**Day 8**

**XML**

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

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

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

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

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

**<contact-info>**

**<company>TutorialsPoint**

**<contact-info>**

**</company>**

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

* XML declaration
* Document type declaration

An XML declaration should abide with the following rules −

* If the XML declaration is present in the XML, it must be placed as the first line in the XML document.
* If the XML declaration is included, it must contain version number attribute.
* The Parameter names and values are case-sensitive.
* The names are always in lower case.
* The order of placing the parameters is important. The correct order is: *version, encoding and standalone.*
* Either single or double quotes may be used.
* The XML declaration has no closing tag i.e. **</?xml>**

# XML Attributes

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

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

**<!DOCTYPE garden [**

**<!ELEMENT garden (plants)\*>**

**<!ELEMENT plants (#PCDATA)>**

**<!ATTLIST plants category CDATA #REQUIRED>**

**]>**

**<garden>**

**<plants category = "flowers" />**

**<plants category = "shrubs">**

**</plants>**

**</garden>**

**Element Attribute Rules**

Following are the rules that need to be followed for attributes −

* An attribute name must not appear more than once in the same start-tag or empty-element tag.
* An attribute must be declared in the Document Type Definition (DTD) using an Attribute-List Declaration.
* Attribute values must not contain direct or indirect entity references to external entities.
* The replacement text of any entity referred to directly or indirectly in an attribute value must not contain a less than sign (**<**)

# CDATA Sections

The term CDATA means, Character Data. CDATA is defined as blocks of text that are not parsed by the parser, but are otherwise recognized as markup.

 each character written inside the CDATA section is ignored by the parser.

**<script>**

**<![CDATA[**

**<message> Welcome to TutorialsPoint </message>**

**]] >**

**</script >**

# Validation

**Validation** is a process by which an XML document is validated. An XML document is said to be valid if its contents match with the elements, attributes and associated document type declaration(DTD), and if the document complies with the constraints expressed in it. Validation is dealt in two ways by the XML parser. They are −

* Well-formed XML document
* Valid XML document

Following is an example of a well-formed XML document −

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

**<!DOCTYPE address**

**[**

**<!ELEMENT address (name,company,phone)>**

**<!ELEMENT name (#PCDATA)>**

**<!ELEMENT company (#PCDATA)>**

**<!ELEMENT phone (#PCDATA)>**

**]>**

**<address>**

**<name>Tanmay Patil</name>**

**<company>TutorialsPoint</company>**

**<phone>(011) 123-4567</phone>**

**</address>**

**DTD**

If an XML document is well-formed and has an associated **Document Type Declaration** (DTD), then it is said to be a valid XML document.

* The XML Document Type Declaration, commonly known as DTD, is a way to describe XML language precisely.
* DTDs check vocabulary and validity of the structure of XML documents against grammatical rules of appropriate XML language.
* An XML DTD can be either specified inside the document, or it can be kept in a separate document and then liked separately.
* A DTD is referred to as an internal DTD if elements are declared within the XML files. To refer it as internal DTD, *standalone* attribute in XML declaration must be set to **yes**. This means, the declaration works independent of an external source.

**Address.xml**

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

**<!DOCTYPE address [**

**<!ELEMENT address (name,company,phone)>**

**<!ELEMENT name (#PCDATA)>**

**<!ELEMENT company (#PCDATA)>**

**<!ELEMENT phone (#PCDATA)>**

**]>**

**<address>**

**<name>Sagar Shinde</name>**

**<company>Transflower</company>**

**<phone>(011) 123-4567</phone>**

**</address>**

## External DTD

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

The following example shows external DTD usage

**Address.xml**

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

**<!DOCTYPE address SYSTEM "address.dtd">**

**<address>**

**<name>Sagar Shinde</name>**

**<company> Transflower </company>**

**<phone>(022) 434-6578</phone>**

**</address>**

**Address.dtd**

**<!ELEMENT address (name,company,phone)>**

**<!ELEMENT name (#PCDATA)>**

**<!ELEMENT company (#PCDATA)>**

**<!ELEMENT phone (#PCDATA)>**

# XML Schemas

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

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

**<xs:schema xmlns:xs = "http://www.w3.org/2001/XMLSchema">**

**<xs:element name = "contact">**

**<xs:complexType>**

**<xs:sequence>**

**<xs:element name = "name" type = "xs:string" />**

**<xs:element name = "company" type = "xs:string" />**

**<xs:element name = "phone" type = "xs:int" />**

**</xs:sequence>**

**</xs:complexType>**

**</xs:element>**

**</xs:schema>**

# Tree Structure

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

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

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

**<Company>**

**<Employee>**

**<FirstName>Jai</FirstName>**

**<LastName>Kiran</LastName>**

**<ContactNo>9876956431</ContactNo>**

**<Email>jai.kiran@transflower.in</Email>**

**<Address>**

**<City>Pune</City>**

**<State>Maharashtra</State>**

**<Zip>411022</Zip>**

**</Address>**

**</Employee>**

**</Company>**

# DOM

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

* A DOM document is a collection of nodes or pieces of information organized in a hierarchy. This hierarchy allows a developer to navigate through the tree looking for specific information. Because it is based on a hierarchy of information, the DOM is said to be *tree based*.
* The XML DOM, on the other hand, also provides an API that allows a developer to add, edit, move, or remove nodes in the tree at any point in order to create an application.

**<!DOCTYPE html>**

**<html>**

**<body>**

**<h1>DOM example </h1>**

**<div>**

**<b>Name:</b> <span id = "name"></span><br>**

**<b>Company:</b> <span id = "company"></span><br>**

**<b>Phone:</b> <span id = "phone"></span>**

**</div>**

**<script>**

**if (window.XMLHttpRequest)**

**{// code for IE7+, Firefox, Chrome, Opera, Safari**

**xmlhttp = new XMLHttpRequest();**

**}**

**else**

**{// code for IE6, IE5**

**xmlhttp = new ActiveXObject("Microsoft.XMLHTTP");**

**}**

**xmlhttp.open("GET","/xml/address.xml",false);**

**xmlhttp.send();**

**xmlDoc = xmlhttp.responseXML;**

**document.getElementById("name").innerHTML=**

**xmlDoc.getElementsByTagName("name")[0].childNodes[0].nodeValue;**

**document.getElementById("company").innerHTML=**

**xmlDoc.getElementsByTagName("company")[0].childNodes[0].nodeValue;**

**document.getElementById("phone").innerHTML=**

**xmlDoc.getElementsByTagName("phone")[0].childNodes[0].nodeValue;**

**</script>**

**</body>**

**</html>**

## Loading the XML Document

**var xmlhttp = new XMLHttpRequest();**

## Selecting Nodes

Unfortunately, there are different ways of dealing with XPath in different browsers. Chrome, Firefox, Edge, Opera, and Safari use the evaluate() method to select nodes:

**xmlDoc.evaluate(xpath, xmlDoc, null, XPathResult.ANY\_TYPE,null);**

Internet Explorer uses the selectNodes() method to select node:

**xmlDoc.selectNodes(xpath);**

**<html>**

**<body>**

**<h1>XML DOM</h1>**

**<p id="demo"></p>**

**<script>**

**var xhttp = new XMLHttpRequest();**

**xhttp.onreadystatechange = function() {**

**if (this.readyState == 4 && this.status == 200) {**

**showResult(this);**

**}**

**};**

**xhttp.open("GET", "books.xml", true);**

**xhttp.send();**

**function showResult(xml) {**

**var xmlDoc = xml.responseXML;**

**var x = xmlDoc.getElementsByTagName('title')[0];**

**var y = x.childNodes[0];**

**document.getElementById("demo").innerHTML =**

**y.nodeValue;**

**}**

**</script>**

**</body>**

**</html>**

**XPath**

* XPath stands for XML Path Language
* XPath uses "path like" syntax to identify and navigate nodes in an XML document
* XPath contains over 200 built-in functions
* XPath is a major element in the XSLT standard
* XPath is a W3C recommendation

**XPath Axes**

An axis represents a relationship to the context (current) node,

and is used to locate nodes relative to that node on the tree.

**AxisName**  **Result**

ancestor Selects all ancestors (parent, grandparent, etc.) of the current node

ancestor-or-self Selects all ancestors (parent, grandparent, etc.) of the current node and the current node itself

attribute Selects all attributes of the current node

child Selects all children of the current node

descendant Selects all descendants (children, grandchildren, etc.) of the current node

descendant-or-self Selects all descendants (children, grandchildren, etc.) of the current node and the current node

itself

following Selects everything in the document after the closing tag of the current node

following-sibling Selects all siblings after the current node

namespace Selects all namespace nodes of the current node

parent Selects the parent of the current node

preceding Selects all nodes that appear before the current node in the document, except ancestors, attribute nodes and namespace nodes

preceding-sibling Selects all siblings before the current node

self Selects the current node

**Location Path Expression**

A location path can be absolute or relative.

**Example Result**

child::book Selects all book nodes that are children of the current node

attribute::lang Selects the lang attribute of the current node

child::\* Selects all element children of the current node

attribute::\* Selects all attributes of the current node

child::text() Selects all text node children of the current node

child::node() Selects all children of the current node

descendant::book Selects all book descendants of the current node

ancestor::book Selects all book ancestors of the current node

ancestor-or-self::book Selects all book ancestors of the current node - and the current as well if it is a book node

child::\*/child::price Selects all price grandchildren of the current node

**XPath Expression Result**

/bookstore/book[1] Selects the first book element that is the child of the bookstore element

/bookstore/book[last()] Selects the last book element that is the child of the bookstore element

/bookstore/book[last()-1] Selects the last but one book element that is the child of the bookstore element

/bookstore/book[position()<3] Selects the first two book elements that are children of the bookstore element

//title[@lang] Selects all the title elements that have an attribute named lang

//title[@lang='en'] Selects all the title elements that have a "lang" attribute with a value of "en"

/bookstore/book[price>35.00] Selects all the book elements of the bookstore element that have a price element with a value greater than 35.00

/bookstore/book[price>35.00]/title Selects all the title elements of the book elements of the bookstore element that have a price element with a value greater than 35.00

**<!DOCTYPE html>**

**<html>**

**<body>**

**<p id="demo"></p>**

**<script>**

**var xhttp = new XMLHttpRequest();**

**xhttp.onreadystatechange = function() {**

**if (this.readyState == 4 && this.status == 200) {**

**showResult(xhttp.responseXML);**

**}**

**};**

**xhttp.open("GET", "books.xml", true);**

**xhttp.send();**

**function showResult(xml) {**

**var txt = "";**

**// path = "/bookstore/book/title"**

**// path = "/bookstore/book[1]/title";**

**// path = "/bookstore/book/price[text()]";**

**// path = "/bookstore/book[price>35]/price";**

**path = "/bookstore/book[price>35]/title";**

**if (xml.evaluate) {**

**var nodes = xml.evaluate(path, xml, null, XPathResult.ANY\_TYPE, null);**

**var result = nodes.iterateNext();**

**while (result) {**

**txt += result.childNodes[0].nodeValue + "<br>";**

**result = nodes.iterateNext();**

**}**

**// Code For Internet Explorer**

**} else if (window.ActiveXObject || xhttp.responseType == "msxml-document") {**

**xml.setProperty("SelectionLanguage", "XPath");**

**nodes = xml.selectNodes(path);**

**for (i = 0; i < nodes.length; i++) {**

**txt += nodes[i].childNodes[0].nodeValue + "<br>";**

**}**

**}**

**document.getElementById("demo").innerHTML = txt;**

**}**

**</script>**

**</body>**

**JSON**

JSON - Javascript Object Notation. JSON is a syntax for storing and exchanging data.It is text written with Javascript object notation.

## Sending Data

If you have data stored in a JavaScript object, you can convert the object into JSON, and send it to a server:

**var obj = {name: "Sameer", age: 43, city: "Pune"};  
var json = JSON.stringify(obj);  
window.location = "customer.php?person=" + json;**

## Receiving Data

If you receive data in JSON format, you can convert it into a JavaScript object:

**var json = '{"name":"Sameer", "age":43, "city":"Pune"}';  
var obj = JSON.parse(json);  
document.getElementById("demo").innerHTML = obj.name;**

## Storing Data

When storing data, the data has to be a certain format, and regardless of where you choose to store it, text is always one of the legal formats. JSON makes it possible to store JavaScript objects as text.

**// Storing data:**

**obj = {name: "Samee", age: 43, city: "Pune"};  
json = JSON.stringify(obj);  
localStorage.setItem("tfl", json);  
  
// Retrieving data:**

**text = localStorage.getItem("tfl");  
obj = JSON.parse(text);  
document.getElementById("demo").innerHTML = obj.name;**

When exchanging data between a browser and a server, the data can only be text.

JSON is text, and we can convert any JavaScript object into JSON, and send JSON to the server.

We can also convert any JSON received from the server into JavaScript objects.

This way we can work with the data as JavaScript objects, with no complicated parsing and translations.

JavaScript arrays can also be used as JSON.

**{"flowers":[  
  { "title":"Gerbera", "description":"Wedding Flower" },  
  { "title":"Carnation", "description":"Worship Flower" },  
  { "title":"Jasmine", "description":"Smelling Flower" }  
]}**

**JSON Files**

* The file type for JSON files is ".json"
* The MIME type for JSON text is "application/json"

## Why JSON is Better Than XML

* XML is much more difficult to parse than JSON.
* JSON is parsed into a ready-to-use JavaScript object.

PHP has some built-in functions to handle JSON.

Objects in PHP can be converted into JSON by using the PHP function json\_encode():

**<?php  
$obj->name = "Ravi Tambade";  
$obj ->age = 43;**

**$obj ->mobile = "9881735801";**

**$obj ->email = "ravi.tambade@transflower.in";**

**$obj ->company = "Transflower Learning Pvt. Ltd.";  
$obj ->city = "Pune";  
  
$objjson = json\_encode($obj);  
  
echo $objjson;  
?>**

**What is AJAX?**

AJAX = **A**synchronous **J**avaScript **A**nd **X**ML.

AJAX is not a programming language.

AJAX just uses a combination of:

* A browser built-in XMLHttpRequest object (to request data from a web server)
* JavaScript and HTML DOM (to display or use the data)

Here is a JavaScript on the client, using an AJAX call to request the PHP file from the example above:

**var xmlhttp = new XMLHttpRequest();  
xmlhttp.onreadystatechange = function() {  
  if (this.readyState == 4 && this.status == 200) {  
    var myObj = JSON.parse(this.responseText);  
    document.getElementById("demo").innerHTML = myObj.name;  
  }  
};  
xmlhttp.open("GET", "demo.php", true);  
xmlhttp.send();**

**AJAX**

* Update a web page without reloading the page
* Request data from a server - after the page has loaded
* Receive data from a server - after the page has loaded
* Send data to a server - in the background

**Steps in AJAX call**

1. An event occurs in a web page (the page is loaded, a button is clicked)

2. An XMLHttpRequest object is created by JavaScript

3. The XMLHttpRequest object sends a request to a web server

4. The server processes the request

5. The server sends a response back to the web page

6. The response is read by JavaScript

7. Proper action (like page update) is performed by JavaScript

All modern browsers support the XMLHttpRequest object.

The XMLHttpRequest object can be used to exchange data with a server behind the scenes. This means that it is possible to update parts of a web page, without reloading the whole page.