1. JavaScript variable

Variables are referred as named containers for storing information. We can place data into these containers and then refer to the data simply by naming the container.

Rules to declare variable in JavaScript

Here are the important rules that must be followed while declaring a variable in JavaScript.

- In JavaScript variable names are case sensitive i.e. a is different from A.
- Variable name can only be started with a underscore (_) or a letter (from a to z or A to Z), or dollar ($) sign.
- Numbers (0 to 9) can only be used after a letter.
- No other special character is allowed in variable name.

Before you use a variable in a JavaScript program, you must declare it. Variables are declared with the var keyword as follows −

```
<script type="text/javascript">
   <!--
      var money;
      var name, age;
   //-->
</script>
```

Variables can be initialized at time of declaration or after declaration as follows −

```
<script type="text/javascript">
   <!--
      var name = "Ali";
      var money;
      money = 2000.50;
   //-->
</script>
```

JavaScript Operators

Operators are used to perform operation on one, two or more operands. Operator is represented by a symbol such as +, =, *, % etc. Following are the operators supported by javascript −

- Arithmetic Operators
- Comparison Operators
- Logical (or Relational) Operators
- Assignment Operators
- Conditional (or ternary) Operators
- Arithmetic Operators

Arithmatic Operators

Following table shows all the arithmetic operators supported by javascript –

Operator	Description	Example
+	Add two operands.	10 + 10 will give 20
-	Subtract second operand from the first.	10 – 10 will give 0
*	Multiply two operands.	10 * 30 will give 300
/	Divide numerator by denominator	10/10 will give 1
%	It is called modulus operator and gives remainder of the division.	10 % 10 will give 0
++	Increment operator, increases integer value by one	10 ++ will give 11
--	Decrement operator, decreases integer value by one	10 – will give 9

Comparison Operators

Following table shows all the comparison operators supported by javascript –

Operator	Description	Example
==	Checks if values of two operands are equal or not, If yes then condition becomes true.	10 == 10 will give true
!=	**Not Equal to operator** Checks if the value of two operands is equal or not, if values are not equal then condition becomes true.	10 !=10 will give false
>	**Greater Than operator** Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	20 > 10 will give true
<	**Less than operator** Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	10 < 20 will give true
>=	**Greater than or equal to operator** Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	10 >=20 will give false
<=	**Less than or equal to operator** Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	10 <=20 will give true.

Logical Operators

Following table shows all the logical operators supported by javascript –

Operator	Description	Example
&&	Logical **AND** operator returns true if both operands are non zero.	10 && 10 will give true.
||	Logical **OR** operator returns true If any of the operand is non zero	10 || 0 will give true.
!	Logical **NOT** operator complements the logical state of its operand.	! (10 && 10) will give false.

Assignment Operators

Following table shows all the assignment operators supported by javascript –

Operator	Description	Example
=	**Simple Assignment operator** Assigns values from right side operands to left side operand.	C = A + B will assign value of A + B into C
+=	**Add AND assignment operator** It adds right operand to the left operand and assign the result to left operand	C += A is equivalent to C = C + A
-=	**Subtract AND assignment operator** It subtracts right operand from the left operand and assign the result to left operand	C -= A is equivalent to C = C - A
*=	**Multiply AND assignment operator** It multiplies right operand with the left operand and assign the result to left operand	C *= A is equivalent to C = C * A
/=	**Divide AND assignment operator** It divides left operand with the right operand and assign the result to left operand	C /= A is equivalent to C = C / A
%=	**Modulus AND assignment operator** Modulus AND assignment operator, It takes modulus using two operands and assign the result to left operand	C %= A is equivalent to C = C % A

Conditional Operator

It is also called ternary operator, since it has three operands.

Operator	Description	Example
?:	Conditional Expression	If Condition is true? Then value X : Otherwise value Y

Control Structure

Control structure actually controls the flow of execution of a program. Following are the several control structure supported by javascript.

- if ... else
- switch case
- do while loop
- while loop
- for loop

If ... else

The if statement is the fundamental control statement that allows JavaScript to make decisions and execute statements conditionally.

Syntax

```
if (expression){
   Statement(s) to be executed if expression is true
}
```

Example

```
<script type="text/javascript">
   <!--
      var age = 20;
      if( age > 18 ){
         document.write("<b>Qualifies for driving</b>");
      }
   //-->
</script>
```

Switch case

The basic syntax of the switch statement is to give an expression to evaluate and several different statements to execute based on the value of the expression. The interpreter checks each case against the value of the expression until a match is found. If nothing matches, a default condition will be used.

Syntax

```
switch (expression) {
   case condition 1: statement(s)
                     break;
   case condition 2: statement(s)
                     break;
   ...
   case condition n: statement(s)
                     break;
   default: statement(s)
}
```

Example

```
<script type="text/javascript">
   <!--
      var grade='A';
      document.write("Entering switch block<br/>");
      switch (grade) {
         case 'A': document.write("Good job<br/>");
            break;
         case 'B': document.write("Pretty good<br/>");
            break;
         case 'C': document.write("Passed<br/>");
            break;
         case 'D': document.write("Not so good<br/>");
            break;
         case 'F': document.write("Failed<br/>");
            break;
         default:  document.write("Unknown grade<br/>")
      }
      document.write("Exiting switch block");
   //-->
</script>
```

Do while Loop

The do...while loop is similar to the while loop except that the condition check happens at the end of the loop. This means that the loop will always be executed at least once, even if the condition is false.

Syntax

```
do{
   Statement(s) to be executed;
} while (expression);
```

Example

```
<script type="text/javascript">
   <!--
      var count = 0;
      document.write("Starting Loop" + "<br/>");
      do{
         document.write("Current Count : " + count + "<br/>");
         count++;
      }while (count < 0);
      document.write("Loop stopped!");
   //-->
</script>
```

This will produce following result –

```
Starting Loop
Current Count : 0
Loop stopped!
```

While Loop

The purpose of a while loop is to execute a statement or code block repeatedly as long as expression is true. Once expression becomes false, the loop will be exited.

Syntax

```
while (expression){
   Statement(s) to be executed if expression is true
}
```

Example

```
<script type="text/javascript">
   <!--
      var count = 0;
      document.write("Starting Loop" + "<br/>");
      while (count < 10){
         document.write("Current Count : " + count + "<br/>");
         count++;
      }
      document.write("Loop stopped!");
   //-->
</script>
```

This will produce following result –

```
Starting Loop
Current Count : 0
Current Count : 1
Current Count : 2
Current Count : 3
Current Count : 4
Current Count : 5
Current Count : 6
Current Count : 7
Current Count : 8
Current Count : 9
Loop stopped!
```

For Loop

The for loop is the most compact form of looping and includes the following three important parts –

- The loop initialization where we initialize our counter to a starting value. The initialization statement is executed before the loop begins.
- The test statement which will test if the given condition is true or not. If condition is true then code given inside the loop will be executed otherwise loop will come out.
- The iteration statement where you can increase or decrease your counter.

Syntax

```
for (initialization; test condition; iteration statement){
   Statement(s) to be executed if test condition is true
}
```

Example

```
<script type="text/javascript">
   <!--
      var count;
      document.write("Starting Loop" + "<br/>");
      for(count = 0; count < 10; count++){
         document.write("Current Count : " + count );
         document.write("<br/>");
      }
      document.write("Loop stopped!");
   //-->
</script>
```

This will produce following result which is similar to while loop −

```
Starting Loop
Current Count : 0
Current Count : 1
Current Count : 2
Current Count : 3
Current Count : 4
Current Count : 5
Current Count : 6
Current Count : 7
Current Count : 8
Current Count : 9
Loop stopped!
```

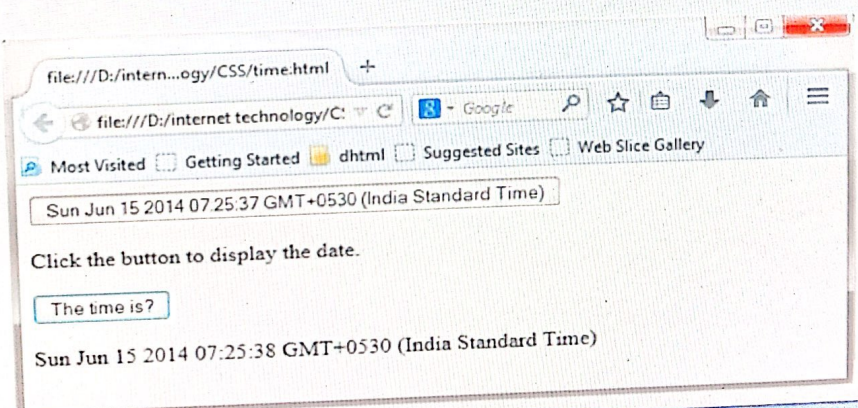
Creating Sample Program

Following is the sample program that shows time, when we click in button.

```
<html>
   <body>
      <button onclick="this.innerHTML=Date()">The time is?</button>
      <p>Click to display the date.</p>
      <button onclick="displayDate()">The time is?</button>
      <script>
         function displayDate() {
            document.getElementById("demo").innerHTML = Date();
         }</script>

         <p id="demo"></p>
      </script>
   </body>
</html>
```

Output



JavaScript Objects 2. 5.

‹ Previous

In JavaScript, objects are king. If you understand objects, you understand JavaScript.

In JavaScript, almost "everything" is an object.

- Booleans can be objects (if defined with the `new` keyword)
- Numbers can be objects (if defined with the `new` keyword)
- Strings can be objects (if defined with the `new` keyword)
- Dates are always objects
- Maths are always objects
- Regular expressions are always objects
- Arrays are always objects
- Functions are always objects
- Objects are always objects

All JavaScript values, except primitives, are objects.

JavaScript Primitives

A **primitive value** is a value that has no properties or methods.

3.14 is a primitive value

A **primitive data type** is data that has a primitive value.

JavaScript defines 7 types of primitive data types:

Examples

- `string`
- `number`
- `boolean`
- `null`
- `undefined`
- `symbol`
- `bigint`

Immutable

Primitive values are immutable (they are hardcoded and cannot be changed).

if x = 3.14, you can change the value of x, but you cannot change the value of 3.14.

Value	Type	Comment
"Hello"	string	"Hello" is always "Hello"
3.14	number	3.14 is always 3.14
true	boolean	true is always true
false	boolean	false is always false
null	null (object)	null is always null
undefined	undefined	undefined is always undefined

Objects are Variables

JavaScript variables can contain single values:

Example

```
let person = "John Doe";
```

Try it Yourself »

JavaScript variables can also contain many values.

Objects are variables too. But objects can contain many values.

Object values are written as **name : value** pairs (name and value separated by a colon).

Example

```
let person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};
```

Try it Yourself »

A JavaScript object is a collection of **named values**

It is a common practice to declare objects with the `const` keyword.

Example

```
const person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};
```

Object Properties

The named values, in JavaScript objects, are called **properties**.

Property	Value
firstName	John
lastName	Doe
age	50
eyeColor	blue

Objects written as name value pairs are similar to:

- Associative arrays in PHP
- Dictionaries in Python
- Hash tables in C
- Hash maps in Java
- Hashes in Ruby and Perl

Object Methods

Methods are **actions** that can be performed on objects.

Object properties can be both primitive values, other objects, and functions.

An **object method** is an object property containing a **function definition**.

Property	Value
firstName	John
lastName	Doe
age	50
eyeColor	blue
fullName	function() {return this.firstName + " " + this.lastName;}

JavaScript objects are containers for named values, called properties and methods.

Creating a JavaScript Object

With JavaScript, you can define and create your own objects.

There are different ways to create new objects:

- Create a single object, using an object literal.
- Create a single object, with the keyword `new`.
- Define an object constructor, and then create objects of the constructed type.
- Create an object using `Object.create()`.

Using an Object Literal

This is the easiest way to create a JavaScript Object.

Using an object literal, you both define and create an object in one statement.

An object literal is a list of name:value pairs (like age:50) inside curly braces {}.

The following example creates a new JavaScript object with four properties:

Example

```
const person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};
```

Try it Yourself »

Spaces and line breaks are not important. An object definition can span multiple lines:

Example

```
const person = {
  firstName: "John",
  lastName: "Doe",
  age: 50,
  eyeColor: "blue"
};
```

Try it Yourself »

This example creates an empty JavaScript object, and then adds 4 properties:

Example

```
const person = {};
person.firstName = "John";
person.lastName = "Doe";
person.age = 50;
person.eyeColor = "blue";
```

Using the JavaScript Keyword new

The following example create a new JavaScript object using `new Object()`, and then adds 4 properties:

Example

```
const person = new Object();
person.firstName = "John";
person.lastName = "Doe";
person.age = 50;
person.eyeColor = "blue";
```

Try it Yourself »

The examples above do exactly the same.

But there is no need to use `new Object()`.

For readability, simplicity and execution speed, use the object literal method.

JavaScript Objects are Mutable

Objects are mutable: They are addressed by reference, not by value.

If person is an object, the following statement will not create a copy of person:

```
const x = person;  // Will not create a copy of person.
```

The object x is **not a copy** of person. It **is** person. Both x and person are the same object.

Any changes to x will also change person, because x and person are the same object.

Example

```
const person = {
  firstName:"John",
  lastName:"Doe",
  age:50, eyeColor:"blue"
}

const x = person;
x.age = 10;       // Will change both x.age and person.age
```

3.
```
<html>
<head>
<title>javascript program</title>

var x, y, z;
x = parseInt(prompt("Enter x value:"));
y =parseInt(prompt("Enter y value:"));
document.write ("x = ");
document.write (x);
document.write (", y = ");
document.write (y);
document.write ("x + y is ");
z = x + y;
document.write (z);
document.write ("x – y is ");
z = x – y;
document.write (z);
document.write ("x * y is ");
z = x * y;
document.write (z);
document.write ("x / y is ");
z = x / y;
document.write (z);

</head>
<body>
</body>
</html>
```

Javascript Data Types 4.

JavaScript provides different **data types** to hold different types of values. There are two types of data types in JavaScript.

1. Primitive data type
2. Non-primitive (reference) data type

JavaScript is a **dynamic type language**, means you don't need to specify type of the variable because it is dynamically used by JavaScript engine. You need to use **var** here to specify the data type. It can hold any type of values such as numbers, strings etc. For example:

```
var a=40;//holding number
var b="Rahul";//holding string
```

JavaScript primitive data types

There are five types of primitive data types in JavaScript. They are as follows:

Data Type	Description
String	represents sequence of characters e.g. "hello"
Number	represents numeric values e.g. 100
Boolean	represents boolean value either false or true
Undefined	represents undefined value
Null	represents null i.e. no value at all

JavaScript non-primitive data types

The non-primitive data types are as follows:

Data Type	Description
Object	represents instance through which we can access members
Array	represents group of similar values
RegExp	represents regular expression

Scoping Rules

Outside of a function or object, variables are within the global space whether explicitly defined with var or not. Within a function or object, if the var statement is used, the defined variable will be local to the construct; without the statement, it will be global.

Commonly, JavaScript developers make assumptions about scoping rules with var that aren't quite true. For example, a var statement found within a for loop does not scope that value to the loop. In this case, it is scoped to either the function it is within or to the global space if it is outside a function or an object.

Just to see what happens consequently, I coded like this,

```
<script type="text/javascript">
    var global1 = true;
    global2 = true;

    function myFunc(){
        var local1 = "locals only"
        global3 = true;

    }

    // j is global here
    for(var j = 0; j < 10; j++){
        document.write("<br>"+j);
    }

    function myFunc(){
        var local1 = "locals only"
        global3 = false;

        // jLocal is within a function , hence local
        for(var jLocal = 0; jLocal < 10; jLocal++){
            document.write("<br>"+j);
        }
    }

    jLocal = jLocal + j;
    jdocument.write("<br>"+"value of jLocal: "+jLocal)
    myFunc();

</script>
```

j

JSON : JSON - ecma5
j : Number - Global
jLocal - Global

6

Because at the bottom of your code you have:

```
...
jLocal = jLocal + j; // defined not in any functions
...
```

Making it global, but not necessary defined.

2

It isn't the case of a local function. myFunc is global, just as the variable jLocal is (albeit the name). Because of hoisting, jLocal is assumed to be declared on top of parent scope.

Looking more carefully, there's two variable's named jLocal. One local to myFunc and an implicit one on global scope.

Want a tip?

Put `"use strict";` just before `var global1 = true;`. An HTML 5 implementation would be able to catch and show your error.

JavaScript Forms 5.

‹ Previous Next ›

JavaScript Form Validation

HTML form validation can be done by JavaScript.

If a form field (fname) is empty, this function alerts a message, and returns false, to prevent the form from being submitted:

JavaScript Example

```
function validateForm() {
  let x = document.forms["myForm"]["fname"].value;
  if (x == "") {
    alert("Name must be filled out");
    return false;
  }
}
```

The function can be called when the form is submitted:

HTML Form Example

```
<form name="myForm" action="/action_page.php" onsubmit="return validateForm()" method="post">
Name: <input type="text" name="fname">
<input type="submit" value="Submit">
</form>
```

Try it Yourself »

JavaScript Can Validate Numeric Input

JavaScript is often used to validate numeric input:

Please input a number between 1 and 10

Submit

Try it Yourself »

Automatic HTML Form Validation

HTML form validation can be performed automatically by the browser:

If a form field (fname) is empty, the `required` attribute prevents this form from being submitted:

HTML Form Example

```
<form action="/action_page.php" method="post">
  <input type="text" name="fname" required>
  <input type="submit" value="Submit">
</form>
```

Try it Yourself »

Automatic HTML form validation does not work in Internet Explorer 9 or earlier.

Data Validation

Data validation is the process of ensuring that user input is clean, correct, and useful.

Typical validation tasks are:

- has the user filled in all required fields?
- has the user entered a valid date?
- has the user entered text in a numeric field?

Most often, the purpose of data validation is to ensure correct user input.

Validation can be defined by many different methods, and deployed in many different ways.

Server side validation is performed by a web server, after input has been sent to the server.

Client side validation is performed by a web browser, before input is sent to a web server.

HTML Constraint Validation

HTML5 introduced a new HTML validation concept called **constraint validation**.

HTML constraint validation is based on:

- Constraint validation **HTML Input Attributes**
- Constraint validation **CSS Pseudo Selectors**
- Constraint validation **DOM Properties and Methods**

Constraint Validation HTML Input Attributes

Attribute	Description
disabled	Specifies that the input element should be disabled
max	Specifies the maximum value of an input element
min	Specifies the minimum value of an input element
pattern	Specifies the value pattern of an input element
required	Specifies that the input field requires an element
type	Specifies the type of an input element

For a full list, go to HTML Input Attributes.

Constraint Validation CSS Pseudo Selectors

Selector	Description
:disabled	Selects input elements with the "disabled" attribute specified
:invalid	Selects input elements with invalid values
:optional	Selects input elements with no "required" attribute specified
:required	Selects input elements with the "required" attribute specified
:valid	Selects input elements with valid values

For a full list, go to CSS Pseudo Classes.

6.

2.10 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.

Syntax

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

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 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 the following ways –

2.10.1 Simple Type

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

```
<xs:element name = "phone_number" type = "xs:int" />
```

2.10.2 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 the 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 follows –

```
<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 follows –

```
<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"/>
```

technologies.

Application of XML Schema

Functionality of XML Schema was exploited in implement many features of the security module. Some of these features are:

WSDL Based Validation

We needed a mechanism to prevent invalid and corrupt SOAP requests from reaching the server and limit messages to the once defined WSDLs. To achieve this, we developed an application that converted given WSDL into an XML Schema which when applied on the SOAP messages reports messages that do not conform to the WSDL definition.

Web Services Access Control

This feature limits a user's access to particular services or operations defined in the WSDL file. In addition converting WSDL into a XML Schema, our schema runtime had the ability to register callbacks on any element in the XML Schema. This was used to implement ACLs for operations in the WSDL.

SQL and Command Injection Protection

Detect and block command injection attacks, commonly hidden as valid parameters. XML Schema facets support for regular expressions was exploited to prevent SQL and Command injection.

Schema Structural and Data-type Validation

As performance is critical for network devices, we had to provide different levels of support for XML Schema. When validation XML messages against XML Schema, we provided options to do only structural validation or structural and data-type validation. This was useful as data-type validation was too much of a performance hit for some applications.

Contents

7.

1. Rule 1: All XML Must Have a Root Element
2. Rule 2: All Tags Must Be Closed
3. Rule 3: All Tags Must Be Properly Nested
4. Rule 4: Tag Names Have Strict Limits
5. Rule 5: Tag Names Are Case Sensitive
6. Rule 6: Tag Names Cannot Contain Spaces
7. Rule 7: Attribute Values Must Appear Within Quotes
8. Rule 8: White Space Is Preserved
9. Rule 9: Avoid HTML Tags (Optional)

XML Syntax Rules

‹ Previous

Next ›

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

XML Documents Must Have a Root Element

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

```
<root>
  <child>
    <subchild>.....</subchild>
  </child>
</root>
```

In this example **<note>** is the root element:

```
<?xml version="1.0" encoding="UTF-8"?>
<note>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

The XML Prolog

This line is called the XML **prolog**:

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

The XML prolog is optional. If it exists, it must come first in the document.

XML documents can contain international characters, like Norwegian øæå or French êèé.

To avoid errors, you should specify the encoding used, or save your XML files as UTF-8.

UTF-8 is the default character encoding for XML documents.

Character encoding can be studied in our Character Set Tutorial.

UTF-8 is also the default encoding for HTML5, CSS, JavaScript, PHP, and SQL.

All XML Elements Must Have a Closing Tag

In XML, it is illegal to omit the closing tag. All elements **must** have a closing tag:

```
<p>This is a paragraph.</p>
<br />
```

Note: The XML prolog does not have a closing tag! This is not an error. The prolog is not a part of the XML document.

XML Tags are Case Sensitive

XML tags are case sensitive. The tag <Letter> is different from the tag <letter>.

Opening and closing tags must be written with the same case:

```
<message>This is correct</message>
```

"Opening and closing tags" are often referred to as "Start and end tags". Use whatever you prefer. It is exactly the same thing.

XML Elements Must be Properly Nested

In HTML, you might see improperly nested elements:

```
<b><i>This text is bold and italic</b></i>
```

In XML, all elements **must** be properly nested within each other:

```
<b><i>This text is bold and italic</i></b>
```

In the example above, "Properly nested" simply means that since the <i> element is opened inside the <b> element, it must be closed inside the <b> element.

XML Attribute Values Must Always be Quoted

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

In XML, the attribute values must always be quoted:

```
<note date="12/11/2007">
  <to>Tove</to>
  <from>Jani</from>
</note>
```

Entity References

Some characters have a special meaning in XML.

If you place a character like "<" inside an XML element, it will generate an error because the parser interprets it as the start of a new element.

This will generate an XML error:

```
<message>salary < 1000</message>
```

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

```
<message>salary &lt; 1000</message>
```

There are 5 pre-defined entity references in XML:

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

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

Comments in XML

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

```
<!-- This is a comment -->
```

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

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

White-space is Preserved in XML

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

XML:	Hello Tove
HTML:	Hello Tove

XML Stores New Line as LF

Windows applications store a new line as: carriage return and line feed (CR+LF).

Unix and Mac OSX use LF.

Old Mac systems use CR.

XML stores a new line as LF.

Well Formed XML

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

How to Create an .Xml File

By Darla Ferrara

XML stands for Extensible Markup Language. This form of computer programming transports and stores data as a text file but does not display it. For the data to display, XML must work in conjunction with another language such as PHP, JavaScript or HTML. Data, stored in a tree structure, uses elements designed and named by the person writing the document. Writing data into an XML file is a straightforward process that just about anyone can master with a little practice.

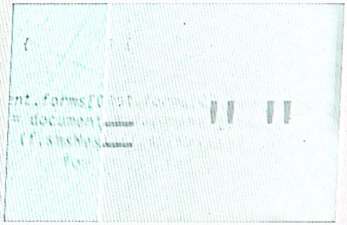


Image Credit: Stockbyte/Stockbyte/Getty Images

Step 1

Open a text editor. To write XML documents you can use a standard text editor such as Notepad. Click on the "Start" button located at the bottom of your desktop. Select "All Programs," then "Accessories." Click on "Notepad" to open the program.

Step 2

Make your declaration at the top of the page. An XML declaration gives the program running your page instructions. A standard declaration starts with <?xml, lists the version and ends with a closing tag. An example of a declaration might look something like this:

The declaration should always be the first line of any XML document.

Step 3

Set up your tree structure with a root element and child elements. The tree structure is the heart of the XML language. A root element is the first line of your tree and it defines or names the data stream you are writing. For instance, you might want to name the category of a product like music as your root element. To do this, your first line would be . The < tells the program you are naming an element. The fact that it is the first item makes it the root. Child elements provide details of the root. Using this example, a child element might be and would be listed underneath the root. All elements require opening and closing tags. A closing tag tells the program you are done with a particular element. The closing tag for music would look like . The closing tag falls at the end of each element. As is a child of , it would be located between the two tags:

Step 4

Check your tree for any errors, such as missing tags. Remember, all elements must have a closing tag.

Step 5

Save your file in the XML format by using .xml as the extension. In most programs, this means clicking on "File" located in the software menu at the top of the screen. Select "File," then "Save As." Type in a name for the file and end it with .xml. The document will save to your hard drive as an XML file. For instance, a music catalog created in XML might be "music.xml."

Create an XML document 8.

1. Start Microsoft Visual Studio 2005 or Microsoft Visual Studio .NET. Then, create a new XML file (on the File menu, point to **New**, and then click **File**).

2. Select the XML File type, and then click **Open**.

3. Add the following data to the XML document to represent a product in a catalog:

XML Copy

```xml
<Product ProductID="123">
    <ProductName>Rugby jersey</ProductName>
</Product>
```

4. Save the file as *Product.xml* in a folder that you will be able to readily access later (the code samples in this article assume a folder named `C:\MyFolder`).

Create a DTD and link to the XML document

1. In Visual Studio 2005 or in Visual Studio .NET, point to New on the File menu, and then click File.
2. Select the Text File type, and then click Open.
3. Add the following DTD declarations to the file to describe the grammar of the XML document:

XML Copy

```
<!ELEMENT Product (ProductName)>
<!ATTLIST Product ProductID CDATA #REQUIRED>
<!ELEMENT ProductName (#PCDATA)>
```

4. Save the file as *Product.dtd* in the same folder as your XML document.
5. Reopen *Product.xml* in Visual Studio 2005 or in Visual Studio .NET; to do this, point to Open on the File menu, and then click File. Add a DOCTYPE statement (below the `?xml version="1.0"` line), as follows. This links the XML document to the DTD file).

XML Copy

```
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE Product SYSTEM "Product.dtd">
```

6. Save the modified XML document as *ProductWithDTD.xml*.

Perform validation by using a DTD

1. In Visual Studio 2005 or in Visual Studio .NET, create a new Visual Basic Console Application project named *ValidateXmlUsingVB*.

2. Visual Studio 2005 or Visual Studio .NET displays a new file named *Module1.vb*. At the beginning of this file, add two Imports statements, as follows:

vbnet Copy

```vbnet
Imports System.Xml ' For XmlTextReader and XmlValidatingReader
Imports System.Xml.Schema ' For XmlSchemaCollection (used later)
```

3. In Module1 (before the start of the Main subroutine), declare a boolean variable named *isValid*, as follows:

vbnet Copy

```vbnet
'If a validation error occurs,
' you will set this flag to False
' in the validation event handler.
Private isValid As Boolean = True
```

4. In the Main subroutine, create an XmlTextReader object to read an XML document from a text file. Then, create an `XmlValidatingReader` object to validate this XML data:

vbnet Copy

```
Dim r As New XmlTextReader("C:\MyFolder\ProductWithDTD.xml")
Dim v As New XmlValidatingReader(r)
```

5. The XmlValidatingReader object has a ValidationType property, which indicates the type of validation required (DTD, XDR, or Schema). Set this property to DTD, as follows:

vbnet Copy

```
v.ValidationType = ValidationType.DTD
```

6. If any validation errors occur, the validating reader generates a validation event. Add the following code to register a validation event handler (you will implement the MyValidationEventHandler subroutine in step 8 of this section):

vbnet Copy

```
AddHandler v.ValidationEventHandler, AddressOf MyValidationEventHandler
```

7. Add the following code to read and validate the XML document. If any validation errors occur, MyValidationEventHandler will be called to handle the error. This subroutine will set `isValid` to *False* (see step 8 of this section). You can check the status of isValid after validation to see whether the document is valid or invalid.

vbnet Copy

```
While v.Read()' Could add code here to process the content.
End While
v.Close()' Check whether the document is valid or invalid.
If isValid Then
    Console.WriteLine("Document is valid")
Else
    Console.WriteLine("Document is invalid")
End If
```

8. After the Main subroutine, write the MyValidationEventHandler subroutine, as follows:

vbnet Copy

```
Public Sub MyValidationEventHandler(ByVal sender As Object, _
 ByVal args As ValidationEventArgs)
     isValid = False
     Console.WriteLine("Validation event" & vbCrLf & args.Message)
End Sub
```

9. Build and run the application.

The application should report that the XML document is valid. 10. In Visual Studio 2005 or in Visual Studio .NET, modify ProductWithDTD.xml to make it invalid (for example, delete the **ProductName** Rugby jersey/ **ProductName** element). 11. Run the application again.

The application should display the following error message:

> Validation event Element 'Product' has incomplete content. Expected 'ProductName'. An error occurred at file:///C:/MyFolder/ProductWithDTD.xml(4, 3). Document is invalid

9.

2.12 XSLT introduction

- XSL (eXtensible Stylesheet Language) is a styling language for XML.
- XSLT stands for XSL Transformations.
- XSLT is the most important part of XSL
- XSLT transforms an XML document into another XML document
- XSLT uses XPath to navigate in XML document.

Example:

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
  <html>
  <body>
    <h2>My CD Collection</h2>
    <table border="1">
      <tr bgcolor="#9acd32">
        <th>Title</th>
        <th>Artist</th>
      </tr>
      <xsl:for-each select="catalog/cd">
        <tr>
          <td><xsl:value-of select="title"/></td>
          <td><xsl:value-of select="artist"/></td>
        </tr>
      </xsl:for-each>
    </table>
  </body>
  </html>
</xsl:template>
</xsl:stylesheet>
```

XSL - More Than a Style Sheet Language
XSL Consists of four:

XSLT - a language for transforming XML documents

XPath - a language for navigating in XML documents

XSL-FO - a language for formatting XML documents (discontinued in 2013)

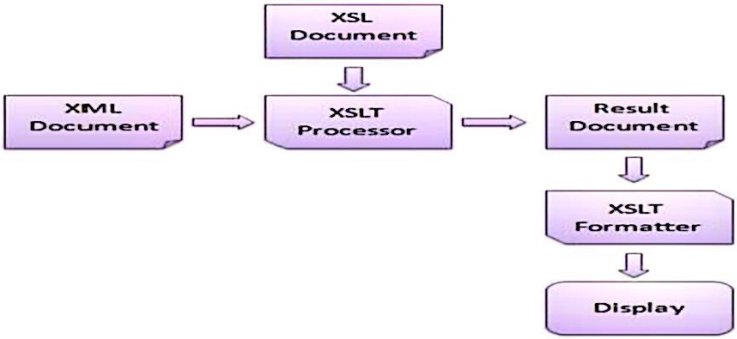
XQuery - a language for querying XML documents

How XSLT Works

•An XSLT stylesheet is used to define the transformation rules to be applied on the target XML document.

•XSLT stylesheet is written in XML format. XSLT Processor takes the XSLT stylesheet and applies the transformation rules on the target XML document and then it generates a formatted document in the form of XML, HTML, or text format.

•This formatted document is then utilized by XSLT formatter to generate the actual output which is to be displayed to the end-user.



Advantages :

•Independent of programming. Transformations are written in a separate xsl file which is again an XML document.

•Output can be altered by simply modifying the transformations in xsl file. No need to change any code. So Web designers can edit the stylesheet and can see the change in the output quickly.

2.14 SAX (Simple API for XML) :

SAX (Simple API for XML) is an event-based parser for XML documents. Unlike a DOM parser, a SAX parser creates no parse tree. SAX is a streaming interface for XML, which means that applications using SAX receive event notifications about the XML document being processed an element, and attribute, at a time in sequential order starting at the top of the document, and ending with the closing of the ROOT element.

- Reads an XML document from top to bottom, recognizing the tokens that make up a well-formed XML document.
- Tokens are processed in the same order that they appear in the document.
- Reports the application program the nature of tokens that the parser has encountered as they occur.
- The application program provides an "event" handler that must be registered with the parser.
- As the tokens are identified, callback methods in the handler are invoked with the relevant information.

When to Use?

You should use a SAX parser when –

- You can process the XML document in a linear fashion from top to down.
- The document is not deeply nested.
- You are processing a very large XML document whose DOM tree would consume too much memory. Typical DOM implementations use ten bytes of memory to represent one byte of XML.
- The problem to be solved involves only a part of the XML document.
- Data is available as soon as it is seen by the parser, so SAX works well for an XML document that arrives over a stream.

Disadvantages of SAX :

- We have no random access to an XML document since it is processed in a forward-only manner.
- If you need to keep track of data that the parser has seen or change the order of items, you must write the code and store the data on your own.

10. **2.13 Document Object Model**

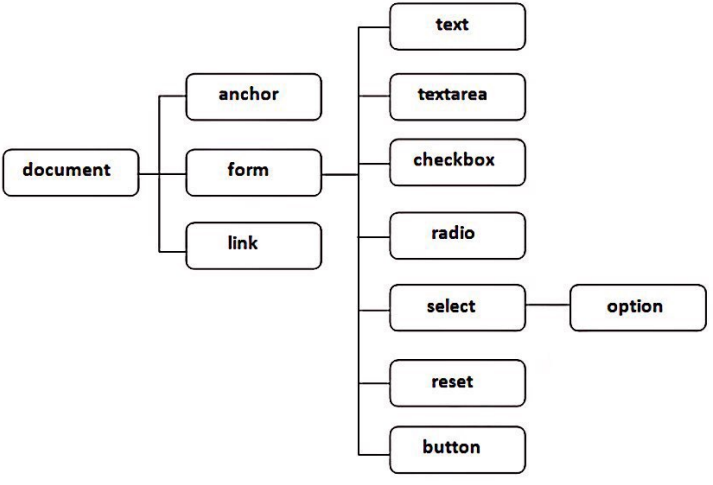
The document object represents the whole html document.

When html document is loaded in the browser, it becomes a document object. It is the root element that represents the html document. It has properties and methods. By the help of document object, we can add dynamic content to our web page.

As mentioned earlier, it is the object of window. So window document Is same as document

Properties of document object :

Let's see the properties of document object that can be accessed and modified by the document object.



ıcument object :

and change the contents of document by its methods.

The important methods of document object are as follows:

Method	Description
write("string")	writes the given string on the doucment.
writeln("string")	writes the given string on the doucment with newline character at the end.
getElementById()	returns the element having the given id value.
getElementsByName()	returns all the elements having the given name value.
getElementsByTagName()	returns all the elements having the given tag name.
getElementsByClassName()	returns all the elements having the given class name.

Accessing field value by document object :

In this example, we are going to get the value of input text by user. Here, we are using document.form1.name.value to get the value of name field.

Here, document is the root element that represents the html document.

form1 is the name of the form.

name is the attribute name of the input text.

value is the property, that returns the value of the input text.

Let's see the simple example of document object that prints name with welcome message.

```
<script type="text/javascript">
function printvalue(){
var name=document.form1.name.value;
alert("Welcome: "+name);
}
</script>
<form name="form1">
Enter Name:<input type="text" name="name"/>
<input type="button" onclick="printvalue()" value="print name"/>
</form>
```