http://www.w3schools.com/xml/xml_whatis.asp

Introduction to XML

XML was designed to transport and store data.

HTML was designed to display data.

What You Should Already Know

Before you continue you should have a basic understanding of the following:

- HTML
- JavaScript

If you want to study these subjects first, find the tutorials on our Home page.

What is XML?

- XML stands for EXtensible Markup Language
- XML is a markup language much like HTML
- XML was designed to carry data, not to display data
- XML tags are not predefined. You must define your own tags
- XML is designed to be self-descriptive
- XML is a W3C Recommendation

The Difference Between XML and HTML

XML is not a replacement for HTML.

XML and HTML were designed with different goals:

- XML was designed to transport and store data, with focus on what data is
- HTML was designed to display data, with focus on how data looks

HTML is about displaying information, while XML is about carrying information.

XML Does Not DO Anything

Maybe it is a little hard to understand, but XML does not DO anything. XML was created to structure, store, and transport information.

The following example is a note to Tove, from Jani, stored as XML:

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

```
</note>
```

The note above is quite self descriptive. It has sender and receiver information, it also has a heading and a message body.

But still, this XML document does not DO anything. It is just information wrapped in tags. Someone must write a piece of software to send, receive or display it.

With XML You Invent Your Own Tags

The tags in the example above (like <to> and <from>) are not defined in any XML standard. These tags are "invented" by the author of the XML document.

That is because the XML language has no predefined tags.

The tags used in HTML are predefined. HTML documents can only use tags defined in the HTML standard (like <p>, <h1>, etc.).

XML allows the author to define his/her own tags and his/her own document structure.

XML is Not a Replacement for HTML

XML is a complement to HTML.

It is important to understand that XML is not a replacement for HTML. In most web applications, XML is used to transport data, while HTML is used to format and display the data.

My best description of XML is this:

XML is a software- and hardware-independent tool for carrying information.

XML is a W3C Recommendation

XML became a W3C Recommendation 10. February 1998.

To read more about the XML activities at W3C, please read our W3C Tutorial.

XML is Everywhere

XML is now as important for the Web as HTML was to the foundation of the Web.

XML is the most common tool for data transmissions between all sorts of applications.

XML is used in many aspects of web development, often to simplify data storage and sharing.

XML Separates Data from HTML

If you need to display dynamic data in your HTML document, it will take a lot of work to edit the HTML each time the data changes.

With XML, data can be stored in separate XML files. This way you can concentrate on using HTML for layout and display, and be sure that changes in the underlying data will not require any changes to the HTML.

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

XML Simplifies Data Sharing

In the real world, computer systems and databases contain data in incompatible formats.

XML data is stored in plain text format. This provides a software- and hardware-independent way of storing data.

This makes it much easier to create data that can be shared by different applications.

XML Simplifies Data Transport

One of the most time-consuming challenges for developers is to exchange data between incompatible systems over the Internet.

Exchanging data as XML greatly reduces this complexity, since the data can be read by different incompatible applications.

XML Simplifies Platform Changes

Upgrading to new systems (hardware or software platforms), is always time consuming. Large amounts of data must be converted and incompatible data is often lost.

XML data is stored in text format. This makes it easier to expand or upgrade to new operating systems, new applications, or new browsers, without losing data.

XML Makes Your Data More Available

Different applications can access your data, not only in HTML pages, but also from XML data sources.

With XML, your data can be available to all kinds of "reading machines" (Handheld computers, voice machines, news feeds, etc), and make it more available for blind people, or people with other disabilities.

XML is Used to Create New Internet Languages

A lot of new Internet languages are created with XML.

Here are some examples:

- XHTML
- WSDL for describing available web services
- WAP and WML as markup languages for handheld devices
- RSS languages for news feeds
- RDF and OWL for describing resources and ontology
- SMIL for describing multimedia for the web

If Developers Have Sense

If they DO have sense, future applications will exchange their data in XML.

The future might give us word processors, spreadsheet applications and databases that can read each other's data in XML format, without any conversion utilities in between

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

An Example XML Document

XML documents use a self-describing and simple syntax:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
```

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

The first line is the XML declaration. It defines the XML version (1.0) and the encoding used (ISO-8859-1 = Latin-1/West European character set).

The next line describes the **root element** of the document (like saying: "this document is a note"):

```
<note>
```

The next 4 lines describe 4 **child elements** of the root (to, from, heading, and body):

```
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
```

And finally the last line defines the end of the root element:

```
</note>
```

You can assume, from this example, that the XML document contains a note to Tove from Jani.

Don't you agree that XML is pretty self-descriptive?

XML Documents Form a Tree Structure

XML documents must contain a **root element**. This element is "the parent" of all other elements.

The elements in an XML document form a document tree. The tree starts at the root and branches to the lowest level of the tree.

All elements can have **sub elements (child elements)**:

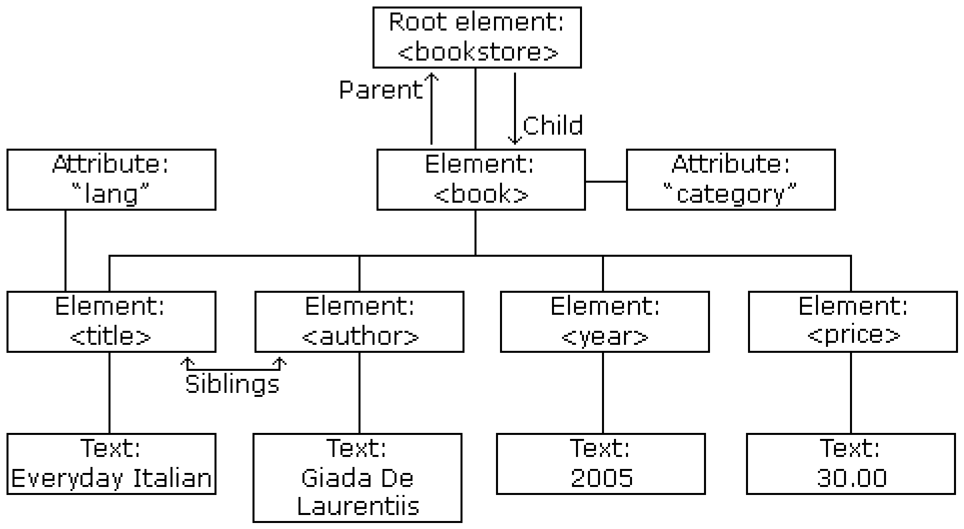
```
<root>
  <child>
    <subchild>.....</subchild>
  </child>
</root>
```

The terms parent, child, and sibling are used to describe the relationships between elements.

Parent elements have children. Children on the same level are called siblings (brothers or sisters).

All elements can have text content and attributes (just like in HTML).

Example:



The image above represents one book in the XML below:

```
<bookstore>
  <book category="COOKING">
    <title lang="en">Everyday Italian</title>
    <author>Giada De Laurentiis</author>
    <year>2005</year>
    <price>30.00</price>
  </book>
  <book category="CHILDREN">
    <title lang="en">Harry Potter</title>
    <author>J K. Rowling</author>
    <year>2005</year>
    <price>29.99</price>
  </book>
  <book category="WEB">
    <title lang="en">Learning XML</title>
    <author>Erik T. Ray</author>
    <year>2003</year>
    <price>39.95</price>
  </book>
</bookstore>
```

The root element in the example is <bookstore>. All <book> elements in the document are contained within <bookstore>.

The <book> element has 4 children: <title>,< author>, <year>, <price>.

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

All XML Elements Must Have a Closing Tag

In HTML, elements do not have to have a closing tag:

```
<p>This is a paragraph
<p>This is another paragraph
```

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

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

Note: You might have noticed from the previous example that the XML declaration did not have a closing tag. This is not an error. The declaration is not a part of the XML document itself, and it has no closing tag.

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

Note: "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 Documents Must Have a Root Element

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

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

XML Attribute Values Must be Quoted

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

In XML, the attribute values must always be quoted.

Study the two XML documents below. The first one is incorrect, the second is correct:

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

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

The error in the first document is that the date attribute in the note element is not quoted.

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>if salary < 1000 then</message>
```

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

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

There are 5 predefined entity references in XML:

&lt;	<	less than
&gt;	>	greater than
&amp;	&	ampersand
&apos;	'	apostrophe
&quot;	"	quotation mark

Note: Only the characters "<" and "&" are strictly illegal in XML. The greater than character is legal, but it is a good habit to replace it.

Comments in XML

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

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

White-space is Preserved in XML

HTML truncates multiple white-space characters to one single white-space:

HTML:	Hello Tove
Output:	Hello Tove

With XML, the white-space in a document is not truncated.

XML Stores New Line as LF

In Windows applications, a new line is normally stored as a pair of characters: carriage return (CR) and line feed (LF). In Unix applications, a new line is normally stored as a LF character. Macintosh applications also use an LF to store a new line.

XML stores a new line as LF

An XML document contains XML Elements.

What is an XML Element?

An XML element is everything from (including) the element's start tag to (including) the element's end tag.

An element can contain other elements, simple text or a mixture of both. Elements can also have attributes.

```
<bookstore>
  <book category="CHILDREN">
    <title>Harry Potter</title>
    <author>J K. Rowling</author>
    <year>2005</year>
    <price>29.99</price>
  </book>
  <book category="WEB">
```

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

In the example above, <bookstore> and <book> have **element contents**, because they contain other elements. <author> has **text content** because it contains text.

In the example above only <book> has an **attribute** (category="CHILDREN").

XML Naming Rules

XML elements must follow these naming rules:

- Names can contain letters, numbers, and other characters
- Names cannot start with a number or punctuation character
- Names cannot start with the letters xml (or XML, or Xml, etc)
- Names cannot contain spaces

Any name can be used, no words are reserved.

Best Naming Practices

Make names descriptive. Names with an underscore separator are nice: <first_name>, <last_name>.

Names should be short and simple, like this: <book_title> not like this: <the_title_of_the_book>.

Avoid "-" characters. If you name something "first-name," some software may think you want to subtract name from first.

Avoid "." characters. If you name something "first.name," some software may think that "name" is a property of the object "first."

Avoid ":" characters. Colons are reserved to be used for something called namespaces (more later).

XML documents often have a corresponding database. A good practice is to use the naming rules of your database for the elements in the XML documents.

Non-English letters like éòá are perfectly legal in XML, but watch out for problems if your software vendor doesn't support them.

XML Elements are Extensible

XML elements can be extended to carry more information.

Look at the following XML example:

```
<note>
<to>Tove</to>
<from>Jani</from>
<body>Don't forget me this weekend!</body>
</note>
```

Let's imagine that we created an application that extracted the <to>, <from>, and <body> elements from the XML document to produce this output:

MESSAGE

To: Tove
From: Jani

Don't forget me this weekend!

Imagine that the author of the XML document added some extra information to it:

```
<note>
<date>2008-01-10</date>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
</note>
```

Should the application break or crash?

No. The application should still be able to find the <to>, <from>, and <body> elements in the XML document and produce the same output.

One of the beauties of XML, is that it can be extended without breaking applications.

XML elements can have attributes, just like HTML.

Attributes provide additional information about an element.

XML Attributes

In HTML, attributes provide additional information about elements:

```
<img src="computer.gif">
<a href="demo.asp">
```

Attributes often provide information that is not a part of the data. In the example below, the file type is irrelevant to the data, but can be important to the software that wants to manipulate the element:

```
<file type="gif">computer.gif</file>
```

XML Attributes Must be Quoted

Attribute values must always be quoted. Either single or double quotes can be used. For a person's sex, the person element can be written like this:

```
<person sex="female">
```

or like this:

```
<person sex='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 Elements vs. Attributes

Take a look at these examples:

```
<person sex="female">
  <firstname>Anna</firstname>
  <lastname>Smith</lastname>
</person>
```

```
<person>
  <sex>female</sex>
  <firstname>Anna</firstname>
  <lastname>Smith</lastname>
</person>
```

In the first example sex is an attribute. In the last, sex is an element. Both examples provide the same information.

There are no rules about when to use attributes or when to use elements. Attributes are handy in HTML. In XML my advice is to avoid them. Use elements instead.

My Favorite Way

The following three XML documents contain exactly the same information:

A date attribute is used in the first example:

```
<note date="10/01/2008">
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

A date element is used in the second example:

```
<note>
  <date>10/01/2008</date>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

An expanded date element is used in the third: (THIS IS MY FAVORITE):

```
<note>
  <date>
    <day>10</day>
    <month>01</month>
    <year>2008</year>
  </date>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

Avoid XML Attributes?

Some of the problems with using attributes are:

- attributes cannot contain multiple values (elements can)
- attributes cannot contain tree structures (elements can)
- attributes are not easily expandable (for future changes)

Attributes are difficult to read and maintain. Use elements for data. Use attributes for information that is not relevant to the data.

Don't end up like this:

```
<note day="10" month="01" year="2008"
to="Tove" from="Jani" heading="Reminder"
body="Don't forget me this weekend!">
```

```
</note>
```

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>
```

The id attributes above are for identifying the different notes. It is not a part of the note itself.

What I'm trying to say here is that metadata (data about data) should be stored as attributes, and the data itself should be stored as elements

XML with correct syntax is "Well Formed" XML.

XML validated against a DTD is "Valid" XML.

Well Formed XML Documents

A "Well Formed" XML document has correct XML syntax.

The syntax rules were described in the previous chapters:

- XML documents must have a root element
- XML elements must have a closing tag
- XML tags are case sensitive
- XML elements must be properly nested
- XML attribute values must be quoted

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
</note>
```

Valid XML Documents

A "Valid" XML document is a "Well Formed" XML document, which also conforms to the rules of a Document Type Definition (DTD):

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE note SYSTEM "Note.dtd">
<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
</note>
```

The DOCTYPE declaration in the example above, is a reference to an external DTD file. The content of the file is shown in the paragraph below.

XML DTD

The purpose of a DTD is to define the structure of an XML document. It defines the structure with a list of legal elements:

```
<!DOCTYPE note
[
<!ELEMENT note (to,from,heading,body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
]>
```

If you want to study DTD, you will find our DTD tutorial on our homepage.

XML Schema

W3C supports an XML-based alternative to DTD, called XML Schema:

```
<xs:element name="note">

<xs:complexType>
  <xs:sequence>
    <xs:element name="to" type="xs:string"/>
    <xs:element name="from" type="xs:string"/>
    <xs:element name="heading" type="xs:string"/>
    <xs:element name="body" type="xs:string"/>
  </xs:sequence>
</xs:complexType>

</xs:element>
```

If you want to study XML Schema, you will find our Schema tutorial on our homepage.

A General XML Validator

To help you check the syntax of your XML files, we have created an XML validator to syntax-check your XML.

Please see the next chapter.

Use our XML validator to syntax-check your XML.

XML Errors Will Stop You

Errors in XML documents will stop your XML applications.

The W3C XML specification states that a program should stop processing an XML document if it finds an error. The reason is that XML software should be small, fast, and compatible.

HTML browsers will display documents with errors (like missing end tags). HTML browsers are big and incompatible because they have a lot of unnecessary code to deal with (and display) HTML errors.

With XML, errors are not allowed.

Syntax-Check Your XML

To help you syntax-check your XML, we have created an XML validator.

Paste your XML into the text area below, and syntax-check it by clicking the "Validate" button.

Note: This only checks if your XML is "Well formed". If you want to validate your XML against a DTD, see the last paragraph on this page.

Syntax-Check an XML File

You can syntax-check an XML file by typing the URL of the file into the input field below, and then click the "Validate" button:

Filename:

http://www.w3schools.com/xml/note_error.xml

Validate

Note: If you get an "Access denied" error, it's because your browser security does not allow file access across domains.

The file "note_error.xml" demonstrates your browsers error handling. If you want see an error free message, substitute the "note_error.xml" with "cd_catalog.xml".

Validate Your XML Against a DTD

If you know DTD, you can validate your XML in the text area below.

Just add the DOCTYPE declaration to your XML and click the "Validate" button:

Note: Only Internet Explorer will actually check your XML against the DTD. Firefox, Mozilla, Netscape, and Opera will not.

Raw XML files can be viewed in all major browsers.

Don't expect XML files to be displayed as HTML pages.

Viewing XML Files

```
<?xml version="1.0" encoding="ISO-8859-1"?>
 - <note>
     <to>Tove</to>
     <from>Jani</from>
     <heading>Reminder</heading>
     <body>Don't forget me this weekend!</body>
  </note>
```

Look at this XML file: note.xml

The XML document will be displayed with color-coded root and child elements. A plus (+) or minus sign (-) to the left of the elements can be clicked to expand or collapse the element structure. To view the raw XML source (without the + and - signs), select "View Page Source" or "View Source" from the browser menu.

Note: In Chrome, Opera, and Safari, only the element text will be displayed. To view the raw XML, you must right click the page and select "View Source"

Viewing an Invalid XML File

If an erroneous XML file is opened, the browser will report the error.

Look at this XML file: note_error.xml

Other XML Examples

Viewing some XML documents will help you get the XML feeling.

An XML CD catalog
This is a CD collection, stored as XML data.

An XML plant catalog
This is a plant catalog from a plant shop, stored as XML data.

A Simple Food Menu
This is a breakfast food menu from a restaurant, stored as XML data.

Why Does XML Display Like This?

XML documents do not carry information about how to display the data.

Since XML tags are "invented" by the author of the XML document, browsers do not know if a tag like <table> describes an HTML table or a dining table.

Without any information about how to display the data, most browsers will just display the XML document as it is.

In the next chapters, we will take a look at different solutions to the display problem, using CSS, XSLT and JavaScript.

With CSS (Cascading Style Sheets) you can add display information to an XML document.

Displaying your XML Files with CSS?

It is possible to use CSS to format an XML document.

Below is an example of how to use a CSS style sheet to format an XML document:

Take a look at this XML file: The CD catalog

Then look at this style sheet: The CSS file

Finally, view: The CD catalog formatted with the CSS file

Below is a fraction of the XML file. The second line links the XML file to the CSS file:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml-stylesheet type="text/css" href="cd_catalog.css"?>
<CATALOG>
  <CD>
    <TITLE>Empire Burlesque</TITLE>
    <ARTIST>Bob Dylan</ARTIST>
    <COUNTRY>USA</COUNTRY>
    <COMPANY>Columbia</COMPANY>
    <PRICE>10.90</PRICE>
    <YEAR>1985</YEAR>
  </CD>
  <CD>
    <TITLE>Hide your heart</TITLE>
    <ARTIST>Bonnie Tyler</ARTIST>
    <COUNTRY>UK</COUNTRY>
    <COMPANY>CBS Records</COMPANY>
    <PRICE>9.90</PRICE>
    <YEAR>1988</YEAR>
  </CD>
.
.
.
</CATALOG>
```

Formatting XML with CSS is not the most common method.

W3C recommend using XSLT instead. See the next chapter

With XSLT you can transform an XML document into HTML.

Displaying XML with XSLT

XSLT is the recommended style sheet language of XML.

XSLT (eXtensible Stylesheet Language Transformations) is far more sophisticated than CSS.

XSLT can be used to transform XML into HTML, before it is displayed by a browser:

Display XML with XSLT

If you want to learn more about XSLT, find our XSLT tutorial on our homepage.

Transforming XML with XSLT on the Server

In the example above, the XSLT transformation is done by the browser, when the browser reads the XML file.

Different browsers may produce different result when transforming XML with XSLT. To reduce this problem the XSLT transformation can be done on the server.

View the result.

Note that the result of the output is exactly the same, either the transformation is done by the web server or by the web browser.

The XMLHttpRequest Object

The XMLHttpRequest object is used to exchange data with a server behind the scenes.

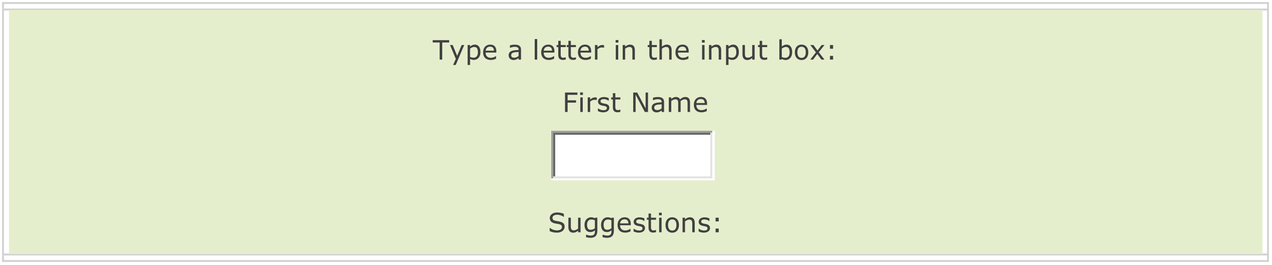
The XMLHttpRequest object is **the developers dream**, because you can:

- 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

To learn more about the XMLHttpRequest object, study our XML DOM tutorial.

XMLHttpRequest Example

When you type a character in the input field below, an XMLHttpRequest is sent to the server - and name suggestions are returned (from a file on the server):



Type a letter in the input box:

First Name

Suggestions:

Create an XMLHttpRequest Object

All modern browsers (IE7+, Firefox, Chrome, Safari, and Opera) have a built-in XMLHttpRequest object.

Syntax for creating an XMLHttpRequest object:

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

Old versions of Internet Explorer (IE5 and IE6) uses an ActiveX Object:

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

In the next chapter, we will use the XMLHttpRequest object to retrieve XML information from a server.

All modern browsers have a built-in XML parser.

An XML parser converts an XML document into an XML DOM object - which can then be manipulated with a JavaScript.

Parse an XML Document

The following code fragment parses an XML document into an XML DOM object:

```
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","books.xml",false);
xmlhttp.send();
xmlDoc=xmlhttp.responseXML;
```

Parse an XML String

The following code fragment parses an XML string into an XML DOM object:

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

if (window.DOMParser)
  {
  parser=new DOMParser();
  xmlDoc=parser.parseFromString(txt,"text/xml");
  }
else // Internet Explorer
  {
  xmlDoc=new ActiveXObject("Microsoft.XMLDOM");
  xmlDoc.async="false";
  xmlDoc.loadXML(txt);
  }
```

Note: Internet Explorer uses the loadXML() method to parse an XML string, while other browsers use the DOMParser object.

Access Across Domains

For security reasons, modern browsers do not allow access across domains.

This means, that both the web page and the XML file it tries to load, must be located on the same server.

The XML DOM

In the next chapter you will learn how to access and retrieve data from the XML DOM object

A DOM (Document Object Model) defines a standard way for accessing and manipulating documents.

The XML DOM

The XML DOM defines a standard way for accessing and manipulating XML documents.

The XML DOM views an XML document as a tree-structure.

All elements can be accessed through the DOM tree. Their content (text and attributes) can be modified or deleted, and new elements can be created. The elements, their text, and their attributes are all known as nodes.

You can learn more about the XML DOM in our XML DOM tutorial.

The HTML DOM

The HTML DOM defines a standard way for accessing and manipulating HTML documents.

All HTML elements can be accessed through the HTML DOM.

You can learn more about the HTML DOM in our HTML DOM tutorial.

Load an XML File - Cross browser Example

The following example parses an XML document ("note.xml") into an XML DOM object, and then extract some info from it with a JavaScript:

Example

```
<html>
<body>
<h1>W3Schools Internal Note</h1>
<p><b>To:</b> <span id="to"></span><br />
<b>From:</b> <span id="from"></span><br />
<b>Message:</b> <span id="message"></span>

<script type="text/javascript">
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","note.xml",false);
xmlhttp.send();
xmlDoc=xmlhttp.responseXML;

document.getElementById("to").innerHTML=
xmlDoc.getElementsByTagName("to")[0].childNodes[0].nodeValue;
document.getElementById("from").innerHTML=
xmlDoc.getElementsByTagName("from")[0].childNodes[0].nodeValue;
document.getElementById("message").innerHTML=
xmlDoc.getElementsByTagName("body")[0].childNodes[0].nodeValue;
```

```
</script>

</body>
</html>
```

Try it yourself »

Important Note!

To extract the text "Tove" from the <to> element in the XML file above ("note.xml"), the syntax is:

```
getElementsByTagName("to")[0].childNodes[0].nodeValue
```

Notice that even if the XML file contains only ONE <to> element you still have to specify the array index [0]. This is because the getElementsByTagName() method returns an array.

Load an XML String - Cross browser Example

The following example parses an XML string into an XML DOM object, and then extract some info from it with a JavaScript:

Example

```
<html>
<body>
<h1>W3Schools Internal Note</h1>
<p><b>To:</b> <span id="to"></span><br />
<b>From:</b> <span id="from"></span><br />
<b>Message:</b> <span id="message"></span></p>

<script>
txt="<note>";
txt=txt+"<to>Tove</to>";
txt=txt+"<from>Jani</from>";
txt=txt+"<heading>Reminder</heading>";
txt=txt+"<body>Don't forget me this weekend!</body>";
txt=txt+"</note>";

if (window.DOMParser)
  {
  parser=new DOMParser();
  xmlDoc=parser.parseFromString(txt,"text/xml");
  }
else // Internet Explorer
  {
  xmlDoc=new ActiveXObject("Microsoft.XMLDOM");
  xmlDoc.async="false";
  xmlDoc.loadXML(txt);
  }

document.getElementById("to").innerHTML=
xmlDoc.getElementsByTagName("to")[0].childNodes[0].nodeValue;
document.getElementById("from").innerHTML=
xmlDoc.getElementsByTagName("from")[0].childNodes[0].nodeValue;
document.getElementById("message").innerHTML=
xmlDoc.getElementsByTagName("body")[0].childNodes[0].nodeValue;
</script>
```

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

Try it yourself »

Add HTML to XML Data

In the following example, we loop through an XML file ("cd_catalog.xml"), and display the content of each CD element as an HTML table row:

Example

```
<html>
<body>

<script type="text/javascript">
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","cd_catalog.xml",false);
xmlhttp.send();
xmlDoc=xmlhttp.responseXML;

document.write("<table border='1'>");
var x=xmlDoc.getElementsByTagName("CD");
for (i=0;i<x.length;i++)
  {
  document.write("<tr><td>");
  document.write(x[i].getElementsByTagName("ARTIST")[0].childNodes[0].nodeValue);
  document.write("</td><td>");
  document.write(x[i].getElementsByTagName("TITLE")[0].childNodes[0].nodeValue);
  document.write("</td></tr>");
  }
document.write("</table>");
</script>

</body>
</html>
```

Try it yourself »

For more information about using JavaScript and the XML DOM, visit our XML DOM tutorial.

This chapter demonstrates some small XML applications built on XML, HTML, XML DOM and JavaScript.

The XML Document Used

In this application we will use the "cd_catalog.xml" file.

Display the First CD in an HTML div Element

The following example gets the XML data from the first CD element and displays it in an HTML element with id="showCD". The displayCD() function is called when the page is loaded:

Example

```
x=xmlDoc.getElementsByTagName("CD");
i=0;

function displayCD()
{
artist=(x[i].getElementsByTagName("ARTIST")[0].childNodes[0].nodeValue);
title=(x[i].getElementsByTagName("TITLE")[0].childNodes[0].nodeValue);
year=(x[i].getElementsByTagName("YEAR")[0].childNodes[0].nodeValue);
txt="Artist: " + artist + "<br />Title: " + title + "<br />Year: "+ year;
document.getElementById("showCD").innerHTML=txt;
}
```

Try it yourself »

Navigate Between the CDs

To navigate between the CDs in the example above, add a next() and previous() function:

Example

```
function next()
{ // display the next CD, unless you are on the last CD
if (i<x.length-1)
  {
  i++;
  displayCD();
  }
}

function previous()
{ // displays the previous CD, unless you are on the first CD
if (i>0)
  {
  i--;
  displayCD();
  }
}
```

Try it yourself »

Show Album Information When Clicking On a CD

The last example shows how you can show album information when the user clicks on a CD:

Try it yourself.

For more information about using JavaScript and the XML DOM, visit our XML DOM tutorial.

XML Namespaces provide a method to avoid element name conflicts.

Name Conflicts

In XML, element names are defined by the developer. This often results in a conflict when trying to mix XML documents from different XML applications.

This XML carries HTML table information:

```
<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>
```

If these XML fragments were added together, there would be a name conflict. Both contain a <table> element, but the elements have different content and meaning.

An XML parser will not know how to handle these differences.

Solving the Name Conflict Using a Prefix

Name conflicts in XML can easily be avoided using a name prefix.

This XML carries information about an HTML table, and a piece of furniture:

```
<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>
```

In the example above, there will be no conflict because the two <table> elements have different names.

XML Namespaces - The xmlns Attribute

When using prefixes in XML, a so-called **namespace** for the prefix must be defined.

The namespace is defined by the **xmlns attribute** in the start tag of an element.

The namespace declaration has the following syntax. xmlns:*prefix*="*URI*".

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

</root>
```

In the example above, the xmlns attribute in the <table> tag give the h: and f: prefixes a qualified namespace.

When a namespace is defined for an element, all child elements with the same prefix are associated with the same namespace.

Namespaces can be declared in the elements where they are used or in the XML root element:

```
<root
xmlns:h="http://www.w3.org/TR/html4/"
xmlns:f="http://www.w3schools.com/furniture">

<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>

</root>
```

Note: The namespace URI is not used by the parser to look up information.

The purpose is to give the namespace a unique name. However, often companies use the namespace as a pointer to a web page containing namespace information.

Try to go to http://www.w3.org/TR/html4/.

Uniform Resource Identifier (URI)

A **Uniform Resource Identifier** (URI) is a string of characters which identifies an Internet Resource.

The most common URI is the **Uniform Resource Locator** (URL) which identifies an Internet domain address. Another, not so common type of URI is the **Universal Resource Name** (URN).

In our examples we will only use URLs.

Default Namespaces

Defining a default namespace for an element saves us from using prefixes in all the child elements. It has the following syntax:

```
xmlns="namespaceURI"
```

This XML carries HTML table information:

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

```
</table>
```

This XML carries information about a piece of furniture:

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

Namespaces in Real Use

XSLT is an XML language that can be used to transform XML documents into other formats, like HTML.

In the XSLT document below, you can see that most of the tags are HTML tags.

The tags that are not HTML tags have the prefix xsl, identified by the namespace xmlns:xsl="http://www.w3.org/1999/XSL/Transform":

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<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>
      <th align="left">Title</th>
      <th align="left">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>
```

If you want to learn more about XSLT, please find our XSLT tutorial at our homepage.

All text in an XML document will be parsed by the parser.

But text inside a CDATA section will be ignored by the parser.

PCDATA - Parsed Character Data

XML parsers normally parse all the text in an XML document.

When an XML element is parsed, the text between the XML tags is also parsed:

```
<message>This text is also parsed</message>
```

The parser does this because XML elements can contain other elements, as in this example, where the <name> element contains two other elements (first and last):

```
<name><first>Bill</first><last>Gates</last></name>
```

and the parser will break it up into sub-elements like this:

```
<name>
  <first>Bill</first>
  <last>Gates</last>
</name>
```

Parsed Character Data (PCDATA) is a term used about text data that will be parsed by the XML parser.

CDATA - (Unparsed) Character Data

The term CDATA is used about text data that should not be parsed by the XML parser.

Characters like "<" and "&" are illegal in XML elements.

"<" will generate an error because the parser interprets it as the start of a new element.

"&" will generate an error because the parser interprets it as the start of an character entity.

Some text, like JavaScript code, contains a lot of "<" or "&" characters. To avoid errors script code can be defined as CDATA.

Everything inside a CDATA section is ignored by the parser.

A CDATA section starts with "**<![CDATA[**" and ends with "**]]>**":

```
<script>
<![CDATA[
function matchwo(a,b)
{
if (a < b && a < 0) then
  {
  return 1;
  }
else
  {
  return 0;
  }
}
]]>
</script>
```

In the example above, everything inside the CDATA section is ignored by the parser.

Notes on CDATA sections:

A CDATA section cannot contain the string "]]>". Nested CDATA sections are not allowed.

The "]]>" that marks the end of the CDATA section cannot contain spaces or line breaks.

XML documents can contain non ASCII characters, like Norwegian æ ø å , or French ê è é.

To avoid errors, specify the XML encoding, or save XML files as Unicode.

XML Encoding Errors

If you load an XML document, you can get two different errors indicating encoding problems:

An invalid character was found in text content.

You get this error if your XML contains non ASCII characters, and the file was saved as single-byte ANSI (or ASCII) with no encoding specified.

Single byte XML file with encoding attribute.

Same single byte XML file with no encoding attribute.

Switch from current encoding to specified encoding not supported.

You get this error if your XML file was saved as double-byte Unicode (or UTF-16) with a single-byte encoding (Windows-1252, ISO-8859-1, UTF-8) specified.

You also get this error if your XML file was saved with single-byte ANSI (or ASCII), with double-byte encoding (UTF-16) specified.

Double byte XML file without encoding.

Same double byte XML file with single byte encoding.

Windows Notepad

Windows Notepad save files as single-byte ANSI (ASCII) by default.

If you select "Save as...", you can specify double-byte Unicode (UTF-16).

Save the XML file below as Unicode (note that the document does not contain any encoding attribute):

```
<?xml version="1.0"?>
<note>
<from>Jani</from>
<to>Tove</to>
<message>Norwegian: æøå. French: êèé</message>
</note>
```

The file above, note_encode_none_u.xml will NOT generate an error. But if you specify a single-byte encoding it will.

The following encoding (open it), will give an error message:

```
<?xml version="1.0" encoding="windows-1252"?>
```

The following encoding (open it), will give an error message:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
```

The following encoding (open it), will give an error message:

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

The following encoding (open it), will NOT give an error:

```
<?xml version="1.0" encoding="UTF-16"?>
```

Conclusion

- Always use the encoding attribute
- Use an editor that supports encoding
- Make sure you know what encoding the editor uses
- Use the same encoding in your encoding attribute

XML on the Server

XML files are plain text files just like HTML files.

XML can easily be stored and generated by a standard web server.

Storing XML Files on the Server

XML files can be stored on an Internet server exactly the same way as HTML files.

Start Windows Notepad and write the following lines:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<note>
  <from>Jani</from>
  <to>Tove</to>
  <message>Remember me this weekend</message>
</note>
```

Save the file on your web server with a proper name like "note.xml".

Generating XML with ASP

XML can be generated on a server without any installed XML software.

To generate an XML response from the server - simply write the following code and save it as an ASP file on the web server:

```
<%
response.ContentType="text/xml"
response.Write("<?xml version='1.0' encoding='ISO-8859-1'?>")
response.Write("<note>")
response.Write("<from>Jani</from>")
response.Write("<to>Tove</to>")
response.Write("<message>Remember me this weekend</message>")
response.Write("</note>")
%>
```

Note that the content type of the response must be set to "text/xml".

See how the ASP file will be returned from the server.

If you want to study ASP, you will find our ASP tutorial on our homepage.

Generating XML with PHP

To generate an XML response from the server using PHP, use following code:

```
<?php
header("Content-type: text/xml");
echo "<?xml version='1.0' encoding='ISO-8859-1'?>";
echo "<note>";
echo "<from>Jani</from>";
echo "<to>Tove</to>";
echo "<message>Remember me this weekend</message>";
echo "</note>";
?>
```

Note that the content type of the response header must be set to "text/xml".

See how the PHP file will be returned from the server.

If you want to study PHP, you will find our PHP tutorial on our homepage.

Generating XML From a Database

XML can be generated from a database without any installed XML software.

To generate an XML database response from the server, simply write the following code and save it as an ASP file on the web server:

```
<%
response.ContentType = "text/xml"
set conn=Server.CreateObject("ADODB.Connection")
conn.provider="Microsoft.Jet.OLEDB.4.0;"
conn.open server.mappath("/db/database.mdb")

sql="select fname,lname from tblGuestBook"
set rs=Conn.Execute(sql)

response.write("<?xml version='1.0' encoding='ISO-8859-1'?>")
response.write("<guestbook>")
while (not rs.EOF)
response.write("<guest>")
response.write("<fname>" & rs("fname") & "</fname>")
response.write("<lname>" & rs("lname") & "</lname>")
response.write("</guest>")
rs.MoveNext()
wend

rs.close()
conn.close()
response.write("</guestbook>")
%>
```

See the real life database output from the ASP file above.

The example above uses ASP with ADO.

If you want to study ASP and ADO, you will find the tutorials on our homepage.

Transforming XML with XSLT on the Server

This ASP transforms an XML file to XHTML on the server:

```
<%
'Load XML
set xml = Server.CreateObject("Microsoft.XMLDOM")
xml.async = false
xml.load(Server.MapPath("simple.xml"))

'Load XSL
set xsl = Server.CreateObject("Microsoft.XMLDOM")
xsl.async = false
xsl.load(Server.MapPath("simple.xsl"))

'Transform file
Response.Write(xml.transformNode(xsl))
%>
```

Example explained

- The first block of code creates an instance of the Microsoft XML parser (XMLDOM), and loads the XML file into memory.
- The second block of code creates another instance of the parser and loads the XSL file into memory.
- The last line of code transforms the XML document using the XSL document, and sends the result as XHTML to your browser. Nice!

See how it works.

Saving XML To a File Using ASP

This ASP example creates a simple XML document and saves it on the server:

```
<%
text="<note>"
text=text & "<to>Tove</to>"
```

```
text=text & "<from>Jani</from>"
text=text & "<heading>Reminder</heading>"
text=text & "<body>Don't forget me this weekend!</body>"
text=text & "</note>"

set xmlDoc=Server.CreateObject("Microsoft.XMLDOM")
xmlDoc.async="false"
xmlDoc.loadXML(text)

xmlDoc.Save("test.xml")
%>
```

XML DOM Advanced

The XML DOM - Advanced

In an earlier chapter of this tutorial we introduced the XML DOM, and we used the getElementsByTagName() method to retrieve data from an XML document.

In this chapter we will explain some other important XML DOM methods.

You can learn more about the XML DOM in our XML DOM tutorial.

Get the Value of an Element

The XML file used in the examples below: books.xml.

The following example retrieves the text value of the first <title> element:

Example

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

Try it yourself »

Get the Value of an Attribute

The following example retrieves the text value of the "lang" attribute of the first <title> element:

Example

```
txt=xmlDoc.getElementsByTagName("title")[0].getAttribute("lang");
```

Try it yourself »

Change the Value of an Element

The following example changes the text value of the first <title> element:

Example

```
x=xmlDoc.getElementsByTagName("title")[0].childNodes[0];
x.nodeValue="Easy Cooking";
```

Try it yourself »

Create a New Attribute

The XML DOM setAttribute() method can be used to change the value of an existing attribute, or to create a new attribute.

The following example adds a new attribute (edition="first") to each <book> element:

Example

```
x=xmlDoc.getElementsByTagName("book");

for(i=0;i<x.length;i++)
  {
  x[i].setAttribute("edition","first");
  }
```

Try it yourself »

Create an Element

The XML DOM createElement() method creates a new element node.

The XML DOM createTextNode() method creates a new text node.

The XML DOM appendChild() method adds a child node to a node (after the last child).

To create a new element with text content, it is necessary to both create a new element node and a new text node, and then append it to an existing node.

The following example creates a new element (<edition>), with the following text: First, and adds it to the first <book> element:

Example

```
newel=xmlDoc.createElement("edition");
newtext=xmlDoc.createTextNode("First");
newel.appendChild(newtext);

x=xmlDoc.getElementsByTagName("book");
x[0].appendChild(newel);
```

Try it yourself »

Example explained:

- Create an <edition> element
- Create a text node with the following text: First
- Append the text node to the new <edition> element
- Append the <edition> element to the first <book> element

Remove an Element

The following example removes the first node in the first <book> element:

Example

```
x=xmlDoc.getElementsByTagName("book")[0];
x.removeChild(x.childNodes[0]);
```

Try it yourself »

Note: The result of the example above may be different depending on what browser you use. Firefox treats new lines as empty text nodes, Internet Explorer does not. You can read more about this and how to avoid it in our XML DOM tutorial.

XML Don't

Here are some technologies you should try to avoid when using XML.

Internet Explorer - XML Data Islands

What is it? An XML data island is XML data embedded into an HTML page.

Why avoid it? XML Data Islands only works with Internet Explorer browsers.

What to use instead? You should use JavaScript and XML DOM to parse and display XML in HTML.

For more information about JavaScript and XML DOM, visit our XML DOM tutorial.

XML Data Island Example

This example uses the XML document "cd_catalog.xml".

Bind the XML document to an <xml> tag in the HTML document. The id attribute defines an id for the data island, and the src attribute points to the XML file:

Example

This example only works in IE

```
<html>
<body>

<xml id="cdcat" src="cd_catalog.xml"></xml>

<table border="1" datasrc="#cdcat">
<tr>
<td><span datafld="ARTIST"></span></td>
<td><span datafld="TITLE"></span></td>
</tr>
</table>

</body>
</html>
```

Try it yourself »

The datasrc attribute of the <table> tag binds the HTML table to the XML data island.

The <span> tags allow the datafld attribute to refer to the XML element to be displayed. In this case, "ARTIST" and "TITLE". As the XML is read, additional rows are created for each <CD> element.

Internet Explorer - Behaviors

What is it? Internet Explorer 5 introduced behaviors. Behaviors are a way to add behaviors to XML (or HTML) elements with the use of CSS styles.

Why avoid it? The behavior attribute is only supported by Internet Explorer.

What to use instead? Use JavaScript and XML DOM (or HTML DOM) instead.

Example 1 - Mouseover Highlight

The following HTML file has a <style> element that defines a behavior for the <h1> element:

```
<html>
<head>
<style type="text/css">
h1 { behavior: url(behave.htc) }
</style>
</head>
<body>

<h1>Mouse over me!!!</h1>

</body>
</html>
```

The XML document "behave.htc" is shown below (The file contains a JavaScript and event handlers for the elements):

```
<attach for="element" event="onmouseover" handler="hig_lite" />
<attach for="element" event="onmouseout" handler="low_lite" />

<script type="text/javascript">
function hig_lite()
{
element.style.color='red';
}

function low_lite()
{
element.style.color='blue';
}
</script>
```

Try it yourself »

Example 2 - Typewriter Simulation

The following HTML file has a <style> element that defines a behavior for elements with an id of "typing":

```
<html>
<head>
<style type="text/css">
#typing
{
behavior:url(typing.htc);
font-family:'courier new';
}
```

```
</style>
</head>
<body>

<span id="typing" speed="100">IE5 introduced DHTML behaviors.
Behaviors are a way to add DHTML functionality to HTML elements
with the ease of CSS.<br /><br />How do behaviors work?<br />
By using XML we can link behaviors to any element in a web page
and manipulate that element.</p>v </span>

</body>
</html>
```

The XML document "typing.htc" is shown below:

```
<attach for="window" event="onload" handler="beginTyping" />
<method name="type" />

<script type="text/javascript">
var i,text1,text2,textLength,t;

function beginTyping()
{
i=0;
text1=element.innerText;
textLength=text1.length;
element.innerText="";
text2="";
t=window.setInterval(element.id+".type()",speed);
}

function type()
{
text2=text2+text1.substring(i,i+1);
element.innerText=text2;
i=i+1;
if (i==textLength)
  {
  clearInterval(t);
  }
}
</script>
```

Try it yourself »

XML Related Technologies

Below is a list of XML technologies.

XHTML (Extensible HTML)
A stricter and cleaner XML based version of HTML.

XML DOM (XML Document Object Model)
A standard document model for accessing and manipulating XML.

XSL (Extensible Style Sheet Language) XSL consists of three parts:

- XSLT (XSL Transform) - transforms XML into other formats, like HTML
- XSL-FO (XSL Formatting Objects)- for formatting XML to screen, paper, etc
- XPath - a language for navigating XML documents

XQuery (XML Query Language)
An XML based language for querying XML data.

DTD (Document Type Definition)
A standard for defining the legal elements in an XML document.

XSD (XML Schema)
An XML-based alternative to DTD.

XLink (XML Linking Language)
A language for creating hyperlinks in XML documents.

XPointer (XML Pointer Language)
Allows the XLink hyperlinks to point to more specific parts in the XML document.

XForms (XML Forms)
Uses XML to define form data.

SOAP (Simple Object Access Protocol)
An XML-based protocol to let applications exchange information over HTTP.

WSDL (Web Services Description Language)
An XML-based language for describing web services.

RDF (Resource Description Framework)
An XML-based language for describing web resources.

RSS (Really Simple Syndication)
A format for syndicating news and the content of news-like sites.

WAP (Wireless Application Protocol)
A XML based language for displaying content on wireless clients, like mobile phones.

SMIL (Synchronized Multimedia Integration Language)
A language for describing audiovisual presentations.

SVG (Scalable Vector Graphics)
Defines graphics in XML format.

XML in Real Life

Some examples of how XML can be used to exchange information.

Example: XML News

XMLNews is a specification for exchanging news and other information.

Using such a standard makes it easier for both news producers and news consumers to produce, receive, and archive any kind of news information across different hardware, software, and programming languages.

An example XMLNews document:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<nitf>
  <head>
    <title>Colombia Earthquake</title>
  </head>
  <body>
    <headline>
      <hl1>143 Dead in Colombia Earthquake</hl1>
    </headline>
    <byline>
```

```
      <bytag>By Jared Kotler, Associated Press Writer</bytag>
    </byline>
    <dateline>
      <location>Bogota, Colombia</location>
      <date>Monday January 25 1999 7:28 ET</date>
    </dateline>
  </body>
</nitf>
```

Example: XML Weather Service

An example of an XML national weather service from NOAA (National Oceanic and Atmospheric Administration):

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<current_observation>

<credit>NOAA's National Weather Service</credit>
<credit_URL>http://weather.gov/</credit_URL>

<image>
  <url>http://weather.gov/images/xml_logo.gif</url>
  <title>NOAA's National Weather Service</title>
  <link>http://weather.gov</link>
</image>

<location>New York/John F. Kennedy Intl Airport, NY</location>
<station_id>KJFK</station_id>
<latitude>40.66</latitude>
<longitude>-73.78</longitude>
<observation_time_rfc822>Mon, 11 Feb 2008 06:51:00 -0500 EST
</observation_time_rfc822>

<weather>A Few Clouds</weather>
<temp_f>11</temp_f>
<temp_c>-12</temp_c>
<relative_humidity>36</relative_humidity>
<wind_dir>West</wind_dir>
<wind_degrees>280</wind_degrees>
<wind_mph>18.4</wind_mph>
<wind_gust_mph>29</wind_gust_mph>
<pressure_mb>1023.6</pressure_mb>
<pressure_in>30.23</pressure_in>
<dewpoint_f>-11</dewpoint_f>
<dewpoint_c>-24</dewpoint_c>
<windchill_f>-7</windchill_f>
<windchill_c>-22</windchill_c>
<visibility_mi>10.00</visibility_mi>

<icon_url_base>http://weather.gov/weather/images/fcicons/</icon_url_base>
<icon_url_name>nfew.jpg</icon_url_name>
<disclaimer_url>http://weather.gov/disclaimer.html</disclaimer_url>
<copyright_url>http://weather.gov/disclaimer.html</copyright_url>

</current_observation>
```

XML Editors

If you are serious about XML, you will benefit from using a professional XML Editor.

XML is Text-based

XML is a text-based markup language.

One great thing about XML is that XML files can be created and edited using a simple text-editor like Notepad.

However, when you start working with XML, you will soon find that it is better to edit XML documents using a professional XML editor.

Why Not Notepad?

Many web developers use Notepad to edit both HTML and XML documents because Notepad is included with the most common OS and it is simple to use. Personally I often use Notepad for quick editing of simple HTML, CSS, and XML files.

But, if you use Notepad for XML editing, you will soon run into problems.

Notepad does not know that you are writing XML, so it will not be able to assist you.

Why an XML Editor?

Today XML is an important technology, and development projects use XML-based technologies like:

- XML Schema to define XML structures and data types
- XSLT to transform XML data
- SOAP to exchange XML data between applications
- WSDL to describe web services
- RDF to describe web resources
- XPath and XQuery to access XML data
- SMIL to define graphics

To be able to write error-free XML documents, you will need an intelligent XML editor!

XML Editors

Professional XML editors will help you to write error-free XML documents, validate your XML against a DTD or a schema, and force you to stick to a valid XML structure.

An XML editor should be able to:

- Add closing tags to your opening tags automatically
- Force you to write valid XML
- Verify your XML against a DTD
- Verify your XML against a Schema
- Color code your XML syntax

ALTOVA® xmlspy® 2010

At W3Schools we have been using XMLSpy for many years. XMLSpy is our favorite XML editor. These are some of the features we especially like:

- Now available in 32-bit and 64-bit versions
- Easy to use
- Context-sensitive entry helpers
- Syntax coloring and pretty printing
- Built in validation & well-formedness checking
- Easy switching between text view and grid view
- Graphical XML Schema editor
- Database import and export for all major databases
- SharePoint® Server support
- Built in templates for many XML document types
- Intelligent XPath 1.0/2.0 auto-completion
- XSLT 1.0/2.0 editor, profiler, and debugger
- XQuery editor, profiler, and debugger
- SOAP client and debugger
- Graphical WSDL 1.1/2.0 editor
- XBRL validation & taxonomy editing
- Support for Office 2007 / OOXML
- Code generation in Java, C++, and C#

Read more about XMLSpy

XMLSpy is just one of the six tools in the Altova MissionKit® XML software suite.
Read more about the Altova MissionKit for XML Developers.

XML Summary. What is Next?

XML Summary

XML can be used to exchange, share, and store data.

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

XML has very simple syntax rules. XML with correct syntax is "Well Formed". Valid XML also validates against a DTD.

XSLT is used to transform XML into other formats like HTML.

All modern browsers have a built-in XML parser that can read and manipulate XML.

The DOM (Document Object Model) defines a standard way for accessing XML.

The XMLHttpRequest object provides a way to communicate with a server after a web page has loaded.

XML Namespaces provide a method to avoid element name conflicts.

Text inside a CDATA section is ignored by the parser.

Our XML examples also represent a summary of this XML tutorial.

What to Study Next?

Our recommendation is to learn about the XML DOM and XSLT.

If you want to learn more about validating XML, we recommend DTD and XML Schema.

Below is a short description of each subject.

XML DOM (Document Object Model)

The XML DOM defines a standard way for accessing and manipulating XML documents.

The XML DOM is platform and language independent and can be used by any programming language like Java, JavaScript, and VBScript.

If you want to learn more about the DOM, please visit our XML DOM tutorial.

XSLT (XML Stylesheet Language Transformations)

XSLT is the style sheet language for XML files.

With XSLT you can transform XML documents into other formats, like XHTML.

If you want to learn more about XSLT, please visit our XSLT tutorial.

XML DTD (Document Type Definition)

The purpose of a DTD is to define what elements, attributes and entities is legal in an XML document.

With DTD, each of your XML files can carry a description of its own format with it.

DTD can be used to verify that the data you receive, and your own data, is valid.

If you want to learn more about DTD, please visit our DTD tutorial.

XML Schema

XML Schema is an XML based alternative to DTD.

Unlike DTD, XML Schemas has support for datatypes, and XML Schema use XML Syntax.

If you want to learn more about XML Schema, please visit our XML Schema tutorial.

XML Examples

These examples demonstrate XML files, XML formatting and XML transformation (XSLT).

They also demonstrate JavaScript used together with XML (AJAX).

Viewing XML Files

View a simple XML file (note.xml)
View the same XML file with an error
View an XML CD catalog
View an XML plant catalog
View an XML food menu

Examples explained

XML and CSS

View an XML CD catalog
View the corresponding CSS file
Display the CD catalog formatted with the CSS file

Examples explained

XML and XSLT

View an XML food menu
View the corresponding XSLT stylesheet
Display the food menu styled with the XSLT stylesheet

Examples explained

Parsing XML and the XML DOM

View a simple XML file (note.xml)
Parse the XML file - Crossbrowser example
Parse an XML string - Crossbrowser example

Examples explained

XML to HTML

View an XML CD catalog
Display XML data in an HTML table

Examples explained

XML Applications

View an XML CD catalog
Show XML data inside an HTML div element
Navigate through XML nodes
A simple CD catalog application

Examples explained

XML Output From a Server

See how ASP can return XML
See how PHP can return XML
View XML output from a database

Examples explained

XML DOM Advanced

Get the value of an XML element
Get the value of an XML attribute
Change the value of an XML element
Add a new attribute to an XML element
Create a new XML element
Remove an XML element

Examples explained

EXAMPLE

Belgian Waffles $5.95

two of our famous Belgian Waffles with plenty of real maple syrup 650 Strawberry Belgian Waffles $7.95

light Belgian waffles covered with strawberries and whipped cream 900 Berry-Berry Belgian Waffles $8.95

light Belgian waffles covered with an assortment of fresh berries and whipped cream 900

French Toast $4.50 thick slices made from our homemade sourdough bread 600

Homestyle Breakfast $6.95 two eggs, bacon or sausage, toast, and our ever-popular hash browns 950

```
<!-- Edited by
XMLSpy® -->
<breakfast_menu>
<food>
<name>Belgian Waffles</name>
<price>$5.95</price>
<description>two of our famous Belgian Waffles with plenty of real maple
syrup</description>
<calories>650</calories>
</food>
<food>
<name>Strawberry Belgian Waffles</name>
<price>$7.95</price>
<description>light Belgian waffles covered with strawberries and whipped
```

```
cream</description>
<calories>900</calories>
</food>
<food>
<name>Berry-Berry Belgian Waffles</name>
<price>$8.95</price>
<description>light Belgian waffles covered with an assortment of fresh
berries and whipped cream</description>
<calories>900</calories>
</food>
<food>
<name>French Toast</name>
<price>$4.50</price>
<description>thick slices made from our homemade sourdough
bread</description>
<calories>600</calories>
</food>
<food>
<name>Homestyle Breakfast</name>
<price>$6.95</price>
<description>two eggs, bacon or sausage, toast, and our ever-popular hash
browns</description>
<calories>950</calories>
</food>
</breakfast_menu>
```