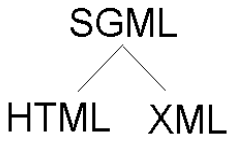
eXtensible Markup Language (XML)



SGML (Standard Generalized Markup Language) is a standard **for how to specify a document markup language or tag set.**

Such a specification is itself a **document type definition** (DTD).

SGML is not in itself a document language, **but a description of how to specify one. It is metadata.**

SGML is based on the idea that documents have structural and other semantic elements that can be described **without reference to how such elements should be displayed.**

The actual display of such a document may vary, depending on the output medium and style preferences.

HTML	✹ **Focus is on presentation/display of content** ✹ Focus on how data looks ✹ Fixed set of predefined tags ✹ No data validation capabilities ✹ Single presentation
XML	✹ **Focus is how to store and send content** ✹ Focus on what data is ✹ Extensible set of **user-defined tags** ✹ Data description language ✹ Standard Data infrastructure ✹ Allows multiple output forms

XML -

- XML stands for eXtensible Markup Language
- Designed to store and transport data
- It is just information wrapped in tags.
- Designed to be self-descriptive
- Software and hardware independent
- XML is a W3C Recommendation

XML Simplifies Things

- It simplifies data sharing
- It simplifies data transport
- It simplifies platform changes
- It simplifies data availability

- Many computer systems contain data in incompatible formats. Exchanging data between incompatible systems (or upgraded systems) is a time-consuming task for web developers. Large amounts of data must be converted, and incompatible data is often lost.

- **XML stores data in plain text format.** This provides a software and hardware independent way of storing, transporting, and sharing data.

- XML also makes it easier to expand or upgrade to new operating systems, new applications, or new browsers, without losing data.

- With XML, data can be available to all kinds of "reading machines" like people, computers, voice machines, news feeds, etc.

XML Does Not DO Anything - XML Example

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

<note>
  <to>Archi</to>
  <from>Parsha</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

The XML above is quite self-descriptive:

- It has sender information.
- It has receiver information
- It has a heading
- It has a message body.

XML is just information wrapped in tags.

Note

To: Archi

From: Parsha

Heading: Reminder

Body: Don't forget me this weekend!

How Can XML be Used?

XML is used in many aspects of web development.

XML is often used to separate data from presentation.

XML Separates Data from Presentation

XML does not carry any information about how to be displayed.

The same XML data can be used in many different presentation scenarios.

Because of this, with XML, there is a full separation between data and presentation.

XML is Often a Complement to HTML

In many HTML applications, XML is used to store or transport data, while HTML is used to format and display the same data.

XML Separates Data from HTML

When displaying data in HTML, you should not have to edit the HTML file **when the data changes**.

With XML, the data can be stored in separate XML files.

Book.xml

Title	Author
Everyday Italian	Giada De Laurentiis
Harry Potter	J K. Rowling
XQuery Kick Start	James McGovern
Learning XML	Erik T. Ray

```xml
<?xml version="1.0" encoding="UTF-8"?>
<bookstore>
  <book>
    <title lang="en">Everyday Italian</title>
    <author>Giada De Laurentiis</author>
  </book>

  <book>
    <title lang="en">Harry Potter</title>
    <author>J K. Rowling</author>
  </book>

  <book>
    <title lang="en">XQuery Kick Start</title>
    <author>James McGovern</author>
  </book>

  <book>
    <title lang="en">Learning XML</title>
    <author>Erik T. Ray</author>
  </book>
</bookstore>
```

With a few lines of JavaScript code, you can read an XML file and update the data content of any HTML page.

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

<bookstore>

  <book category="cooking">

    <title lang="en">Everyday Italian</title>

    <author>Giada De Laurentiis</author>

    <year>2005</year>

    <price>30.00</price>

  </book>

</bookstore>
```

Title	Author	Category	Price	Language	Year
Everyday Italian	Giada De Laurentiis	Cooking	$30	English	2005
Harry Potter	J K. Rowling	Children	$29.99	English	2005
XQuery Kick Start	James McGovern	Web	$49.99	English	2003
Learning XML	Erik T. Ray	Web	$39.95	English	2003

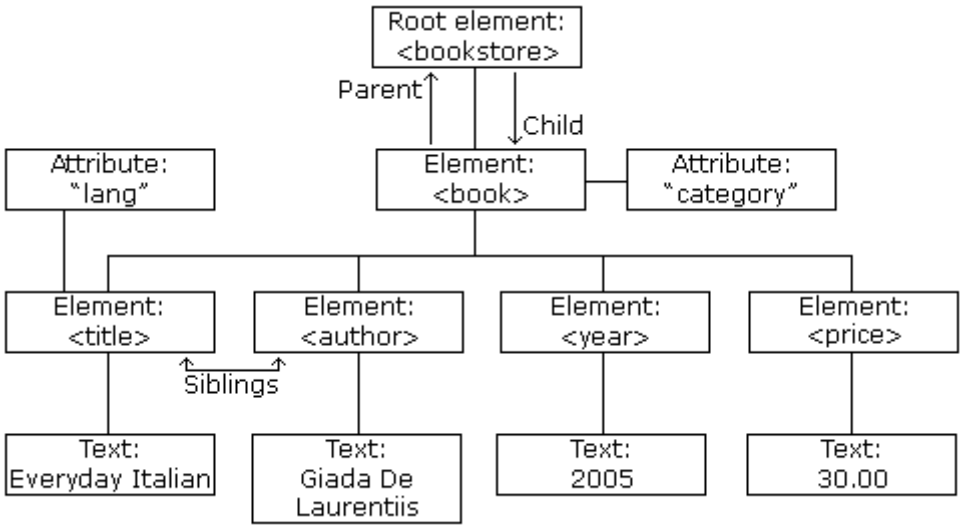
Transaction Data

Thousands of XML formats exist, in many different industries, to describe day-to-day data transactions:

- Stocks and Shares
- Financial transactions
- Medical data
- Mathematical data
- Scientific measurements
- News information
- Weather services

• XML Tree Structure

XML documents form a tree structure that starts at "the root" and branches to "the leaves".



XML documents are formed as **element trees**.

An XML tree starts at a **root element** and branches from the root to **child elements**.

All elements can have sub elements (child elements):

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

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

<bookstore>
  <book category="cooking">
    <title lang="en">Everyday Italian</title>
    <author>Giada De Laurentiis</author>
    <year>2005</year>
    <price>30.00</price>
  </book>
</bookstore>
```

Self-Describing Syntax

XML uses a much self-describing syntax.

A prolog defines the XML version and the character encoding:

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

The next line is the **root element** of the document:

```xml
<bookstore>
```

The next line starts a <book> element:

```xml
<book category="cooking">
```

The <book> elements have **4 child elements**: <title>, <author>, <year>, <price>.

```xml
<title lang="en">Everyday Italian</title>
<author>Giada De Laurentiis</author>
<year>2005</year>
<price>30.00</price>
```

The next line ends the book element:

```xml
</book>
```

XML Syntax Rules - Well Formed" XML documents.

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

Rule-1

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.

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.

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

Rule-2

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

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

```
<?xml version="1.0" encoding="UTF-8"?>
<bookstore>
  <book category="cooking">
    <title lang="en">Everyday Italian</title>
    <author>Giada De Laurentiis</author>
    <year>2005</year>
    <price>30.00</price>
  </book>
</bookstore>
```

In this example **bookstore** is root element.

Rule-3

All elements **must** have a closing tag:

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

Rule-4

XML tags are case sensitive.

The tag <Letter> is different from the tag <letter>.

Rule-5

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

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

Rule-6

In XML, the attribute values must always be quoted:

```
<note date="12/11/2007">
  <to>Archi</to>
  <from>Parsha</from>
</note>
```

Rule-7

Use of entity references to avoid parsing errors

There are 5 pre-defined entity references in XML:

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

```
<message>salary < 1000</message>    ---- will generate error.
```

```
 It must be
```

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

Rule-8

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 -->
```

Rule-9

White-spaces are preserved in XML

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

XML:	Hello Mamta
HTML:	Hello Mamta

Rule-10

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.

What is an XML Element?

An XML element is everything from (including) the element's start root tag to (including) the all tags representing child.....subchild elements tags up to element's end root tag.

```
<price>29.99</price>
```

An element can contain:

- text
- attributes
- other elements
- or a mix of the above

```
<bookstore>
   <book category="web">
    <title>Learning XML</title>
    <author>Erik T. Ray</author>
    <year>2003</year>
    <price>39.95</price>
  </book>
</bookstore>
```

Empty XML Elements

An element with no content is said to be empty.

In XML, you can indicate an empty element like this:

```
<element></element>    or    <element />
```

Empty elements can have attributes.

XML Naming Rules

XML elements must follow these naming rules:

- Element names are case-sensitive
- Element names must start with a letter or underscore
- Element names cannot start with the letters xml (or XML, or Xml, etc)
- Element names can contain letters, digits, hyphens, underscores, and periods
- Element names cannot contain spaces

Any name can be used, no words are reserved (except xml).

Best Naming Practices

Create descriptive names, like this: <person>, <firstname>, <lastname>.

Create short and simple names, like this: <book_title> not like this: <the_title_of_the_book>.

Avoid first-name

Avoid first.name

Avoid ":". Colons are reserved for namespaces

Naming Styles

There are no naming styles defined for XML elements. But here are some commonly used:

Style	Example	Description
Lower case	<firstname>	All letters lower case
Upper case	<FIRSTNAME>	All letters upper case
Underscore	<first_name>	Underscore separates words
Pascal case	<FirstName>	Uppercase first letter in each word
Camel case	<firstName>	Uppercase first letter in each word except the first

XML Attributes

XML elements can have attributes, just like HTML.

Attributes are designed to contain data related to a specific element.

Attribute values must always be quoted. Either single or double quotes can be used.

For a person's gender, the <person> element can be written like this:

```
<person gender="female">
```

or like this:

```
<person gender='female'>
```

If the attribute value itself contains double quotes you can use single quotes, like in this example:

```
<gangster name='George "Shotgun" Ziegler'>
```

or you can use character entities:

```
<gangster name="George &quot;Shotgun&quot; Ziegler">
```

XML Attributes for Metadata

Sometimes ID references are assigned to elements. These IDs can be used to identify XML elements in much the same way as the id attribute in HTML. This example demonstrates this:

```
<messages>
  <note id="501">
    <to>Tove</to>
    <from>Jani</from>
    <heading>Reminder</heading>
    <body>Don't forget me this weekend!</body>
  </note>
  <note id="502">
    <to>Jani</to>
    <from>Tove</from>
    <heading>Re: Reminder</heading>
    <body>I will not</body>
  </note>
</messages>
```

XML Namespaces ---denoted as colon :

XML Namespaces provide a method to avoid element name conflicts.

```
<table>
  <tr>
    <td>Apples</td>
    <td>Bananas</td>
  </tr>
</table>
```

This XML carries information about a table (a piece of furniture):

```
<table>
  <name>African Coffee Table</name>
  <width>80</width>
  <length>120</length>
</table>
```

Resolving conflict using Namespace

```
<h:table>
  <h:tr>
    <h:td>Apples</h:td>
    <h:td>Bananas</h:td>
  </h:tr>
</h:table>

<f:table>
  <f:name>African Coffee Table</f:name>
  <f:width>80</f:width>
  <f:length>120</f:length>
</f:table>
```

XML Namespaces - The xmlns Attribute

```
<root>

<h:table xmlns:h="http://www.w3.org/TR/html4/">
  <h:tr>
    <h:td>Apples</h:td>
    <h:td>Bananas</h:td>
  </h:tr>
</h:table>

<f:table xmlns:f="https://www.w3schools.com/furniture">
  <f:name>African Coffee Table</f:name>
  <f:width>80</f:width>
  <f:length>120</f:length>
</f:table>

</root>
```

The XMLHttpRequest Object

The XMLHttpRequest object can be used to request data from a web server.

The XMLHttpRequest object can:

- 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

Sending an XMLHttpRequest

A common JavaScript syntax for using the XMLHttpRequest object looks much like this:

```
Step-I var xhttp = new XMLHttpRequest();

xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
       // Typical action to be performed when the document is ready:
       document.getElementById("demo").innerHTML = xhttp.responseText;
    }
};
Step-II xhttp.open("GET", "filename", true);

Step-III xhttp.send();
```

The first line in the example above creates an **XMLHttpRequest** object:

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

The **onreadystatechange** property specifies a function to be executed every time the status of the XMLHttpRequest object changes:

```
xhttp.onreadystatechange = function()
```

When **readyState** property is 4 and the **status** property is 200, the response is ready:

```
if (this.readyState == 4 && this.status == 200)
```

The **responseText** property returns the server response as a text string.

The text string can be used to update a web page:

```
document.getElementById("demo").innerHTML = xhttp.responseText;
```

XML Parser

The XML DOM (Document Object Model) defines the properties and methods for accessing and editing XML.

However, before an XML document can be accessed, it must be loaded into an XML DOM object.

All modern browsers have a built-in XML parser that can convert text into an XML DOM object.

```
<html>
<body>

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

<script>
var text, parser, xmlDoc;

text = "<bookstore><book>" +
"<title>Everyday Italian</title>" +
"<author>Giada De Laurentiis</author>" +
"<year>2005</year>" +
"</book></bookstore>";

parser = new DOMParser();
xmlDoc = parser.parseFromString(text,"text/xml");

document.getElementById("demo").innerHTML =
xmlDoc.getElementsByTagName("title")[0].childNodes[0].nodeValue;
</script>

</body>
</html>
```

O/p - Everyday Italian

The XMLHttpRequest Object

The **XMLHttpRequest Object** has a built in XML Parser.

The **responseText** property returns the response as a string.

The **responseXML** property returns the response as an XML DOM object.

If you want to use the response as an XML DOM object, you can use the responseXML property.

```
<!DOCTYPE html>
<html>
<body>
<h2>My CD Collection:</h2>

<button type="button" onclick="loadXMLDoc()">
Get my CD collection</button>

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

<script>
function loadXMLDoc() {
  var xmlhttp = new XMLHttpRequest();
  xmlhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
      myFunction(this);
```

```
    }
  };
  xmlhttp.open("GET", "cd_catalog.xml", true);
  xmlhttp.send();
}

function myFunction(xml) {
  var x, i, xmlDoc, txt;
  xmlDoc = xml.responseXML;
  txt = "";
  x = xmlDoc.getElementsByTagName("ARTIST");
  for (i = 0; i< x.length; i++) {
    txt += x[i].childNodes[0].nodeValue + "<br>";
  }
  document.getElementById("demo").innerHTML = txt;
}
</script>

</body>
</html>
```