<html>
   <head>
      <script>
         function loadXMLDoc(filename) {
            if (window.XMLHttpRequest) {
               xhttp = new XMLHttpRequest();
            } else // code for IE5 and IE6 {
               xhttp = new ActiveXObject("Microsoft.XMLHTTP");
            }
            xhttp.open("GET",filename,false);
            xhttp.send();
            return xhttp.responseXML;
         }
      </script>
   </head>
   <body>
      <script>
         xmlDoc = loadXMLDoc("/dom/node.xml");

         x = xmlDoc.documentElement;

         z = xmlDoc.getElementsByTagName("FirstName");
         document.write("<b>Content of FirstName element before replace operation</b><br>");
         for (i=0;i<z.length;i++) {
            document.write(z[i].childNodes[0].nodeValue);
            document.write("<br>");
         }
         //create a Employee element, FirstName element and a text node
         newNode = xmlDoc.createElement("Employee");
         newTitle = xmlDoc.createElement("Name");
         newText = xmlDoc.createTextNode("MS Dhoni");

         //add the text node to the title node,
         newTitle.appendChild(newText);
         //add the title node to the book node
```

```
         newNode.appendChild(newTitle);

         y = xmlDoc.getElementsByTagName("Employee")[0]
         //replace the first book node with the new node
         x.replaceChild(newNode,y);

         z = xmlDoc.getElementsByTagName("FirstName");
         document.write("<b>Content of FirstName element after replace operation</b><br>");
         for (i = 0;i<z.length;i++) {
            document.write(z[i].childNodes[0].nodeValue);
            document.write("<br>");
         }
      </script>
   </body>
</html>
```

Execution

Save this file as replacenode_example.htm on the server path (this file and node.xml should be on the same path in your server). We will get the output as shown below –

```
Content of FirstName element before replace operation
Tanmay
Taniya
Tanisha

Content of FirstName element after replace operation
Taniya
Tanisha
```

replaceData()

The method replaceData() replaces the characters starting at the specified 16-bit unit offset with the specified string.

Syntax

The replaceData() has the following syntax –

```
void replaceData(int offset, int count, java.lang.String arg) throws DOMException
```

Where

offset – is the offset from which to start replacing.

count – is the number of 16-bit units to replace. If the sum of offset and count exceeds length, then all the 16-bit units to the end of the data are replaced.

arg – the *DOMString* with which the range must be replaced.

Example

The following example (replacedata_example.htm) parses an XML document (node.xml) into an XML DOM object and replaces it.

```
<!DOCTYPE html>
<html>
   <head>
      <script>
         function loadXMLDoc(filename) {
            if (window.XMLHttpRequest) {
               xhttp = new XMLHttpRequest();
            } else // code for IE5 and IE6 {
               xhttp = new ActiveXObject("Microsoft.XMLHTTP");
            }
            xhttp.open("GET",filename,false);
            xhttp.send();
            return xhttp.responseXML;
         }
      </script>
   </head>
   <body>
      <script>
         xmlDoc = loadXMLDoc("/dom/node.xml");

         x = xmlDoc.getElementsByTagName("ContactNo")[0].childNodes[0];
         document.write("<b>ContactNo before replace operation:</b> "+x.nodeValue);
         x.replaceData(1,5,"9999999");
         document.write("<br>");
         document.write("<b>ContactNo after replace operation:</b> "+x.nodeValue);

      </script>
   </body>
</html>
```

In the above example –

x.replaceData(2,3,"999"); – Here *x* holds the text of the specified element <ContactNo> whose text is replaced by the new text *"9999999"*, starting from the position *1* till the length of *5*.

Execution

Save this file as *replacedata_example.htm* on the server path (this file and node.xml should be on the same path in your server). We will get the output as shown below –

```
ContactNo before replace operation: 1234567890


ContactNo after replace operation: 199999997890
```

XML DOM - Remove Node

In this chapter, we will study about the XML DOM *Remove Node* operation. The remove node operation removes the specified node from the document. This operation can be implemented to remove the nodes like text node, element node or an attribute node.

Following are the methods that are used for remove node operation –

removeChild()

removeAttribute()

removeChild()

The method *removeChild()* removes the child node indicated by *oldChild* from the list of children, and returns it. Removing a child node is equivalent to removing a text node. Hence, removing a child node removes the text node associated with it.

Syntax

The syntax to use removeChild() is as follows –

```
Node removeChild(Node oldChild) throws DOMException
```

Where,

oldChild – is the node being removed.

This method returns the node removed.

Example - Remove Current Node

The following example (removecurrentnode_example.htm) parses an XML document (node.xml) into an XML DOM object and removes the specified node <ContactNo> from the parent node.

```
<!DOCTYPE html>
<html>
   <head>
      <script>
         function loadXMLDoc(filename) {
            if (window.XMLHttpRequest) {
               xhttp = new XMLHttpRequest();
            } else // code for IE5 and IE6 {
               xhttp = new ActiveXObject("Microsoft.XMLHTTP");
            }
            xhttp.open("GET",filename,false);
            xhttp.send();
            return xhttp.responseXML;
         }
      </script>
   </head>
   <body>
      <script>
         xmlDoc = loadXMLDoc("/dom/node.xml");

         document.write("<b>Before remove operation, total ContactNo elements: </b>");
         document.write(xmlDoc.getElementsByTagName("ContactNo").length);
         document.write("<br>");

         x = xmlDoc.getElementsByTagName("ContactNo")[0];
```

```
        x.parentNode.removeChild(x);

        document.write("<b>After remove operation, total ContactNo elements: </b>");
        document.write(xmlDoc.getElementsByTagName("ContactNo").length);
      </script>
   </body>
</html>
```

In the above example –

x = xmlDoc.getElementsByTagName("ContactNo")[0] gets the element <ContactNo> indexed at 0.

x.parentNode.removeChild(x); removes the element <ContactNo> indexed at 0 from the parent node.

Execution

Save this file as *removecurrentnode_example.htm* on the server path (this file and node.xml should be on the same path in your server). We get the following result –

```
Before remove operation, total ContactNo elements: 3
After remove operation, total ContactNo elements: 2
```

Example - Remove Text Node

The following example (removetextNode_example.htm) parses an XML document (node.xml) into an XML DOM object and removes the specified child node <FirstName>.

```
<!DOCTYPE html>
<html>
   <head>
      <script>
         function loadXMLDoc(filename) {
            if (window.XMLHttpRequest) {
               xhttp = new XMLHttpRequest();
            } else // code for IE5 and IE6 {
                xhttp = new ActiveXObject("Microsoft.XMLHTTP");
            }
            xhttp.open("GET",filename,false);
            xhttp.send();
            return xhttp.responseXML;
         }
      </script>
   </head>
   <body>
      <script>
         xmlDoc = loadXMLDoc("/dom/node.xml");

         x = xmlDoc.getElementsByTagName("FirstName")[0];

         document.write("<b>Text node of child node before removal is:</b> ");
         document.write(x.childNodes.length);
         document.write("<br>");

         y = x.childNodes[0];
```

```
         x.removeChild(y);
         document.write("<b>Text node of child node after removal is:</b> ");
         document.write(x.childNodes.length);

      </script>
   </body>
</html>
```

In the above example –

x = xmlDoc.getElementsByTagName("FirstName")[0]; – gets the first element <FirstName> to the *x* indexed at 0.

y = x.childNodes[0]; – in this line *y* holds the child node to be remove.

x.removeChild(y); – removes the specified child node.

Execution

Save this file as *removetextNode_example.htm* on the server path (this file and node.xml should be on the same path in your server). We get the following result –

```
Text node of child node before removal is: 1
Text node of child node after removal is: 0
```

removeAttribute()

The method removeAttribute() removes an attribute of an element by name.

Syntax

Syntax to use *removeAttribute()* is as follows –

```
void removeAttribute(java.lang.String name) throws DOMException
```

Where,

name – is the name of the attribute to remove.

Example

The following example (removeelementattribute_example.htm) parses an XML document (node.xml) into an XML DOM object and removes the specified attribute node.

```
<!DOCTYPE html>
<html>
   <head>
      <script>
         function loadXMLDoc(filename) {
            if (window.XMLHttpRequest) {
               xhttp = new XMLHttpRequest();
            } else // code for IE5 and IE6 {
                xhttp = new ActiveXObject("Microsoft.XMLHTTP");
            }
            xhttp.open("GET",filename,false);
```

```
            xhttp.send();
            return xhttp.responseXML;
         }
      </script>
   </head>
   <body>

      <script>
         xmlDoc = loadXMLDoc("/dom/node.xml");

         x = xmlDoc.getElementsByTagName('Employee');

         document.write(x[1].getAttribute('category'));
         document.write("<br>");

         x[1].removeAttribute('category');

         document.write(x[1].getAttribute('category'));

      </script>
   </body>
</html>
```

In the above example –

document.write(x[1].getAttribute('category')); – value of attribute *category* indexed at 1st position is invoked.

x[1].removeAttribute('category'); – removes the attribute value.

Execution

Save this file as *removeelementattribute_example.htm* on the server path (this file and node.xml should be on the same path in your server). We get the following result –

```
Non-Technical
null
```

XML DOM - Clone Node

In this chapter, we will discucss the *Clone Node* operation on XML DOM object. Clone node operation is used to create a duplicate copy of the specified node. *cloneNode()* is used for this operation.

cloneNode()

This method returns a duplicate of this node, i.e., serves as a generic copy constructor for nodes. The duplicate node has no parent (parentNode is null) and no user data.

Syntax

The *cloneNode()* method has the following syntax –

```
Node cloneNode(boolean deep)
```

deep – If true, recursively clones the subtree under the specified node; if false, clone only the node itself (and its attributes, if it is an Element).

This method returns the duplicate node.

Example

The following example (clonenode_example.htm) parses an XML document (node.xml) into an XML DOM object and creates a deep copy of the first *Employee* element.

```
<!DOCTYPE html>
<html>
   <head>
      <script>
         function loadXMLDoc(filename) {
            if (window.XMLHttpRequest) {
               xhttp = new XMLHttpRequest();
            } else // code for IE5 and IE6 {
               xhttp = new ActiveXObject("Microsoft.XMLHTTP");
            }
            xhttp.open("GET",filename,false);
            xhttp.send();
            return xhttp.responseXML;
         }
      </script>
   </head>
   <body>
      <script>
         xmlDoc = loadXMLDoc("/dom/node.xml");

         x = xmlDoc.getElementsByTagName('Employee')[0];
         clone_node = x.cloneNode(true);
         xmlDoc.documentElement.appendChild(clone_node);

         firstname = xmlDoc.getElementsByTagName("FirstName");
         lastname = xmlDoc.getElementsByTagName("LastName");
         contact = xmlDoc.getElementsByTagName("ContactNo");
         email = xmlDoc.getElementsByTagName("Email");
         for (i = 0;i < firstname.length;i++) {
            document.write(firstname[i].childNodes[0].nodeValue+'
               '+lastname[i].childNodes[0].nodeValue+',
               '+contact[i].childNodes[0].nodeValue+',  '+email[i].childNodes[0].nodeValue);
            document.write("<br>");
         }
      </script>
   </body>
</html>
```

As you can see in the above example, we have set the *cloneNode()* param to *true*. Hence each of the child element under the *Employee* element is copied or cloned.

Execution

Save this file as *clonenode_example.htm* on the server path (this file and node.xml should be on the same path in your server). We will get the output as shown below –

```
Tanmay Patil, 1234567890, tanmaypatil@xyz.com
Taniya Mishra, 1234667898, taniyamishra@xyz.com
Tanisha Sharma, 1234562350, tanishasharma@xyz.com
Tanmay Patil, 1234567890, tanmaypatil@xyz.com
```

You will notice that the first *Employee* element is cloned completely.

DOM - Node Object

Node interface is the primary datatype for the entire Document Object Model. The node is used to represent a single XML element in the entire document tree.

A node can be any type that is an attribute node, a text node or any other node. The attributes *nodeName, nodeValue* and *attributes* are included as a mechanism to get at node information without casting down to the specific derived interface.

Attributes

The following table lists the attributes of the *Node* object –

Attribute	Type	Description
attributes	NamedNodeMap	This is of type *NamedNodeMap* containing the attributes of this node (if it is an Element) or null otherwise. *This has been removed. Refer* ***specs***
baseURI	DOMString	It is used to specify absolute base URI of the node.
childNodes	NodeList	It is a *NodeList* that contains all children of this node. If there are no children, this is a *NodeList* containing no nodes.
firstChild	Node	It specifies the first child of a node.
lastChild	Node	It specifies the last child of a node.
localName	DOMString	It is used to specify the name of the local part of a node. *This has been removed. Refer* ***specs*** .
namespaceURI	DOMString	It specifies the namespace URI of a node. *This has been removed. Refer* ***specs***
nextSibling	Node	It returns the node immediately following this node. If there is no such node, this returns null.
nodeName	DOMString	The name of this node, depending on its type.
nodeType	unsigned short	It is a code representing the type of the underlying object.

nodeValue	DOMString	It is used to specify the value of a node depending on their types.
ownerDocument	Document	It specifies the *Document* object associated with the node.
parentNode	Node	This property specifies the parent node of a node.
prefix	DOMString	This property returns the namespace prefix of a node. *This has been removed. Refer* ***specs***
previousSibling	Node	This specifies the node immediately preceding the current node.
textContent	DOMString	This specifies the textual content of a node.

Node Types

We have listed the node types as below –

ELEMENT_NODE

ATTRIBUTE_NODE

ENTITY_NODE

ENTITY_REFERENCE_NODE

DOCUMENT_FRAGMENT_NODE

TEXT_NODE

CDATA_SECTION_NODE

COMMENT_NODE

PROCESSING_INSTRUCTION_NODE

DOCUMENT_NODE

DOCUMENT_TYPE_NODE

NOTATION_NODE

Methods

Below table lists the different Node Object methods –

S.No.	Method & Description
1	**appendChild(Node newChild)** This method adds a node after the last child node of the specified element node. It returns the added node.

2	**cloneNode(boolean deep)** This method is used to create a duplicate node, when overridden in a derived class. It returns the duplicated node.
3	**compareDocumentPosition(Node other)** This method is used to compare the position of the current node against a specified node according to the document order. Returns *unsigned short*, how the node is positioned relatively to the reference node.
4	**getFeature(DOMString feature, DOMString version)** Returns the DOM Object which implements the specialized APIs of the specified feature and version, if any, or null if there is no object. *This has been removed. Refer* ***specs*** .
5	**getUserData(DOMString key)** Retrieves the object associated to a key on this node. The object must first have been set to this node by calling the setUserData with the same key. Returns the DOMUserData associated to the given key on this node, or null if there was none. *This has been removed. Refer* ***specs*** .
6	**hasAttributes()** Returns whether this node (if it is an element) has any attributes or not. Returns *true if* any attribute is present in the specified node else returns *false*. *This has been removed. Refer* ***specs*** .
7	**hasChildNodes()** Returns whether this node has any children. This method returns *true* if the current node has child nodes otherwise *false*.
8	**insertBefore(Node newChild, Node refChild)** This method is used to insert a new node as a child of this node, directly before an existing child of this node. It returns the node being inserted.
9	**isDefaultNamespace(DOMString namespaceURI)** This method accepts a namespace URI as an argument and returns a *Boolean* with a value of *true* if the namespace is the default namespace on the given node or *false* if not.
10	**isEqualNode(Node arg)**

	This method tests whether two nodes are equal. Returns *true* if the nodes are equal, *false* otherwise.
11	**isSameNode(Node other)** This method returns whether current node is the same node as the given one. Returns *true* if the nodes are the same, *false* otherwise. *This has been removed. Refer **specs*** .
12	**isSupported(DOMString feature, DOMString version)** This method returns whether the specified DOM module is supported by the current node. Returns *true* if the specified feature is supported on this node, *false* otherwise. *This has been removed. Refer **specs*** .
13	**lookupNamespaceURI(DOMString prefix)** This method gets the URI of the namespace associated with the namespace prefix.
14	**lookupPrefix(DOMString namespaceURI)** This method returns the closest prefix defined in the current namespace for the namespace URI. Returns an associated namespace prefix if found or null if none is found.
15	**normalize()** Normalization adds all the text nodes including attribute nodes which define a normal form where structure of the nodes which contain elements, comments, processing instructions, CDATA sections, and entity references separates the text nodes, i.e, neither adjacent Text nodes nor empty Text nodes.
16	**removeChild(Node oldChild)** This method is used to remove a specified child node from the current node. This returns the node removed.
17	**replaceChild(Node newChild, Node oldChild)** This method is used to replace the old child node with a new node. This returns the node replaced.
18	**setUserData(DOMString key, DOMUserData data, UserDataHandler handler)**

	This method associates an object to a key on this node. The object can later be retrieved from this node by calling *getUserData* with the same key. This returns the *DOMUserData* previously associated to the given key on this node. *This has been removed. Refer* ***specs*** .

DOM - NodeList Object

The NodeList object specifies the abstraction of an ordered collection of nodes. The items in the NodeList are accessible via an integral index, starting from 0.

Attributes

The following table lists the attributes of the NodeList object –

Attribute	Type	Description
length	unsigned long	It gives the number of nodes in the node list.

Methods

The following is the only method of the NodeList object.

S.No.	Method & Description
1	**item()** It returns the *index*th item in the collection. If index is greater than or equal to the number of nodes in the list, this returns null.

DOM - NamedNodeMap Object

The *NamedNodeMap* object is used to represent collections of nodes that can be accessed by name.

Attributes

The following table lists the Property of the NamedNodeMap Object.

Attribute	Type	Description
length	unsigned long	It gives the number of nodes in this map. The range of valid child node indices is 0 to length-1 inclusive.

Methods

The following table lists the methods of the *NamedNodeMap* object.

S.No.	Methods & Description
1	**getNamedItem ()** Retrieves the node specified by name.
2	**getNamedItemNS ()** Retrieves a node specified by local name and namespace URI.
3	**item ()** Returns the *index*th item in the map. If index is greater than or equal to the number of nodes in this map, this returns null.
4	**removeNamedItem ()** Removes a node specified by name.
5	**removeNamedItemNS ()** Removes a node specified by local name and namespace URI.
6	**setNamedItem ()** Adds a node using its *nodeName attribute. If a node with that name is already present in this map, it is replaced by the new one.*
7	**setNamedItemNS ()** Adds a node using its *namespaceURI* and *localName*. If a node with that namespace URI and that local name is already present in this map, it is replaced by the new one. Replacing a node by itself has no effect.

DOM - DOMImplementation Object

The *DOMImplementation* object provides a number of methods for performing operations that are independent of any particular instance of the document object model.

Methods

Following table lists the methods of the *DOMImplementation* object –

S.No.	Method & Description

1	**createDocument(namespaceURI, qualifiedName, doctype)** It creates a DOM Document object of the specified type with its document element.
2	**createDocumentType(qualifiedName, publicId, systemId)** It creates an empty *DocumentType* node.
3	**getFeature(feature, version)** This method returns a specialized object which implements the specialized APIs of the specified feature and version. *This has been removed. Refer* ***specs*** .
4	**hasFeature(feature, version)** This method tests if the DOM implementation implements a specific feature and version.

DOM - DocumentType Object

The *DocumentType* objects are the key to access the document's data and in the document, the doctype attribute can have either the null value or the DocumentType Object value. These DocumentType objects act as an interface to the entities described for an XML document.

Attributes

The following table lists the attributes of the *DocumentType* object –

Attribute	Type	Description
name	DOMString	It returns the name of the DTD which is written immediately next to the keyword !DOCTYPE.
entities	NamedNodeMap	It returns a NamedNodeMap object containing the general entities, both external and internal, declared in the DTD.
notations	NamedNodeMap	It returns a NamedNodeMap containing the notations declared in the DTD.
internalSubset	DOMString	It returns an internal subset as a string, or null if there is none. *This has been removed. Refer* ***specs*** .
publicId	DOMString	It returns the public identifier of the external subset.
systemId	DOMString	It returns the system identifier of the external subset. This may be an absolute URI or not.

Methods

DocumentType inherits methods from its parent, *Node*, and implements the *ChildNode* interface.

DOM - ProcessingInstruction Object

ProcessingInstruction gives that application-specific information which is generally included in the prolog section of the XML document.

Processing instructions (PIs) can be used to pass information to applications. PIs can appear anywhere in the document outside the markup. They can appear in the prolog, including the document type definition (DTD), in textual content, or after the document.

A PI starts with a special tag **<?** and ends with **?>**. Processing of the contents ends immediately after the string **?>** is encountered.

Attributes

The following table lists the attributes of the *ProcessingInstruction* object –

Attribute	Type	Description
data	DOMString	It is a character that describes the information for the application to process immediately preceding the ?>.
target	DOMString	This identifies the application to which the instruction or the data is directed.

DOM - Entity Object

Entity interface represents a known entity, either parsed or unparsed, in an XML document. The *nodeName* attribute that is inherited from *Node* contains the name of the entity.

An Entity object does not have any parent node, and all its successor nodes are read-only.

Attributes

The following table lists the attributes of the *Entity* object –

Attribute	Type	Description
inputEncoding	DOMString	This specifies the encoding used by the external parsed entity. Its value is *null* if it is an entity from the internal subset or if it is not known.

notationName	DOMString	For an unparsed entities, it gives the name of the notation and its value is *null* for the parsed entities.
publicId	DOMString	It gives the name of the public identifier associated with the entity.
systemId	DOMString	It gives the name of the system identifier associated with the entity.
xmlEncoding	DOMString	It gives the xml encoding included as a part of the text declaration for the external parsed entity, null otherwise.
xmlVersion	DOMString	It gives the xml version included as a part of the text declaration for the external parsed entity, null otherwise.

DOM - Entity Reference Object

The *EntityReference* objects are the general entity references which are inserted into the XML document providing scope to replace the text. The EntityReference Object does not work for the pre-defined entities since they are considered to be expanded by the HTML or the XML processor.

This interface does not have properties or methods of its own but inherits from *Node*.

DOM - Notation Object

In this chapter, we will study about the XML DOM *Notation object*. The notation object property provides a scope to recognize the format of elements with a notation attribute, a particular processing instruction or a non-XML data. The Node Object properties and methods can be performed on the Notation Object since that is also considered as a Node.

This object inherits methods and properties from *Node*. Its *nodeName* is the notation name. Has no parent.

Attributes

The following table lists the attributes of the *Notation* object –

Attribute	Type	Description
publicID	DOMString	It gives the name of the public identifier associated with the notation.
systemID	DOMString	It gives the name of the system identifier associated with the notation.

DOM - Element Object

The XML elements can be defined as building blocks of XML. Elements can behave as containers to hold text, elements, attributes, media objects or all of these. Whenever parser parses an XML document against the well-formedness, parser navigates through an element node. An element node contains the text within it which is called as the text node.

Element object inherits the properties and the methods of the Node object as element object is also considered as a Node. Other than the node object properties and methods it has the following properties and methods.

Properties

The following table lists the attributes of the *Element* object –

Attribute	Type	Description
tagName	DOMString	It gives the name of the tag for the specified element.
schemaTypeInfo	TypeInfo	It represents the type information associated with this element. *This has been removed. Refer* ***specs*** .

Methods

Below table lists the Element Object methods –

Methods	Type	Description
getAttribute()	DOMString	Retrieves the value of the attribute if exists for the specified element.
getAttributeNS()	DOMString	Retrieves an attribute value by local name and namespace URI.
getAttributeNode()	Attr	Retrieves the name of the attribute node from the current element.
getAttributeNodeNS()	Attr	Retrieves an Attr node by local name and namespace URI.
getElementsByTagName()	NodeList	Returns a NodeList of all descendant Elements with a given tag name, in document order.
getElementsByTagNameNS()	NodeList	Returns a NodeList of all the descendant Elements with a given local name and namespace URI in document order.
hasAttribute()	boolean	Returns true when an attribute with a

		given name is specified on this element or has a default value, false otherwise.
hasAttributeNS()	boolean	Returns true when an attribute with a given local name and namespace URI is specified on this element or has a default value, false otherwise.
removeAttribute()	No Return Value	Removes an attribute by name.
removeAttributeNS	No Return Value	Removes an attribute by local name and namespace URI.
removeAttributeNode()	Attr	Specified attribute node is removed from the element.
setAttribute()	No Return Value	Sets a new attribute value to the existing element.
setAttributeNS()	No Return Value	Adds a new attribute. If an attribute with the same local name and namespace URI is already present on the element, its prefix is changed to be the prefix part of the qualifiedName, and its value is changed to be the value parameter.
setAttributeNode()	Attr	Sets a new attribute node to the existing element.
setAttributeNodeNS	Attr	Adds a new attribute. If an attribute with that local name and that namespace URI is already present in the element, it is replaced by the new one.
setIdAttribute	No Return Value	If the parameter isId is true, this method declares the specified attribute to be a user-determined ID attribute. *This has been removed. Refer* ***specs*** .
setIdAttributeNS	No Return Value	If the parameter isId is true, this method declares the specified attribute to be a user-determined ID attribute. *This has been removed. Refer* ***specs*** .

DOM - Attribute Object

Attr interface represents an attribute in an Element object. Typically, the allowable values for the attribute are defined in a schema associated with the document. *Attr* objects are

not considered as part of the document tree since they are not actually child nodes of the element they describe. Thus for the child nodes *parentNode*, *previousSibling* and *nextSibling* the attribute value is *null*.

Attributes

The following table lists the attributes of the *Attribute* object –

Attribute	Type	Description
name	DOMString	This gives the name of the attribute.
specified	boolean	It is a boolean value which returns true if the attribute value exists in the document.
value	DOMString	Returns the value of the attribute.
ownerElement	Element	It gives the node to which attribute is associated or null if attribute is not in use.
isId	boolean	It returns whether the attribute is known to be of type ID (i.e. to contain an identifier for its owner element) or not.

DOM - CDATASection Object

In this chapter, we will study about the XML DOM *CDATASection Object*. The text present within an XML document is parsed or unparsed depending on what it is declared. If the text is declared as Parse Character Data (PCDATA), it is parsed by the parser to convert an XML document into an XML DOM Object. On the other hand, if the text is declared as the unparsed Character Data (CDATA) the text within is not parsed by the XML parser. These are not considered as the markup and will not expand the entities.

The purpose of using the CDATASection object is to escape the blocks of text containing characters that would otherwise be regarded as markup. **"]]>"**, this is the only delimiter recognized in a CDATA section that ends the CDATA section.

The CharacterData.data attribute holds the text that is contained by the CDATA section. This interface inherits the *CharatcterData* interface through the *Text* interface.

> There are no methods and attributes defined for the CDATASection object. It only directly implements the *Text* interface.

DOM - Comment Object

In this chapter, we will study about the *Comment object*. Comments are added as a notes or the lines for understanding the purpose of an XML code. Comments can be used to include related links, information and terms. These may appear anywhere in the XML code.

The comment interface inherits the *CharacterData* interface representing the content of the comment.

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.

> There are no methods and attributes defined for the Comment object. It inherits those of its parent, *CharacterData*, and indirectly those of *Node*.

DOM - XMLHttpRequest Object

XMLHttpRequest object establishes a medium between a web page's client-side and server-side that can be used by the many scripting languages like JavaScript, JScript, VBScript and other web browser to transfer and manipulate the XML data.

With the XMLHttpRequest object it is possible to update the part of a web page without reloading the whole page, request and receive the data from a server after the page has been loaded and send the data to the server.

Syntax

An XMLHttpRequest object can be instatiated as follows –

```
xmlhttp = new XMLHttpRequest();
```

To handle all browsers, including IE5 and IE6, check if the browser supports the XMLHttpRequest object as below –

```
if(window.XMLHttpRequest) // for Firefox, IE7+, Opera, Safari, ... {
   xmlHttp = new XMLHttpRequest();
} else if(window.ActiveXObject) // for Internet Explorer 5 or 6 {
   xmlHttp = new ActiveXObject("Microsoft.XMLHTTP");
}
```

Examples to load an XML file using the XMLHttpRequest object can be referred here

Methods

The following table lists the methods of the XMLHttpRequest object –

S.No.	Method & Description
1	**abort()** Terminates the current request made.
2	**getAllResponseHeaders()** Returns all the response headers as a string, or null if no response has been received.
3	**getResponseHeader()** Returns the string containing the text of the specified header, or null if either the response has not yet been received or the header doesn't exist in the response.
4	**open(method,url,async,uname,pswd)** It is used in conjugation with the Send method to send the request to the server. The open method specifies the following parameters – **method** – specifies the type of request i.e. Get or Post. **url** – it is the location of the file. **async** – indicates how the request should be handled. It is boolean value. where, 'true' means the request is processed asynchronously without waiting for a Http response. 'false' means the request is processed synchronously after receiving the Http response. **uname** – is the username. **pswd** – is the password.
5	**send(string)** It is used to send the request working in conjugation with the Open method.
6	**setRequestHeader()** Header contains the label/value pair to which the request is sent.

Attributes

The following table lists the attributes of the XMLHttpRequest object −

S.No.	Attribute & Description
1	**onreadystatechange** It is an event based property which is set on at every state change.
2	**readyState** This describes the present state of the XMLHttpRequest object. There are five possible states of the readyState property − **readyState = 0** − means request is yet to initialize. **readyState = 1** − request is set. **readyState = 2** − request is sent. **readyState = 3** − request is processing. **readyState = 4** − request is completed.
3	**responseText** This property is used when the response from the server is a text file.
4	**responseXML** This property is used when the response from the server is an XML file.
5	**status** Gives the status of the Http request object as a number. For example, "404" or "200".
6	**statusText** Gives the status of the Http request object as a string. For example, "Not Found" or "OK".

Examples

node.xml contents are as below −

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

```
   <Employee category = "Technical">
      <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>
```

Retrieve specific information of a resource file

Following example demonstrates how to retrive specific information of a resource file using the method *getResponseHeader()* and the property *readState*.

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv = "content-type" content = "text/html; charset = iso-8859-2" />
         <script>
            function loadXMLDoc() {
               var xmlHttp = null;
               if(window.XMLHttpRequest) // for Firefox, IE7+, Opera, Safari, ... {
                  xmlHttp = new XMLHttpRequest();
               }
               else if(window.ActiveXObject) // for Internet Explorer 5 or 6 {
                  xmlHttp = new ActiveXObject("Microsoft.XMLHTTP");
               }

               return xmlHttp;
            }

            function makerequest(serverPage, myDiv) {
               var request =  loadXMLDoc();
               request.open("GET", serverPage);
               request.send(null);

               request.onreadystatechange = function() {
                  if (request.readyState == 4) {
                     document.getElementById(myDiv).innerHTML = request.getResponseHeader("Conten
                  }
               }
            }
      </script>
   </head>
   <body>
      <button type = "button" onclick="makerequest('/dom/node.xml', 'ID')">Click me to get the sp
```

```
      <div id = "ID">Specific header information is returned.</div>
   </body>
</html>
```

Execution

Save this file as *elementattribute_removeAttributeNS.htm* on the server path (this file and node_ns.xml should be on the same path in your server). We will get the output as shown below –

```
Before removing the attributeNS: en
After removing the attributeNS: null
```

Retrieve header infomation of a resource file

Following example demonstrates how to retrieve the header information of a resource file, using the method ***getAllResponseHeaders()*** using the property ***readyState***.

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html; charset=iso-8859-2" />
         <script>
            function loadXMLDoc() {
               var xmlHttp = null;

               if(window.XMLHttpRequest) // for Firefox, IE7+, Opera, Safari, ... {
                   xmlHttp = new XMLHttpRequest();
               } else if(window.ActiveXObject) //  for Internet Explorer 5 or 6 {
                   xmlHttp = new ActiveXObject("Microsoft.XMLHTTP");
               }

                return xmlHttp;
            }

            function makerequest(serverPage, myDiv) {
               var request =  loadXMLDoc();
               request.open("GET", serverPage);
               request.send(null);
               request.onreadystatechange = function() {
                  if (request.readyState == 4) {
                     document.getElementById(myDiv).innerHTML = request.getAllResponseHeaders();
                  }
                }
            }
      </script>
   </head>
   <body>
      <button type = "button" onclick = "makerequest('/dom/node.xml', 'ID')">
         Click me to load the AllResponseHeaders</button>
      <div id = "ID"></div>
   </body>
</html>
```

Execution

Save this file as *http_allheader.html* on the server path (this file and node.xml should be on the same path in your server). We will get the output as shown below (depends on the browser) –

```
Date: Sat, 27 Sep 2014 07:48:07 GMT Server: Apache Last-Modified:
      Wed, 03 Sep 2014 06:35:30 GMT Etag: "464bf9-2af-50223713b8a60" Accept-Ranges: bytes Vary: Accept-En
      Content-Encoding: gzip Content-Length: 256 Content-Type: text/xml
```

DOM - DOMException Object

The *DOMException* represents an abnormal event happening when a method or a property is used.

Properties

Below table lists the properties of the DOMException object

S.No.	Property & Description
1	**name** Returns a DOMString that contains one of the string associated with an error constant (as seen the table below).

Error Types

S.No.	Type & Description
1	**IndexSizeError** The index is not in the allowed range. For example, this can be thrown by the Range object. (Legacy code value: 1 and legacy constant name: INDEX_SIZE_ERR)
2	**HierarchyRequestError** The node tree hierarchy is not correct. (Legacy code value: 3 and legacy constant name: HIERARCHY_REQUEST_ERR)
3	**WrongDocumentError**

	The object is in the wrong document. (Legacy code value: 4 and legacy constant name: WRONG_DOCUMENT_ERR)
4	**InvalidCharacterError** The string contains invalid characters. (Legacy code value: 5 and legacy constant name: INVALID_CHARACTER_ERR)
5	**NoModificationAllowedError** The object cannot be modified. (Legacy code value: 7 and legacy constant name: NO_MODIFICATION_ALLOWED_ERR)
6	**NotFoundError** The object cannot be found here. (Legacy code value: 8 and legacy constant name: NOT_FOUND_ERR)
7	**NotSupportedError** The operation is not supported. (Legacy code value: 9 and legacy constant name: NOT_SUPPORTED_ERR)
8	**InvalidStateError** The object is in an invalid state. (Legacy code value: 11 and legacy constant name: INVALID_STATE_ERR)
9	**SyntaxError** The string did not match the expected pattern. (Legacy code value: 12 and legacy constant name: SYNTAX_ERR)
10	**InvalidModificationError** The object cannot be modified in this way. (Legacy code value: 13 and legacy constant name: INVALID_MODIFICATION_ERR)
11	**NamespaceError** The operation is not allowed by Namespaces in XML. (Legacy code value: 14 and legacy constant name: NAMESPACE_ERR)
12	**InvalidAccessError**

	The object does not support the operation or argument. (Legacy code value: 15 and legacy constant name: INVALID_ACCESS_ERR)
13	**TypeMismatchError** The type of the object does not match the expected type. (Legacy code value: 17 and legacy constant name: TYPE_MISMATCH_ERR) This value is deprecated, the JavaScript TypeError exception is now raised instead of a DOMException with this value.
14	**SecurityError** The operation is insecure. (Legacy code value: 18 and legacy constant name: SECURITY_ERR)
15	**NetworkError** A network error occurred. (Legacy code value: 19 and legacy constant name: NETWORK_ERR)
16	**AbortError** The operation was aborted. (Legacy code value: 20 and legacy constant name: ABORT_ERR)
17	**URLMismatchError** The given URL does not match another URL. (Legacy code value: 21 and legacy constant name: URL_MISMATCH_ERR)
18	**QuotaExceededError** The quota has been exceeded. (Legacy code value: 22 and legacy constant name: QUOTA_EXCEEDED_ERR)
19	**TimeoutError** The operation timed out. (Legacy code value: 23 and legacy constant name: TIMEOUT_ERR)
20	**InvalidNodeTypeError** The node is incorrect or has an incorrect ancestor for this operation. (Legacy code value: 24 and legacy constant name: INVALID_NODE_TYPE_ERR)

21	**DataCloneError** The object cannot be cloned. (Legacy code value: 25 and legacy constant name: DATA_CLONE_ERR)
22	**EncodingError** The encoding operation, being an encoding or a decoding one, failed (No legacy code value and constant name).
23	**NotReadableError** The input/output read operation failed (No legacy code value and constant name).

Example

Following example demonstrates how using a not well-formed XML document causes a DOMException.

error.xml contents are as below −

```
<?xml version = "1.0" encoding = "UTF-8" standalone = "no" ?>
<Company id = "companyid">
   <Employee category = "Technical" id = "firstelement" type = "text/html">
      <FirstName>Tanmay</first>
      <LastName>Patil</LastName>
      <ContactNo>1234567890</ContactNo>
      <Email>tanmaypatil@xyz.com</Email>
   </Employee>
</Company>
```

Following example demonstrates the usage of the *name* attribute −

```
<html>
   <head>
      <script>
         function loadXMLDoc(filename) {
            if (window.XMLHttpRequest) {
               xhttp = new XMLHttpRequest();
            } else // code for IE5 and IE6 {
               xhttp = new ActiveXObject("Microsoft.XMLHTTP");
            }
            xhttp.open("GET",filename,false);
            xhttp.send();
            return xhttp.responseXML;
         }
      </script>
   </head>
   <body>
      <script>
         try {
```

```
            xmlDoc = loadXMLDoc("/dom/error.xml");
            var node  = xmlDoc.getElementsByTagName("to").item(0);
            var refnode = node.nextSibling;
            var newnode = xmlDoc.createTextNode('That is why you fail.');
            node.insertBefore(newnode, refnode);
         } catch(err) {
            document.write(err.name);
         }
      </script>
   </body>
</html>
```

Execution

Save this file as *domexcption_name.html* on the server path (this file and error.xml should be on the same path in your server). We will get the output as shown below –

```
TypeError
```

Previous Page Next Page

Enter email for newsletter go