**Object of javascript :** [**https://www.javatpoint.com/javascript-objects**](https://www.javatpoint.com/javascript-objects)

**What is xml**

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

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

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

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

## **XML Usage**

A short list of XML usage says it all:

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

## **What is Markup?**

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

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

<message>

<text>Hello, world!</text>

</message>

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

## **Is XML a Programming Language?**

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

**What is DTD**

**Follow this url for this**

[**https://www.w3schools.com/xml/xml\_dtd\_intro.asp**](https://www.w3schools.com/xml/xml_dtd_intro.asp)

**XML:- document type definition :-** [**https://www.tutorialspoint.com/xml/xml\_dtds.htm**](https://www.tutorialspoint.com/xml/xml_dtds.htm)

DTDs have the following benefits:

* DTD support is ubiquitous due to its inclusion in the XML 1.0 standard.
* DTDs are terse compared to element-based schema languages and consequently present more information in a single screen.
* DTDs allow the declaration of [standard public entity sets](https://en.wikipedia.org/wiki/SGML_entity) for publishing characters.
* DTDs define a *document type* rather than the types used by a namespace, thus grouping all constraints for a document in a single collection.

DTDs have the following limitations:

* They have no explicit support for newer [features](https://en.wikipedia.org/wiki/Feature_(software_design)) of XML, most importantly [namespaces](https://en.wikipedia.org/wiki/XML_Namespace).
* They lack expressiveness. XML DTDs are simpler than SGML DTDs and there are certain structures that cannot be expressed with regular grammars. DTDs only support rudimentary datatypes.
* They lack readability. DTD designers typically make heavy use of parameter entities (which behave essentially as textual [macros](https://en.wikipedia.org/wiki/Macro_(computer_science))), which make it easier to define complex grammars, but at the expense of clarity.

**XML SCHEMAS (**[**https://www.tutorialspoint.com/xml/xml\_schemas.htm**](https://www.tutorialspoint.com/xml/xml_schemas.htm)**)**

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

## **Syntax**

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

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

### Example

The following example shows how to use schema:

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

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

<xs:element name="contact">

<xs:complexType>

<xs:sequence>

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

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

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

</xs:sequence>

</xs:complexType>

</xs:element>

</xs:schema>

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

## **Elements**

As we saw in the [XML - Elements](https://www.tutorialspoint.com/xml/xml_elements.htm) chapter, elements are the building blocks of XML document. An element can be defined within an XSD as follows:

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

## **Definition Types**

You can define XML schema elements in following ways:

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

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

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

<xs:element name="Address">

<xs:complexType>

<xs:sequence>

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

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

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

</xs:sequence>

</xs:complexType>

</xs:element>

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

**Global Types -**With global type, you can define a single type in your document, which can be used by all other references. For example, suppose you want to generalize the *person* and *company* for different addresses of the company. In such case, you can define a general type as below:

<xs:element name="AddressType">

<xs:complexType>

<xs:sequence>

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

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

</xs:sequence>

</xs:complexType>

</xs:element>

Now let us use this type in our example as below:

<xs:element name="Address1">

<xs:complexType>

<xs:sequence>

<xs:element name="address" type="AddressType" />

<xs:element name="phone1" type="xs:int" />

</xs:sequence>

</xs:complexType>

</xs:element>

<xs:element name="Address2">

<xs:complexType>

<xs:sequence>

<xs:element name="address" type="AddressType" />

<xs:element name="phone2" type="xs:int" />

</xs:sequence>

</xs:complexType>

</xs:element>

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

## **Attributes**

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

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

**What is DOM?**

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 Stands for **Document Object Model** and it represent an XML Document into tree format which each element representing tree branches.



The XML DOM defines a standard way for accessing and manipulating XML documents. It presents an XML document as a tree-structure.

## **The XML DOM**

All XML elements can be accessed through the XML DOM.

The XML DOM is:

* A standard object model for XML
* A standard programming interface for XML
* Platform- and language-independent
* A W3C standard

In other words: **The XML DOM is a standard for how to get, change, add, or delete XML elements.**

**Properties of DOM**

Programmers can build documents, navigate their structure, and add, modify, or delete elements and content.

Provides a standard programming interface that can be used in a wide variety of environments and applications.

## **Get the Value of an XML Element**

This code retrieves the text value of the first <title> element in an XML document:

### Example

txt = xmlDoc.getElementsByTagName("title")[0].childNodes[0].nodeValue;

* **xmlDoc** - the XML DOM object created by the parser.
* **getElementsByTagName("title")[0]** - get the first <title> element
* **childNodes[0]** - the first child of the <title> element (the text node)
* **nodeValue** - the value of the node (the text itself)

## **XML DOM Properties**

These are some typical DOM properties:

* x.nodeName - the name of x
* x.nodeValue - the value of x
* x.parentNode - the parent node of x
* x.childNodes - the child nodes of x
* x.attributes - the attributes nodes of x

Note: In the list above, x is a node object.

## **XML DOM Methods**

* x.getElementsByTagName(*name*) - get all elements with a specified tag name
* x.appendChild(*node*) - insert a child node to x
* x.removeChild(*node*) - remove a child node from x

Note: In the list above, x is a node object.

**Applet**

Applet is a special type of program that is embedded in the webpage to generate the dynamic content. It runs inside the browser and works at client side.

An **applet** is a Java program that runs in a Web browser. An applet can be a fully functional Java application because it has the entire Java API at its disposal.

**There are some important differences between an applet and a standalone Java application, including the following** −

* An applet is a Java class that extends the java.applet.Applet class.
* A main() method is not invoked on an applet, and an applet class will not define main().
* Applets are designed to be embedded within an HTML page.
* When a user views an HTML page that contains an applet, the code for the applet is downloaded to the user's machine.
* A JVM is required to view an applet. The JVM can be either a plug-in of the Web browser or a separate runtime environment.
* The JVM on the user's machine creates an instance of the applet class and invokes various methods during the applet's lifetime.
* Applets have strict security rules that are enforced by the Web browser. The security of an applet is often referred to as sandbox security, comparing the applet to a child playing in a sandbox with various rules that must be followed.
* Other classes that the applet needs can be downloaded in a single Java Archive (JAR) file.

Advantage of Applet

There are many advantages of applet. They are as follows:

* It works at client side so less response time.
* Secured
* It can be executed by browsers running under many plateforms, including Linux, Windows, Mac Os etc.

Drawback of Applet

* Plugin is required at client browser to execute applet.

### Hierarchy of Applet

### hierarchy of applet

### Lifecycle of Java Applet

1. Applet is initialized.
2. Applet is started.
3. Applet is painted.
4. Applet is stopped.
5. Applet is destroyed.

Lifecycle methods for Applet:

The java.applet.Applet class 4 life cycle methods and java.awt.Component class provides 1 life cycle methods for an applet.

java.applet.Applet class

For creating any applet java.applet.Applet class must be inherited. It provides 4 life cycle methods of applet.

1. **public void init():** is used to initialized the Applet. It is invoked only once.
2. **public void start():** is invoked after the init() method or browser is maximized. It is used to start the Applet.
3. **public void stop():** is used to stop the Applet. It is invoked when Applet is stop or browser is minimized.
4. **public void destroy():** is used to destroy the Applet. It is invoked only once.

java.awt.Component class

The Component class provides 1 life cycle method of applet.

1. **public void paint(Graphics g):** is used to paint the Applet. It provides Graphics class object that can be used for drawing oval, rectangle, arc etc.

How to run an Applet?

There are two ways to run an applet

1. By html file.
2. By appletViewer tool (for testing purpose).

Simple example of Applet by html file:

To execute the applet by html file, create an applet and compile it. After that create an html file and place the applet code in html file. Now click the html file.

1. //First.java
2. **import** java.applet.Applet;
3. **import** java.awt.Graphics;
4. **public** **class** First **extends** Applet{
6. **public** **void** paint(Graphics g){
7. g.drawString("welcome",150,150);
8. }
10. }

myapplet.html

1. <html>
2. <body>
3. <applet code="First.class" width="300" height="300">
4. </applet>
5. </body>
6. </html>

## **The Applet Class**

Every applet is an extension of the *java.applet.Applet class*. The base Applet class provides methods that a derived Applet class may call to obtain information and services from the browser context.

These include methods that do the following −

* Get applet parameters
* Get the network location of the HTML file that contains the applet
* Get the network location of the applet class directory
* Print a status message in the browser
* Fetch an image
* Fetch an audio clip
* Play an audio clip
* Resize the applet

**Follow this below link for applet class**

[**http://www.jwrider.com/riderist/java/classapplet.htm**](http://www.jwrider.com/riderist/java/classapplet.htm)

# Event Handling in Applet

|  |
| --- |
| As we perform event handling in AWT or Swing, we can perform it in applet also. Let's see the simple example of event handling in applet that prints a message by click on the button. |

Applets inherit a group of event-handling methods from the Container class. The Container class defines several methods, such as processKeyEvent and processMouseEvent, for handling particular types of events, and then one catch-all method called processEvent.

## **ava Event classes and Listener interfaces**

|  |  |
| --- | --- |
| **Event Classes** | **Listener Interfaces** |
| ActionEvent | ActionListener |
| MouseEvent | MouseListener and MouseMotionListener |
| MouseWheelEvent | MouseWheelListener |
| KeyEvent | KeyListener |
| ItemEvent | ItemListener |
| TextEvent | TextListener |
| AdjustmentEvent | AdjustmentListener |
| WindowEvent | WindowListener |
| ComponentEvent | ComponentListener |
| ContainerEvent | ContainerListener |
| FocusEvent | FocusListener |

## **Steps to perform Event Handling**

Following steps are required to perform event handling:

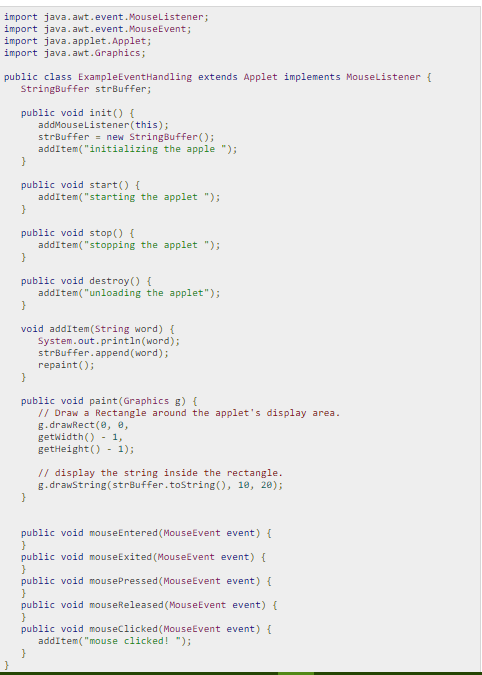
1. Register the component with the Listener

## **Registration Methods**

For registering the component with the Listener, many classes provide the registration methods. For example:

* **Button**
  + public void addActionListener(ActionListener a){}
* **MenuItem**
  + public void addActionListener(ActionListener a){}
* **TextField**
  + public void addActionListener(ActionListener a){}
  + public void addTextListener(TextListener a){}
* **TextArea**
  + public void addTextListener(TextListener a){}
* **Checkbox**
  + public void addItemListener(ItemListener a){}
* **Choice**
  + public void addItemListener(ItemListener a){}
* **List**
  + public void addItemListener(ItemListener a){}

Example of mouse event handling



**Above example url**

[**https://www.tutorialspoint.com/java/java\_applet\_basics.htm**](https://www.tutorialspoint.com/java/java_applet_basics.htm)

**An another example of event handling that is on button click event**

**import** java.applet.Applet;

**import** java.awt.Button;

**import** java.awt.TextField;

**import** java.awt.event.ActionEvent;

**import** java.awt.event.ActionListener;

**public** **class** applet\_btn\_event **extends** Applet **implements** ActionListener {

Button b;

TextField tf;

StringBuffer strBuffer;

**public** **void** init() {

strBuffer = **new** StringBuffer();

addItem("initializing the apple ");

tf = **new** TextField();

tf.setBounds(30, 40, 150, 20);

b = **new** Button("Click");

b.setBounds(80, 150, 60, 50);

add(b);

add(tf);

b.addActionListener(**this**);

setLayout(**null**);

}

**public** **void** start() {

addItem("starting the applet ");

}

**public** **void** stop() {

addItem("stopping the applet ");

}

**public** **void** destroy() {

addItem("unloading the applet");

}

**void** addItem(String word) {

System.***out***.println(word);

strBuffer.append(word);

repaint();

}

**public** **void** actionPerformed(ActionEvent e) {

tf.setText("Welcome");

}

}

**AWT:-** [**https://www.javatpoint.com/java-awt**](https://www.javatpoint.com/java-awt)