XML Tutorial
 (***BASIC)***

XML stands for EXtensible Markup Language.
XML was designed to transport and store data.
In this tutorial you will learn about XML, and the difference between XML and HTML.
XML is important to know, and very easy to learn.

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 pure information wrapped in tags. Someone must write a piece of software to send, receive or display it.
XML is Just Plain Text
XML is nothing special. It is just plain text. Software that can handle plain text can also handle XML.
However, XML-aware applications can handle the XML tags specially. The functional meaning of the tags depends on the nature of the application.
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 (and the structure of 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 own tags and his 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.
XML is Everywhere
We have been participating in XML development since its creation. It has been amazing to see how quickly the XML standard has developed, and how quickly a large number of software vendors have adopted the standard.
XML is now as important for the Web as HTML was to the foundation of the Web.
XML is everywhere. It is the most common tool for data transmissions between all sorts of applications, and is becoming more and more popular in the area of storing and describing information.

How Can XML be Used?
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, you can read an external XML file and update the data content of your HTML.
You will learn more about this in a later chapter of this tutorial.
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 different applications can share.

XML Simplifies Data Transport
With XML, data can easily be exchanged between incompatible systems.
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 very 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
Since XML is independent of hardware, software and application, XML can make your data more available and useful.
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 the latest version of HTML
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 a pure text format, without any conversion utilities in between.
We can only pray that all the software vendors will agree.

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

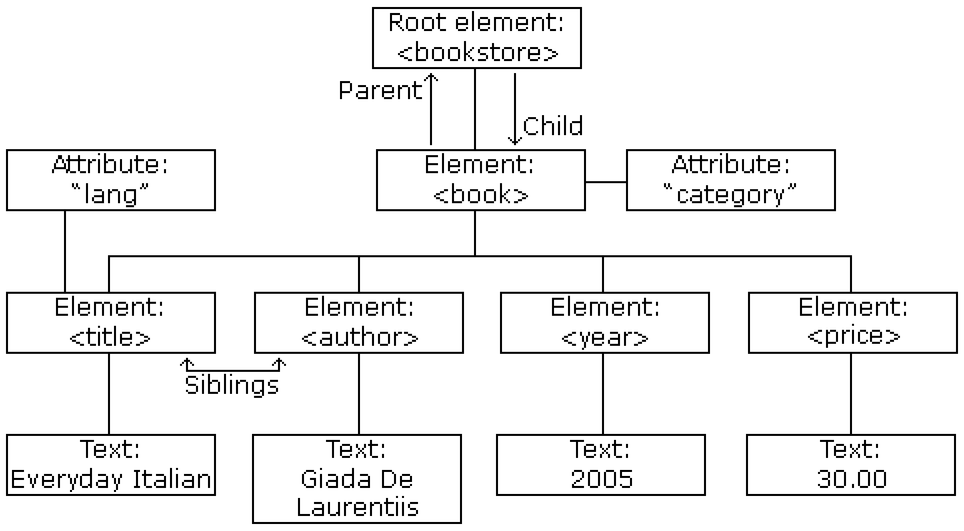
XML documents use a self-describing and simple syntax:<?xml version="1.0" encoding="ISO-8859-1"?>
<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
</note>

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>.

XML Syntax Rules
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, you will often see elements that don't 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 elements are defined using XML tags.
XML tags are case sensitive. With XML, 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 will often 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 value 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 -->
With XML, White Space is Preserved
HTML reduces multiple white space characters to a single white space:

HTML:	Hello my name is Tove
Output:	Hello my name is Tove.

With XML, the white space in your 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). The character pair bears some resemblance to the typewriter actions of setting a new line. In Unix applications, a new line is normally stored as a LF character. Macintosh applications use only a CR character to store a new line.
XML Elements

An XML document contains XML Elements.
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 often be extended without breaking applications.

XML Attributes
XML elements can have attributes in the start tag, just like HTML.
Attributes provide additional information about elements.
XML Attributes
From HTML you will remember this: <img src="computer.gif">. The "src" attribute provides additional information about the <img> element.
In HTML (and in XML) 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 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 enclosed in quotes, but either single or double quotes can be used. For a person's sex, the person tag 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 and 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 above is just an identifier, to identify 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 that data itself should be stored as elements.

XML Validation
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>
```

XML Validator

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.

Viewing XML Files

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 Netscape, 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"

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.

Displaying XML with CSS

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:

Sample XML file: The CD catalog
Empire Burlesque Bob Dylan USA Columbia 10.90 1985 Hide your heart Bonnie Tyler UK CBS Records 9.90 1988 Greatest Hits Dolly Parton USA RCA 9.90 1982 Still got the blues Gary Moore UK Virgin records 10.20 1990 Eros Eros Ramazzotti EU BMG 9.90 1997 One night only Bee Gees UK Polydor 10.90 1998 Sylvias Mother Dr.Hook UK CBS 8.10 1973 Maggie May Rod Stewart UK Pickwick 8.50 1990 Romanza Andrea Bocelli EU Polydor 10.80 1996 When a man loves a woman Percy Sledge USA Atlantic 8.70 1987 Black angel Savage Rose EU Mega 10.90 1995 1999 Grammy Nominees Many USA Grammy 10.20 1999 For the good times Kenny Rogers UK Mucik Master 8.70 1995 Big Willie style Will Smith USA Columbia 9.90 1997 Tupelo Honey Van Morrison UK Polydor 8.20 1971 Soulsville Jorn Hoel Norway WEA 7.90 1996 The very best of Cat Stevens UK Island 8.90 1990 Stop Sam Brown UK A and M 8.90 1988 Bridge of Spies T'Pau UK Siren 7.90 1987 Private Dancer Tina Turner UK Capitol 8.90 1983 Midt om natten Kim Larsen EU Medley 7.80 1983 Pavarotti Gala Concert Luciano Pavarotti UK DECCA 9.90 1991 The dock of the bay Otis Redding USA Atlantic 7.90 1987 Picture book Simply Red EU Elektra 7.20 1985 Red The Communards UK London 7.80 1987 Unchain my heart Joe Cocker USA EMI 8.20 1987

Sample style sheet: The CSS file
```
CATALOG
{
background-color: #ffffff;
width: 100%;
}
CD
{
display: block;
margin-bottom: 30pt;
margin-left: 0;
}
TITLE
{
color: #FF0000;
font-size: 20pt;
}
ARTIST
{
color: #0000FF;
font-size: 20pt;
}
COUNTRY,PRICE,YEAR,COMPANY
{
display: block;
color: #000000;
margin-left: 20pt;
}
```

SOURCE OF The CD catalog formatted with the CSS file:
http://w3schools.com/xml/cd_catalog_with_css.xml

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.

Displaying XML with XSLT
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.
One way to use XSLT is to transform XML into HTML before it is displayed by the browser as demonstrated in these examples:

XML FILE SAMPLE:
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

SEE RESULT ON THIS URL: *http://w3schools.com/xml/simplexsl.xml*

Below is a fraction of the XML file. The second line links the XML file to the XSLT file:<?xml version="1.0" encoding="ISO-8859-1"?>

```
<?xml-stylesheet type="text/xsl" href="simple.xsl"?>
<breakfast_menu>
  <food>
    <name>Belgian Waffles</name>
    <price>$5.95</price>
    <description>
      two of our famous Belgian Waffles
    </description>
    <calories>650</calories>
  </food>
</breakfast_menu>
```

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.
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.

XML JAVASCRIPT

XML Parser
Most browsers have a built-in XML parser to read and manipulate XML.
The parser converts XML into a JavaScript accessible object.
Parsing XML
All modern browsers have a built-in XML parser that can be used to read and manipulate XML.
The parser reads XML into memory and converts it into an XML DOM object that can be accessed with JavaScript.

You will learn more about the XML DOM in the next chapter of this tutorial.
There are some differences between Microsoft's XML parser and the parsers used in other browsers. The Microsoft parser supports loading of both XML files and XML strings (text), while other browsers use separate parsers. However, all parsers contain functions to traverse XML trees, access, insert, and delete nodes (elements) and their attributes.
In this tutorial we will show you how to create scripts that will work in both Internet Explorer and other browsers.
Note: When we talk about parsing XML, we often use the term "Nodes" about XML elements.
Loading XML with Microsoft's XML Parser

Microsoft's XML parser is built into Internet Explorer 5 and higher.
The following JavaScript fragment loads an XML document ("note.xml") into the parser:var
xmlDoc=new ActiveXObject("Microsoft.XMLDOM");
xmlDoc.async="false";
xmlDoc.load("note.xml");

Example explained:
The first line of the script above creates an empty Microsoft XML document object.
The second line turns off asynchronized loading, to make sure that the parser will not continue execution of the script before the document is fully loaded.
The third line tells the parser to load an XML document called "note.xml".

The following JavaScript fragment loads a string called txt into the parser:var xmlDoc=new
ActiveXObject("Microsoft.XMLDOM");
xmlDoc.async="false";
xmlDoc.loadXML(txt);

Note: The loadXML() method is used for loading strings (text), load() is used for loading files.

XML Parser in Firefox and Other Browsers
The following JavaScript fragment loads an XML document ("note.xml") into the parser:var
xmlDoc=document.implementation.createDocument("","",null);
xmlDoc.async="false";
xmlDoc.load("note.xml");

Example explained:
The first line of the script above creates an empty XML document object.
The second line turns off asynchronized loading, to make sure that the parser will not continue execution of the script before the document is fully loaded.
The third line tells the parser to load an XML document called "note.xml".
The following JavaScript fragment loads a string called txt into the parser:var parser=new
DOMParser();
var doc=parser.parseFromString(txt,"text/xml");

Example explained:
The first line of the script above creates an empty XML document object.
The second line tells the parser to load a string called txt.

Note: Internet Explorer uses the loadXML() method to parse an XML string, while other browsers uses 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 examples on W3Schools all open XML files located on the W3Schools domain.

If you want to use the example above on one of your web pages, the XML files you load must be located on your own server. Otherwise the xmlDoc.load() method, will generate the error "Access is denied".

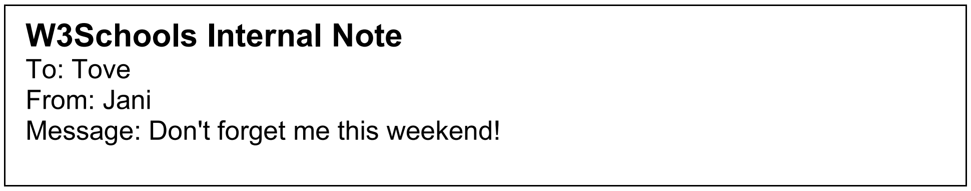
XML DOM
The DOM (Document Object Model) defines a standard way for accessing and manipulating documents.
The XML DOM
The XML DOM (XML Document Object Model) defines a standard way for accessing and manipulating XML documents.
The DOM views XML documents as a tree-structure. All elements can be accessed 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.

In the examples below we use the following DOM reference to get the text from the <to> element:
xmlDoc.getElementsByTagName("to")[0].childNodes[0].nodeValue
xmlDoc - the XML document created by the parser.
getElementsByTagName("to")[0] - the first <to> element
childNodes[0] - the first child of the <to> element (the text node)
nodeValue - the value of the node (the text itself)

The HTML DOM
The HTML DOM (HTML Document Object Model) defines a standard way for accessing and manipulating HTML documents.
All HTML elements can be accessed through the HTML DOM.
In the examples below we use the following DOM reference to change the text of the HTML element where id="to":
document.getElementById("to").innerHTML=
document - the HTML document
getElementById("to") - the HTML element where id="to"
innerHTML - the inner text of the HTML element

Parsing an XML File - A Cross browser Example
The following code loads an XML document ("note.xml") into the XML parser:
```
<html>
<head>
<script type="text/javascript">
function parseXML()
{
try //Internet Explorer
  {
  xmlDoc=new ActiveXObject("Microsoft.XMLDOM");
  }
catch(e)
  {
  try //Firefox, Mozilla, Opera, etc.
    {
    xmlDoc=document.implementation.createDocument("","",null);
    }
  catch(e)
    {
    alert(e.message);
    return;
    }
  }
xmlDoc.async=false;
xmlDoc.load("note.xml");

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>
</head>
<body onload="parseXML()">
<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></body></html>
```
Output:



W3Schools Internal Note
To: Tove
From: Jani
Message: Don't forget me this weekend!

Important Note
To extract the text "Jani" from the XML, the syntax is:getElementsByTagName("from")[0].childNodes[0].nodeValue
In the XML example there is only one <from> tag, but you still have to specify the array index [0], because the XML parser method getElementsByTagName() returns an array of all <from> nodes.

Parsing an XML String - A Cross browser Example
The following code loads and parses an XML string:
```
<html><head>
<script type="text/javascript">
function parseXML()
{
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>";
try //Internet Explorer
  {
  xmlDoc=new ActiveXObject("Microsoft.XMLDOM");
  xmlDoc.async="false";
  xmlDoc.loadXML(text);
  }
catch(e)
  {
  try // Firefox, Mozilla, Opera, etc.
    {
    parser=new DOMParser();
    xmlDoc=parser.parseFromString(text,"text/xml");
    }
  catch(e)
    {
    alert(e.message);
    return;
    }
  }
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></head>
<body onload="parseXML()">
<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>
</body></html>
```

XML to HTML

This chapter explains how to display XML data as HTML.

Display data as html table:

Example:

```
<html><body>
<script type="text/javascript">
var xmlDoc=null;
if (window.ActiveXObject)
{// code for IE
xmlDoc=new ActiveXObject("Microsoft.XMLDOM");}
else if (document.implementation.createDocument)
{// code for Mozilla, Firefox, Opera, etc.
xmlDoc=document.implementation.createDocument("","",null);}
else{
alert('Your browser cannot handle this script');}
if (xmlDoc!=null) {
xmlDoc.async=false;
xmlDoc.load("cd_catalog.xml");
var x=xmlDoc.getElementsByTagName("CD");
document.write("<table border='1'>");
document.write("<thead>");
document.write("<tr><th>Artist</th><th>Title</th></tr>");
document.write("</thead>");
document.write("<tfoot>");
document.write("<tr><th colspan='2'>This is my CD collection</th></tr>");
document.write("</tfoot>");
for (var i=0;i<x.length;i++)
{
document.write("<tr>");
document.write("<td>");
document.write(x[i].getElementsByTagName("ARTIST")[0].childNodes[0].nodeValue);
document.write("</td>");
document.write("<td>");
document.write(x[i].getElementsByTagName("TITLE")[0].childNodes[0].nodeValue);
document.write("</td>");
document.write("</tr>");}
document.write("</table>");}
</script>
</body></html>
```

SAMPLE OUTPUT:

Artist	Title
Bob Dylan	Empire Burlesque
Bonnie Tyler	Hide your heart
Dolly Parton	Greatest Hits
Gary Moore	Still got the blues
Eros Ramazzotti	Eros
Bee Gees	One night only
Dr.Hook	Sylvias Mother
Rod Stewart	Maggie May
Andrea Bocelli	Romanza
Percy Sledge	When a man loves a woman
Savage Rose	Black angel
Many	1999 Grammy Nominees
Kenny Rogers	For the good times
Will Smith	Big Willie style
Van Morrison	Tupelo Honey
Jorn Hoel	Soulsville
Cat Stevens	The very best of
Sam Brown	Stop
T'Pau	Bridge of Spies
Tina Turner	Private Dancer
Kim Larsen	Midt om natten
Luciano Pavarotti	Pavarotti Gala Concert
Otis Redding	The dock of the bay
Simply Red	Picture book
The Communards	Red
Joe Cocker	Unchain my heart
This is my CD collection	

Display XML Data in HTML

In the last chapter, we explained how to parse XML and access the DOM with JavaScript.
In this example, we loop through an XML file (cd_catalog.xml), and display each CD element as an HTML table row:

```
<html>
<body>

<script type="text/javascript">
var xmlDoc=null;
if (window.ActiveXObject)
{// code for IE
xmlDoc=new ActiveXObject("Microsoft.XMLDOM");
}
else if (document.implementation.createDocument)
{// code for Mozilla, Firefox, Opera, etc.
xmlDoc=document.implementation.createDocument("","",null);
}
else
{
alert('Your browser cannot handle this script');}
if (xmlDoc!=null){
xmlDoc.async=false;
xmlDoc.load("cd_catalog.xml");
document.write("<table border='1'>");
var x=xmlDoc.getElementsByTagName("CD");
for (i=0;i<x.length;i++){
document.write("<tr>");
document.write("<td>");
document.write(
x[i].getElementsByTagName("ARTIST")[0].childNodes[0].nodeValue);
document.write("</td>");
document.write("<td>");
document.write(
x[i].getElementsByTagName("TITLE")[0].childNodes[0].nodeValue);
document.write("</td>");
document.write("</tr>");}
document.write("</table>");}
</script></body></html>
```

Example explained
We check the browser, and load the XML using the correct parser
We create an HTML table with <table border="1">
We use getElementsByTagName() to get all XML CD nodes
For each CD node, we display data from ARTIST and TITLE as table data.
We end the table with </table>
For more information about using JavaScript and the XML DOM, visit our XML DOM tutorial.
Access Across Domains
For security reasons, modern browsers does 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 examples on W3Schools all open XML files located on the W3Schools domain.
If you want to use the example above on one of your web pages, the XML files you load must be located on your own server. Otherwise the xmlDoc.load() method, will generate the error "Access is denied".

The XMLHttpRequest Object

The XMLHttpRequest object provides a way to communicate with a server after a web page has loaded.
What is the XMLHttpRequest Object?
The XMLHttpRequest object is the developer's dream, because you can:
Update a web page with new data 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
The XMLHttpRequest object is supported in all modern browsers.

Creating an XMLHttpRequest Object
Creating an XMLHttpRequest object is done with one single line of JavaScript.
In all modern browsers (including IE7):var xmlhttp=new XMLHttpRequest()
In Internet Explorer 5 and 6:var xmlhttp=new ActiveXObject("Microsoft.XMLHTTP")
Example:

```
<script type="text/javascript">
var xmlhttp;
function loadXMLDoc(url)
{
xmlhttp=null;
if (window.XMLHttpRequest)
  {// code for all new browsers
  xmlhttp=new XMLHttpRequest();
  }
else if (window.ActiveXObject)
  {// code for IE5 and IE6
  xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");
  }
if (xmlhttp!=null)
  {
  xmlhttp.onreadystatechange=state_Change;
  xmlhttp.open("GET",url,true);
  xmlhttp.send(null);
  }
else
  {
  alert("Your browser does not support XMLHTTP.");
  }
}
function state_Change()
{
if (xmlhttp.readyState==4)
  {// 4 = "loaded"
  if (xmlhttp.status==200)
    {// 200 = OK
    // ...our code here...
    }
  else
    {
    alert("Problem retrieving XML data");
    }  }}</script>
```

Why Use Async=true?
Our examples use "true" in the third parameter of open().
This parameter specifies whether the request should be handled asynchronously.

True means that the script continues to run after the send() method, without waiting for a response from the server.
The onreadystatechange event complicates the code. But it is the safest way if you want to prevent the code from stopping if you don't get a response from the server.
By setting the parameter to "false", your can avoid the extra onreadystatechange code. Use this if it's not important to execute the rest of the code if the request fails.

Is the XMLHttpRequest Object a W3C Standard?
The XMLHttpRequest object is not specified in any W3C recommendation.
However, the W3C DOM Level 3 "Load and Save" specification contains some similar functionality, but these are not implemented in any browsers yet.

XML Application
This chapter demonstrates a small XML application built with HTML and JavaScript
The XML Example Document
Look at the following XML document ("cd_catalog.xml"), that represents a CD catalog:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<CATALOG>
  <CD>
    <TITLE>Empire Burlesque</TITLE>
    <ARTIST>Bob Dylan</ARTIST>
    <COUNTRY>USA</COUNTRY>
    <COMPANY>Columbia</COMPANY>
    <PRICE>10.90</PRICE>
    <YEAR>1985</YEAR>  </CD>
```

Load the XML Document

To load the XML document (cd_catalog.xml), we use the same code as we used in the XML Parser chapter:var xmlDoc;

```
if (window.ActiveXObject)
  {// code for IE
  xmlDoc=new ActiveXObject("Microsoft.XMLDOM");
  }
else if (document.implementation.createDocument)
  {// code for Firefox, Mozilla, Opera, etc.
  xmlDoc=document.implementation.createDocument("","",null);
  }
else
  {
  alert('Your browser cannot handle this script');
  }
xmlDoc.async=false;
xmlDoc.load("cd_catalog.xml");
```

After the execution of this code, xmlDoc is an XML DOM object, accessible by JavaScript.

Display XML Data as an HTML Table
The following code displays an HTML table filled with data from the XML DOM object:document.write("<table border='1'>");

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

For each CD element in the XML document, a table row is created. Each table row contains two table data cells with ARTIST and TITLE data from the current CD element.

Display XML Data in any HTML Element

XML data can be copied into any HTML element that can display text.
The code below is part of the <head> section of the HTML file. It gets the XML data from the first <CD> element and displays it in the HTML element with the id="show":var

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

function display()
{
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("show").innerHTML=txt;
}
```

The body of the HTML document contains an onload eventattribute that will call the display() function when the page has loaded. It also contains a <div id='show'> element to receive the XML data.<body onload="display()">

```
<div id='show'></div>
</body>
```

With the example above, you will only see data from the first CD element in the XML document.
To navigate to the next line of data, you have to add some more code.
Add a Navigation Script

To add navigation to the example above, create two functions called next() and previous():function next(){

```
if (i<x.length-1)
  {
  i++;
  display();
  }}
function previous()
{
if (i>0)
  {
  i--;
  display();
  }}
```

All Together Now
With a little creativity you can create a full application.
If you use what you have learned on this page, and a little imagination, you can easily develop this into a full application.
The next() function makes sure that nothing is displayed if you already are at the last CD element, and the previous () function makes sure that nothing is displayed if you already are at the first CD element.
The next() and previous() functions are called by clicking next/previous buttons:<input type="button" onclick="previous()" value="previous" />
<input type="button" onclick="next()" value="next" />

XML ADVANCED
XML Namespaces
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>
```

XML CDATA
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 Encoding
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.
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.

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.

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>")
%>
```

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 (Document Object Model) defines a standard way for accessing and manipulating XML documents.
The XML DOM
The DOM views XML documents 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.
In an earlier chapter of this tutorial we introduced the XML DOM , and used the XML DOM getElementsByTagName() method to retrieve data from a DOM tree.
In this chapter we will describe some other commonly used XML DOM methods. In the examples, we have used the XML file books.xml, and a JavaScript function to load the XML file into an DOM object called xmlDoc.

Get the Value of an Element
The following code retrieves the text value of the first <title> element:x=xmlDoc.getElementsByTagName("title")[0].childNodes[0];
txt=x.nodeValue;
Result: txt = "Everyday Italian"

Get the Value of an Attribute
The following code retrieves the text value of the "lang" attribute of the first <title> element:txt=xmlDoc.getElementsByTagName("title")[0].getAttribute("lang");
Result: txt = "en"

Change the Value of an Element
The following code changes the text value of the first <title>
element:x=xmlDoc.getElementsByTagName("title")[0].childNodes[0];
x.nodeValue="Easy Cooking";

Change the Value of an Attribute
The setAttribute() method can be used to change the value of an existing attribute, or to create a new attribute.
The following code adds a new attribute called "edition" (with the value "first") to each <book>
element:x=xmlDoc.getElementsByTagName("book");
for(i=0;i<x.length;i++) {
  x[i].setAttribute("edition","first"); }

Create an Element
The createElement() method creates a new element node.
The createTextNode() method creates a new text node.
The 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 create both an element node and a text node.
The following code creates an element (<edition>), and adds it to the first <book>
element:newel=xmlDoc.createElement("edition");
newtext=xmlDoc.createTextNode("First");
newel.appendChild(newtext);
x=xmlDoc.getElementsByTagName("book");
x[0].appendChild(newel);

Example explained:
Create an <edition> element
Create a text node with value = "First"
Append the text node to the <edition> element
Append the <edition> element to the first <book> element

Remove an Element
The removeChild() method removes a specified node (or element).
The following code fragment will remove the first node in the first <book>
element:x=xmlDoc.getElementsByTagName("book")[0];
x.removeChild(x.childNodes[0]);
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 don't. You can read more about this and how to avoid it in the 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.
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:<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>
```
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:<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>
```
The behavior file contains a JavaScript and event handlers for the elements.

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>
</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>
```

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>
<two_day_history_url>
http://www.weather.gov/data/obhistory/KJFK.html
</two_day_history_url>
<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®

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:
Easy to use
Automatic tag completion
Context-sensitive entry helpers
Automatic well-formedness checking
Syntax coloring and pretty printing
Built in DTD and/or XML Schema-based validation
Easy switching between text view and grid view
Built in graphical XML Schema editor
Powerful conversion utilities
Database import and export
Built in templates for many XML document types
Built in XPath 1.0/2.0 analyzer
XSLT 1.0/2.0 editor, profiler, and debugger
XQuery editor, profiler, and debugger
SOAP client and debugger
Graphical WSDL editor
Powerful project management capabilities
Code generation in Java, C++, and C#
Support for Office 2007 / OOXML

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 build-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.

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.
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.

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.

By: DataIntegratedEntity22592
Source: http://w3schools.com/xml/xml_whatis.asp