

# Introduction to XML

XML was designed to describe data and to focus on what data is.

HTML was designed to display data and to focus on how data looks.

## 1 What you should already know

Before you continue, you should have some basic understanding of the following:

- WWW, HTML and the basics of building Web pages
- Web scripting languages like JavaScript or VBScript

If you want to study these subjects first, before you start reading about XML, you can find the tutorials you need at [W3Schools' Home Page](#).

## 2 What is XML?

- XML stands for EXtensible Markup Language
- XML is a **markup language** much like HTML
- XML was designed to **describe data**
- XML tags are not predefined. You must **define your own tags**
- XML uses a **Document Type Definition (DTD)** or an **XML Schema** to describe the data
- XML with a DTD or XML Schema is designed to be **self-descriptive**

## 3 The main difference between XML and HTML

**XML was designed to carry data.**

XML is not a replacement for HTML.

XML and HTML were designed with different goals:

XML was designed to describe data and to focus on what data is.

HTML was designed to display data and to focus on how data looks.

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

## 4 XML does not DO anything

**XML was not designed to DO anything.**

Maybe it is a little hard to understand, but XML does not DO anything. XML was created to structure, store and to send 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 has a header and a message body. It also has sender and receiver information. But still, this XML document does not DO anything. It is just pure information wrapped in XML tags. Someone must write a piece of software to send, receive or display it.

## 5 XML is free and extensible

**XML tags are not predefined. You must "invent" your own tags.**

The tags used to mark up HTML documents and the structure of HTML documents are predefined. The author of HTML documents can only use tags that are defined in the HTML standard (like <p>, <h1>, etc.).

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

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.

## 6 XML is a complement to HTML

**XML is not a replacement for HTML.**

It is important to understand that XML is not a replacement for HTML. In future Web development it is most likely that XML will be used to describe the data, while HTML will be used to format and display the same data.

My best description of XML is this: **XML is a cross-platform, software and hardware independent tool for transmitting information.**

## 7 XML in future Web development

**XML is going to be everywhere.**

We have been participating in XML development since its creation. It has been amazing to see how quickly the XML standard has been developed and how quickly a large number of software vendors have adopted the standard.

We strongly believe that XML will be as important to the future of the Web as HTML has been to the foundation of the Web and that XML will be the most common tool for all data manipulation and data transmission.

## 8 XML Joke

Question: When should I use XML?

Answer: When you need a buzzword in your resume.

# How can XML be Used?

It is important to understand that XML was designed to store, carry, and exchange data. XML was not designed to display data.

## 8.1 XML can Separate Data from HTML

**With XML, your data is stored outside your HTML.**

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

XML data can also be stored inside HTML pages as "Data Islands". You can still concentrate on using HTML only for formatting and displaying the data.

## 8.2 XML is used to Exchange Data

**With XML, data can be exchanged between incompatible systems.**

In the real world, computer systems and databases contain data in incompatible formats. One of the most time-consuming challenges for developers has been to exchange data between such systems over the Internet.

Converting the data to XML can greatly reduce this complexity and create data that can be read by many different types of applications.

## 8.3 XML and B2B

**With XML, financial information can be exchanged over the Internet.**

Expect to see a lot about XML and B2B (Business To Business) in the near future.

XML is going to be the main language for exchanging financial information between businesses over the Internet. A lot of interesting B2B applications are under development.

## 8.4 XML can be used to Share Data

**With XML, plain text files can be used to share data.**

Since XML data is stored in plain text format, XML provides a software- and hardware-independent way of sharing data.

This makes it much easier to create data that different applications can work with. It also makes it easier to expand or upgrade a system to new operating systems, servers, applications, and new browsers.

## **8.5 XML can be used to Store Data**

**With XML, plain text files can be used to store data.**

XML can also be used to store data in files or in databases. Applications can be written to store and retrieve information from the store, and generic applications can be used to display the data.

## **8.6 XML can make your Data more Useful**

**With XML, your data is available to more users.**

Since XML is independent of hardware, software and application, you can make your data available to other than only standard HTML browsers.

Other clients and applications can access your XML files as data sources, like they are accessing databases. Your data can be made available to all kinds of "reading machines" (agents), and it is easier to make your data available for blind people, or people with other disabilities.

## **8.7 XML can be used to Create new Languages**

**XML is the mother of WAP and WML.**

The Wireless Markup Language (WML), used to markup Internet applications for handheld devices like mobile phones, is written in XML.

You can read more about WML in our [WML tutorial](#).

## **8.8 If Developers have Sense**

**If they DO have sense, all 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 Microsoft and all the other software vendors will agree.

# XML Syntax

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

Because of this, creating software that can read and manipulate XML is very easy to do.

## 8.9 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 in the document - the XML declaration - defines the XML version and the character encoding used in the document. In this case the document conforms to the 1.0 specification of XML and uses the ISO-8859-1 (Latin-1/West European) character set.

The next line describes the root element of the document (like it was 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>
```

Can you detect from this example that the XML document contains a Note to Tove from Jani? Don't you agree that XML is pretty self-descriptive?

## 8.10 All XML elements must have a closing tag

**With XML, it is illegal to omit the closing tag.**

In HTML some elements do not have to have a closing tag. The following code is legal in HTML:

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

In XML all elements must have a closing tag, like this:

```
<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. It is not an XML element, and it should not have a closing tag.

## 8.11 XML tags are case sensitive

**Unlike HTML, XML tags are case sensitive.**

With XML, the tag <Letter> is different from the tag <letter>.

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

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

## 8.12 All XML elements must be properly nested

**Improper nesting of tags makes no sense to XML.**

In HTML some elements can be improperly nested within each other like this:

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

In XML all elements must be properly nested within each other like this:

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

## 8.13 All XML documents must have a root element

**All XML documents must contain a single tag pair to define a root element.**

All other elements must be within this root element.

All elements can have sub elements (child elements). Sub elements must be correctly nested within their parent element:

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

---

## 8.14 Attribute values must always be quoted

**With XML, it is illegal to omit quotation marks around attribute values.**

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:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<note date=12/11/2002>
<to>Tove</to>
<from>Jani</from>
</note>
```

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<note date="12/11/2002">
<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.

This is correct: date="12/11/2002". This is incorrect: date=12/11/2002.

## 8.15 With XML, white space is preserved

**With XML, the white space in your document is not truncated.**

This is unlike HTML. With HTML, a sentence like this:

Hello        my name is Tove,

will be displayed like this:

Hello my name is Tove,

because HTML reduces multiple, consecutive white space characters to a single white space.

## 8.16 With XML, CR / LF is converted to LF

**With XML, a new line is always stored as LF.**

Do you know what a typewriter is? Well, a typewriter is a mechanical device which was used last century to produce printed documents. :-)

After you have typed one line of text on a typewriter, you have to manually return the printing carriage to the left margin position and manually feed the paper up one line.

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.

## **8.17 Comments in XML**

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

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

## **8.18 There is nothing special about XML**

There is nothing special about XML. It is just plain text with the addition of some XML tags enclosed in angle brackets.

Software that can handle plain text can also handle XML. In a simple text editor, the XML tags will be visible and will not be handled specially.

In an XML-aware application however, the XML tags can be handled specially. The tags may or may not be visible, or have a functional meaning, depending on the nature of the application.

# XML Elements

XML Elements are extensible and they have relationships.

XML Elements have simple naming rules.

## 8.19 XML Elements are Extensible

**XML documents can be extended to carry more information.**

Look at the following XML NOTE 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>2002-08-01</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.

**XML documents are Extensible.**

## 8.20 XML Elements have Relationships

**Elements are related as parents and children.**

To understand XML terminology, you have to know how relationships between XML elements are named, and how element content is described.

Imagine that this is a description of a book:

## My First XML

### Introduction to XML

- What is HTML
- What is XML

### XML Syntax

- Elements must have a closing tag
- Elements must be properly nested

Imagine that this XML document describes the book:

```
<book>
<title>My First XML</title>
<prod id="33-657" media="paper"></prod>
<chapter>Introduction to XML
<para>What is HTML</para>
<para>What is XML</para>
</chapter>

<chapter>XML Syntax
<para>Elements must have a closing tag</para>
<para>Elements must be properly nested</para>
</chapter>

</book>
```

Book is the **root element**. Title, prod, and chapter are **child elements** of book. Book is the **parent element** of title, prod, and chapter. Title, prod, and chapter are **siblings** (or **sister elements**) because they have the same parent.

## 8.21 Elements have Content

**Elements can have different content types.**

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

An element can have **element** content, **mixed** content, **simple** content, or **empty** content. An element can also have **attributes**.

In the example above, book has **element content**, because it contains other elements. Chapter has **mixed content** because it contains both text and other elements. Para has **simple content** (or **text content**) because it contains only text. Prod has **empty content**, because it carries no information.

In the example above only the prod element has **attributes**. The **attribute** named id has the **value** "33-657". The **attribute** named media has the **value** "paper".

## 8.22 Element Naming

### XML elements must follow these naming rules:

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

Take care when you "invent" element names and follow these simple rules:

Any name can be used, no words are reserved, but the idea is to make names descriptive. Names with an underscore separator are nice.

Examples: <first\_name>, <last\_name>.

Avoid "-" and "." in names. For example, if you name something "first-name," it could be a mess if your software tries to subtract name from first. Or if you name something "first.name," your software may think that "name" is a property of the object "first."

Element names can be as long as you like, but don't exaggerate. Names should be short and simple, like this: <book\_title> not like this: <the\_title\_of\_the\_book>.

XML documents often have a corresponding database, in which fields exist corresponding to elements in the XML document. 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 element names, but watch out for problems if your software vendor doesn't support them.

The ":" should not be used in element names because it is reserved to be used for something called namespaces (more later).

# XML Attributes

XML elements can have attributes in the start tag, just like HTML.

Attributes are used to provide additional information about elements.

## 8.23 XML Attributes

**XML elements can have 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:

```
  
<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>
```

## 8.24 Quote Styles, "female" or 'female'?

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'>
```

**Note:** If the attribute value itself contains double quotes it is necessary to use single quotes, like in this example:

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

**Note:** If the attribute value itself contains single quotes it is necessary to use double quotes, like in this example:

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

## 8.25 Use of Elements vs. Attributes

Data can be stored in child elements or in 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 a child element. Both examples provide the same information.

There are no rules about when to use attributes, and when to use child elements. My experience is that attributes are handy in HTML, but in XML you should try to avoid them. Use child elements if the information feels like data.

## 8.26 My Favorite Way

I like to store data in child elements.

The following three XML documents contain exactly the same information:

A date attribute is used in the first example:

```
<note date="12/11/2002">
<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>12/11/2002</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>12</day>
  <month>11</month>
  <year>2002</year>
</date>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
</note>
```

## 8.27 Avoid using attributes?

**Should you avoid using attributes?**

Some of the problems with using attributes are:

- attributes cannot contain multiple values (child elements can)
- attributes are not easily expandable (for future changes)
- attributes cannot describe structures (child elements can)
- attributes are more difficult to manipulate by program code
- attribute values are not easy to test against a Document Type Definition (DTD) - which is used to define the legal elements of an XML document

If you use attributes as containers for data, you end up with documents that are difficult to read and maintain. Try to use **elements** to describe data. Use attributes only to provide information that is not relevant to the data.

Don't end up like this ( if you think this looks like XML, you have not understood the point):

```
<note day="12" month="11" year="2002"
to="Tove" from="Jani" heading="Reminder"
body="Don't forget me this weekend!">
</note>
```

## 8.28 An Exception to my Attribute rule

**Rules always have exceptions.**

My rule about attributes has one exception:

Sometimes I assign ID references to elements. These ID references can be used to access XML elements in much the same way as the NAME or ID attributes in HTML. This example demonstrates this:

```
<messages>
  <note id="p501">
    <to>Tove</to>
    <from>Jani</from>
    <heading>Reminder</heading>
    <body>Don't forget me this weekend!</body>
  </note>

  <note id="p502">
    <to>Jani</to>
    <from>Tove</from>
    <heading>Re: Reminder</heading>
    <body>I will not!</body>
  </note>
</messages>
```

The ID in these examples is just a counter, or a unique identifier, to identify the different notes in the XML file, and not a part of the note data.

What I am 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.

## 8.29 "Well Formed" XML documents

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

A "Well Formed" XML document is a document that conforms to the XML syntax rules that were described in the previous chapters:

```
<?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>
```

## 8.30 "Valid" XML documents

**A "Valid" XML document also conforms to a DTD.**

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

## 8.31 XML DTD

**A DTD defines the legal elements of an XML document.**

The purpose of a DTD is to define the legal building blocks of an XML document. It defines the document structure with a list of legal elements. You can read more about DTD, and how to validate your XML documents in our [DTD tutorial](#).

## 8.32 XML Schema

**XML Schema is an XML based alternative to DTD.**

W3C supports an alternative to DTD called XML Schema. You can read more about XML Schema in our [Schema tutorial](#).

## 8.33 Errors will Stop you

**Errors in XML documents will stop the XML program.**

The W3C XML specification states that a program should not continue to process an XML document if it finds a validation error. The reason is that XML software should be easy to write, and that all XML documents should be compatible.

With HTML it was possible to create documents with lots of errors (like when you forget an end tag). One of the main reasons that HTML browsers are so big and incompatible, is that they have their own ways to figure out what a document should look like when they encounter an HTML error.

With XML this should not be possible.

## 8.34 A general XML Validator

To help you validate your xml files, we have created this link so that you can [Validate any XML file](#).

# XML Browser Support

In our XML tutorial we have focused on Internet Explorer 5.0+.

## 8.35 XML on this Web Site

**Many browsers support XML. We are focusing on Internet Explorer 5.0+.**

Some of you have complained about this, but we think this is the most practical way to demonstrate real XML examples for you over the Internet.

## 8.36 XML in Mozilla Firefox 1.0

Mozilla Firefox 1.0 supports the XML 1.0 standard.

## 8.37 XML in Netscape 6

Netscape 6 supports XML.

To look at the XML source in Netscape 6: right-click on the page and select "View Page Source".

## 8.38 XML in Internet Explorer 5.0

Internet Explorer 5.0 supports the XML 1.0 standard.

Internet Explorer 5.0 supports most of the international standards for XML 1.0 and the XML DOM (Document Object Model). These standards are set by the World Wide Web Consortium (W3C).

Internet Explorer 5.0 has the following XML support:

- Viewing of XML documents
- Full support for W3C DTD standards
- XML embedded in HTML as Data Islands
- Binding XML data to HTML elements
- Transforming and displaying XML with XSL
- Displaying XML with CSS
- Access to the XML DOM

Internet Explorer 5.0 also has support for Behaviors:

- Behaviors is a Microsoft-only technology
- Behaviors can separate scripts from an HTML page.
- Behaviors can store XML data on the client's disk.

Examples of all these features are given in the next chapters of this web site.

# Viewing XML Files

Raw XML files can be viewed in Mozilla Firefox, IE 5.0+ and in Netscape 6.

However, to make XML documents to display like nice web pages, you will have to add some display information.

## 8.39 Viewing XML Files

### In Mozilla Firefox and IE 5.0+:

Click on a link to an XML file, type the direct URL in the address bar, or double-click on the name of an XML file in a folder. 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.

### In Netscape 6:

Open the XML file (via a link or URL), then right-click in XML file and select "View Page Source". The XML document will then be displayed with color-coded root and child elements.

Look at this XML file: [note.xml](#)

**Note: Do not expect XML files to be formatted like HTML documents!**

## 8.40 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](#)

## 8.41 Other XML Examples

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

### [An XML CD catalog](#)

This is my father's CD collection, stored as XML data (old and boring titles I think.... :-)).

### [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.

## 8.42 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, XSL, JavaScript, and XML Data Islands.

# Displaying XML with CSS

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

## 8.43 Displaying your future XML files with CSS?

Would you use CSS to format your future XML files? No, we don't think so! But we could not resist giving it a try:

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, `<?xml-stylesheet type="text/css" href="cd_catalog.css"?>`, 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>
```

We DO NOT believe that formatting XML with CSS is the future of the Web. Even if it looks right to use CSS this way, we DO believe that formatting with XSL will be the new standard (as soon as all main browsers support it).

## 8.44 Creating your future Homepages with XML?

Will you be writing your future Homepages in XML? No, we don't think you will! But we could not resist giving it a try : [A homepage written in XML](#).

We DO NOT believe that XML will be used to create future Homepages.

# Displaying XML with XSL

With XSL you can add display information to your XML document.

## 8.45 Displaying XML with XSL

**XSL is the preferred style sheet language of XML.**

XSL (the eXtensible Stylesheet Language) is far more sophisticated than CSS. One way to use XSL is to transform XML into HTML before it is displayed by the browser as demonstrated in these examples:

If you have Netscape 6 or IE 5 or higher you can view [the XML file](#) and [the XSL style sheet](#).

[View the result in IE 6](#)

[View the result in IE 5](#)

Below is a fraction of the XML file, with an added XSL reference. The second line, `<?xml-stylesheet type="text/xsl" href="simple.xsl"?>`, links the XML file to the XSL 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 XSL, please visit our [XSL tutorial](#).

# XML in Data Islands

With Internet Explorer 5.0 and higher, XML can be embedded within HTML pages in Data Islands.

## 8.46 XML Embedded in HTML

**The unofficial <xml> tag is used to embed XML data within HTML.**

XML data can be embedded directly in an HTML page like this:

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

Or a separate XML file can be embedded like this:

```
<xml id="note" src="note.xml">
</xml>
```

**Note that the <xml> tag is an HTML element, not an XML element.**

## 8.47 Data Binding

**Data Islands can be bound to HTML elements (like HTML tables).**

In the example below, an XML Data Island with an ID "cdcat" is loaded from an external XML file. An HTML table is bound to the Data Island with a data source attribute, and finally the tabledata elements are bound to the XML data with a data field attribute inside a span.

```
<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>
```

If you are running IE 5.0 or higher, you can [try it yourself](#).  
With IE 5.0 and higher you can also view the [external XML file](#).  
Also try [this example](#), demonstrating <thead>, <tbody>, and <tfoot>.

# The Microsoft XML Parser

To read and update - create and manipulate - an XML document, you need an XML parser.

## 8.48 Using the XML parser

**The Microsoft XML parser comes with Microsoft Internet Explorer 5.0.**

Once you have installed IE 5.0, the parser is available to scripts, both inside HTML documents and inside ASP files. The parser features a language-neutral programming model that supports:

- JavaScript, VBScript, Perl, VB, Java, C++ and more
- W3C XML 1.0 and XML DOM
- DTD and validation

If you are using JavaScript in IE 5.0, you can create an XML document object with the following code:

```
var xmlDoc=new ActiveXObject("Microsoft.XMLDOM")
```

If you are using VBScript, you create the XML document object with the following code:

```
set xmlDoc=CreateObject("Microsoft.XMLDOM")
```

If you are using VBScript in ASP, you can use the following code:

```
set xmlDoc=Server.CreateObject("Microsoft.XMLDOM")
```

## 8.49 Loading an XML file into the parser

**XML files can be loaded into the parser using script code.**

The following code loads an XML document (note.xml) into the XML parser:

```
<script type="text/javascript">
var xmlDoc = new ActiveXObject("Microsoft.XMLDOM")
xmlDoc.async="false"
xmlDoc.load("note.xml")
// ..... processing the document goes here
</script>
```

The second line in the code above creates an instance of the Microsoft XML parser.

The third line turns off asynchronous loading, to make sure that the parser will not continue execution before the document is fully loaded.

The fourth line tells the parser to load the XML document called note.xml.

## 8.50 Loading pure XML text into the parser

**XML text can also be loaded from a text string.**

The following code loads a text string into the XML parser:

```
<script type="text/javascript">
var text="<note>"
text=text+"<to>Tove</to><from>Jani</from>"
text=text+"<heading>Reminder</heading>"
text=text+"<body>Don't forget me this weekend!</body>"
text=text+"</note>"

var xmlDoc = new ActiveXObject("Microsoft.XMLDOM")
xmlDoc.async="false"
xmlDoc.loadXML(text)
// ..... processing the document goes here
</script>
```

Note that the "loadXML" method (instead of the "load" method) is used to load a text string.

## 8.51 Displaying XML with JavaScript

**To display XML you can use JavaScript.**

JavaScript (or VBScript) can be used to import data from an XML file and display the XML data inside an HTML page.

To see how XML and HTML complement each other this way; first look at the **XML** document ([note.xml](#)), then open the **HTML** document ([note.htm](#)) that contains a JavaScript which reads the XML file and displays the information inside predefined spans in the HTML page.

To see how it works, [Try It Yourself](#)

You can see a lot more of this kind of JavaScript in our [DOM School](#).

# XML in Real Life

Some real-life examples of how XML can be used to carry information.

## 8.52 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.

## 8.53 An example XML News document:

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

<nitf>

<head>
<title>Colombia Earthquake</title>
</head>

<body>

<body.head>
<headline>
<h1>143 Dead in Colombia Earthquake</h1>
</headline>
<byline>
<bytag>By Jared Kotler, Associated Press Writer</bytag>
</byline>
<dateline>
<location>Bogota, Colombia</location>
<story.date>Monday January 25 1999 7:28 ET</story.date>
</dateline>
</body.head>

</body>

</nitf>
```

# XML Namespaces

XML Namespaces provide a method to avoid element name conflicts.

## 8.54 Name Conflicts

Since element names in XML are not fixed, very often a name conflict will occur when two different documents use the same names describing two different types of elements.

This XML document carries information in a table:

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

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

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

If these two XML documents were added together, there would be an element name conflict because both documents contain a `<table>` element with different content and definition.

## 8.55 Solving Name Conflicts using a Prefix

This XML document carries information in a table:

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

This XML document carries information about a piece of furniture:

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

Now the element name conflict is gone because the two documents use a different name for their `<table>` element (`<h:table>` and `<f:table>`).

By using a prefix, we have created two different types of `<table>` elements.

## 8.56 Using Namespaces

This XML document carries information in a table:

```
<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>
```

This XML document carries information about a piece of furniture:

```
<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>
```

Instead of using only prefixes, an **xmlns attribute** has been added to the <table> tag to give the element prefix a **qualified name** associated with a **namespace**.

## 8.57 The Namespace Attribute

The namespace attribute is placed in the start tag of an element and has the following syntax:

```
xmlns:namespace-prefix="namespace"
```

In the examples above, the namespace itself is defined using an Internet address:

```
xmlns:f="http://www.w3schools.com/furniture"
```

The W3C namespace specification states that the namespace itself should be a **Uniform Resource Identifier (URI)**.

When a namespace is defined in the start tag of an element, all child elements with the same prefix are associated with the same namespace.

Note that the address used to identify the namespace, is not used by the parser to look up information. The only purpose is to give the namespace a unique name. However, very often companies use the namespace as a pointer to a real Web page containing information about the namespace.

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

## 8.58 Uniform Resource Identifiers

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.

Since our furniture example uses an internet address to identify its namespace, we can be sure that our namespace is unique.

## 8.59 Default Namespaces

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

```
<element xmlns="namespace">
```

This XML document carries information in a table:

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

This XML document 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>
```

## 8.60 Namespaces in Real Use

When you start using XSL, you will soon see namespaces in real use. XSL style sheets are used to transform XML documents into other formats like HTML.

If you take a close look at the XSL document below, you will see that most of the tags are HTML tags. The tags that are not HTML tags have the prefix `xsl`, identified by the namespace "<http://www.w3.org/TR/xsl>":

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/xsl">
<xsl:template match="/">
  <html>
  <body>
    <table border="2" bgcolor="yellow">
      <tr>
        <th>Title</th>
        <th>Artist</th>
      </tr>
      <xsl:for-each select="CATALOG/CD">
      <tr>
        <td><xsl:value-of select="TITLE"/></td>
        <td><xsl:value-of select="ARTIST"/></td>
      </tr>
      </xsl:for-each>
    </table>
  </body>
</html>
</xsl:template>
</xsl:stylesheet>
```

# XML CDATA

All text in an XML document will be **parsed** by the parser. Only text inside a CDATA section is **ignored** by the parser.

## 8.61 Parsed 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>
```

## 8.62 Escape Characters

**Illegal XML characters have to be replaced by entity references.**

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. You cannot write something like this:

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

To avoid this, you have to **replace** the "<" character with an **entity reference**, like this:

```
<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

Entity references always start with the "&" character and end with the ";" character.

**Note:** Only the characters "<" and "&" are strictly illegal in XML. Apostrophes, quotation marks and greater than signs are legal, but it is a good habit to replace them.

## 8.63 CDATA

**Everything inside a CDATA section is ignored by the parser.**

If your text contains a lot of "<" or "&" characters - as program code often does - the XML element can be defined as a CDATA section.

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

### **Notes on CDATA sections:**

A CDATA section cannot contain the string "]]>", therefore, nested CDATA sections are not allowed.

Also make sure there are no spaces or line breaks inside the "]]>" string.

# XML Encoding

XML documents can contain foreign characters like Norwegian æøå, or French êëé.

To let your XML parser understand these characters, you should save your XML documents as Unicode.

## 8.64 Windows 95/98 Notepad

**Windows 95/98 Notepad cannot save files in Unicode format.**

You can use Notepad to edit and save XML documents that contain foreign characters (like Norwegian or French æøå and êëé),

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

But if you save the file and [open it with IE 5.0](#), you will get an **ERROR MESSAGE**.

## 8.65 Windows 95/98 Notepad with Encoding

**Windows 95/98 Notepad files must be saved with an encoding attribute.**

To avoid this error you can add an encoding attribute to your XML declaration, but you cannot use Unicode.

The encoding below ([open it with IE 5.0](#)), will NOT give an error message:

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

The encoding below ([open it with IE 5.0](#)), will NOT give an error message:

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

The encoding below ([open it with IE 5.0](#)), will NOT give an error message:

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

The encoding below ([open it with IE 5.0](#)), WILL give an error message:

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

## 8.66 Windows 2000 Notepad

### Windows 2000 Notepad can save files as Unicode.

The Notepad editor in Windows 2000 supports Unicode. If you select to save this XML file 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 following file; [note\\_encode\\_none\\_u.xml](#), will NOT give an error message if you open it with IE 5.0, but if you open it with Netscape 6.2, you WILL get an error message.

## 8.67 Windows 2000 Notepad with Encoding

### Windows 2000 Notepad files saved as Unicode use "UTF-16" encoding.

If you add an encoding attribute to XML files saved as Unicode, windows encoding values will generate an error.

This encoding ([open it](#)), will NOT give an error message:

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

This encoding ([open it](#)), will NOT give an error message:

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

This encoding ([open it](#)), will NOT give an error message:

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

The following file; [note\\_encode\\_utf16\\_u.xml](#), will NOT give an error message if you open it with IE 5.0, but if you open it with Netscape 6.2, you WILL get an error message.

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

## 8.68 Error Messages

If you try to load an XML document into Internet Explorer 5, you can get two different errors indicating encoding problems:

### An invalid character was found in text content.

You will get this error message if a character in the XML document does not match the encoding attribute. Normally you will get this error message if your XML document contains

"foreign" characters, and the file was saved with a single-byte encoding editor like Notepad, and no encoding attribute was specified.

### **Switch from current encoding to specified encoding not supported.**

You will get this error message if your file was saved as Unicode/UTF-16 but the encoding attribute specified a single-byte encoding like Windows-1252, ISO-8859-1 or UTF-8. You can also get this error message if your document was saved with single-byte encoding, but the encoding attribute specified a double-byte encoding like UTF-16.

## **8.69 Conclusion**

The conclusion is that the encoding attribute has to specify the encoding used when the document was saved. My best advice to avoid errors is:

- **Use an editor that supports encoding.**
- **Make sure you know what encoding it uses.**
- **Use the same encoding attribute in your XML documents.**

# A Simple XML Server

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

## 8.70 Storing XML on the Server

**XML files can be stored on your Internet server.**

XML files can be stored on your Internet server, in exactly the same way as HTML files.

Start up your Notepad editor 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>
```

All you have to do is to save the file on your Internet server with a proper name like "note.xml", before the XML document is ready to use.

**Note:** The XML file must be in the same directory (folder) as your HTML files, and the MIME type of XML files should be set to text/xml.

## 8.71 Generating XML with ASP

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

To generate an XML response from your server - simply write the following code and save it as an ASP file on your 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 XML. [See how the ASP file will be returned from the server.](#)

(ASP stands for Active Server Pages. If you don't know how to write ASP, you can study it in our [ASP Tutorial](#))

## 8.72 Getting XML from a Database

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

The XML response from the previous example can easily be modified to fetch its data from a database.

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

```
<%
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)
rs.MoveFirst()
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 this page.](#)

The example above uses ASP with ADO. If you don't know how to use ADO, you can study it in our [ADO tutorial](#).

# XML Applications

This chapter demonstrates a small framework for an XML application.

## 8.73 Start with an XML document

**First we start with a simple XML document.**

Take a look at our original demonstration document, the 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>
  .
  .
  ... more ...
  .
```

If you have Internet Explorer 5.0 or higher, [see the full XML file](#).

## 8.74 Load the document into a Data Island

**A Data Island can be used to access the XML file.**

To get your XML document "inside" an HTML page, add an XML Data Island to the page.

```
<xml src="cd_catalog.xml" id="xmldso" async="false">
</xml>
```

With the example code above, the XML file "cd\_catalog.xml" will be loaded into an "invisible" Data Island called "xmldso". The `async="false"` attribute is added to the Data Island to make sure that all the XML data is loaded before any other HTML processing takes place.

## 8.75 Bind the Data Island to an HTML Table

**An HTML table can be used to display the XML data.**

To make your XML data visible on your HTML page, you must "bind" your XML Data Island to an HTML element.

To bind your XML data to an HTML table, add a data source attribute to the table, and add data field attributes to `<span>` elements inside the table data:

```

<table datasrc="#xmldso" width="100%" border="1">

<thead>
<th>Title</th>
<th>Artist</th>
<th>Year</th>
</thead>

<tr align="left">
<td><span datafld="TITLE"></span></td>
<td><span datafld="ARTIST"></span></td>
<td><span datafld="YEAR"></span></td>
</tr>
</table>

```

If you have Internet Explorer 5.0 or higher: [See how the XML data is displayed inside an HTML table.](#)

## 8.76 Bind the Data Island to <span> or <div> elements

**<span> or <div> elements can be used to display XML data.**

You don't have to use a table to display your XML data. Data from a Data Island can be displayed anywhere on an HTML page.

All you have to do is to add some <span> or <div> elements to your page. Use the data source attribute to bind the elements to the Data Island, and the data field attribute to bind each element to an XML element, like this:

```

<br />Title:
<span datasrc="#xmldso" datafld="TITLE"></span>
<br />Artist:
<span datasrc="#xmldso" datafld="ARTIST"></span>
<br />Year:
<span datasrc="#xmldso" datafld="YEAR"></span>

```

or like this:

```

<br />Title:
<div datasrc="#xmldso" datafld="TITLE"></div>
<br />Artist:
<div datasrc="#xmldso" datafld="ARTIST"></div>
<br />Year:
<div datasrc="#xmldso" datafld="YEAR"></div>

```

If you have Internet Explorer 5.0 or higher: [See how the XML data is displayed inside the HTML elements.](#)

Note that if you use a <div> element, the data will be displayed on a new line.

With the examples above, you will only see one line of your XML data. To navigate to the next line of data, you have to add some scripting to your code.

## 8.77 Add a Navigation Script to your XML

**Navigation has to be performed by a script.**

To add navigation to the XML Data Island, create a script that calls the `movenext()` and `moveprevious()` methods of the Data Island.

```
<script type="text/javascript">
function movenext()
{
  x=xmldso.recordset
  if (x.absoluteposition < x.recordcount)
  {
    x.movenext()
  }
}
function moveprevious()
{
  x=xmldso.recordset
  if (x.absoluteposition > 1)
  {
    x.moveprevious()
  }
}
</script>
```

If you have Internet Explorer 5.0 or higher: [See how you can navigate through the XML records.](#)

## 8.78 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.

If you are running Internet Explorer 5.0 or higher: [See how you can add a little fancy to this application.](#)

# XML HTTP Requests

If you are using IE 5.0 or higher, XML data can be requested from a server using an HTTP request.

## 8.79 The Browser Request

**An HTTP request from the browser, can request XML from a server:**

```
var objHTTP = new ActiveXObject("Microsoft.XMLHTTP")
objHTTP.Open('GET','httprequest.asp',false)
objHTTP.Send()
```

To view the result from the request, you can display the result in your browser:

```
document.all['A1'].innerText= objHTTP.status
document.all['A2'].innerText= objHTTP.statusText
document.all['A3'].innerText= objHTTP.responseText
```

[Try it Yourself using JavaScript](#)

[Try it Yourself using VBScript](#)

## 8.80 Communicating with the Server

**With HTTP requests you can "communicate" with a server.**

[Communicating with a server using XML](#)

In the example the response is "faked" on the server with this ASP code:

```
<%
response.ContentType="text/xml "
txt="<answer><text>12 Years</text></answer>"
response.write(txt)
%>
```

So, the answer will always be 12 years, no matter what question is asked. In real life, you have to write some code to analyze the question and respond with a correct answer.

# XML Behaviors - the new DHTML?

A behavior is a CSS attribute selector. It can point to an XML file that contains code to be executed against elements in a Web page.

Behaviors is not a W3C standard, but a Microsoft-only technology.

## 8.81 Behaviors - What are they?

**A behavior is a new CSS attribute selector.**

A behavior selector can point to a separate XML file that contains code to be executed against XML or HTML elements in a Web page.

Did you understand that? A method for completely removing script code from HTML pages? That's great! Now we can start writing script libraries, and attach our scripts to any element we want!

## 8.82 How does it work?

Take a look at this HTML file. It has a <style> element that defines a behavior for the <h1> element:

```
<html>
<head>
<style>
h1 { behavior: url(behave.htc) }
</style>
</head>

<body>
<h1>Move your Mouse over me</h1>
</body>
</html>
```

Try it yourself with [this example](#), and move the mouse over the text.

The behavior code is stored in an XML document behave.htc as shown below:

```
<component>
<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=255
  }
  function low_lite()
  {
    element.style.color=0
  }
</script>
</component>
```

The behavior file contains JavaScript. The script is wrapped in a <component> element. The component wrapper also contains the event handlers for the script. Nice behavior, isn't it?

# XML Related Technologies

This chapter contains a list of technologies that are important to the understanding and development of XML applications.

It can also be viewed as "where to go from here" information, if you want to study more XML.

## XHTML - Extensible HTML

XHTML is the re-formulation of HTML 4.01 in XML. XHTML 1.0 is the latest version of HTML. [Read more in our XHTML tutorial.](#)

## CSS - Cascading Style Sheets

CSS style sheets can be added to XML documents to provide display information. [Read more in our CSS tutorial.](#)

## XSL - Extensible Style Sheet Language

XSL consists of three parts: XML Document Transformation (renamed XSLT, see below), a pattern matching syntax (renamed XPath, see below), and a formatting object interpretation.

## XSLT - XML Transformation

XSLT is far more powerful than CSS. It can be used to transform XML files into many different output formats. [Read more in our XSLT tutorial.](#)

## XPath - XML Pattern Matching

XPath is a language for addressing parts of an XML document. XPath was designed to be used by both XSLT and XPointer.

## XLink - XML Linking Language

The XML Linking Language (XLink) allows elements to be inserted into XML documents in order to create links between XML resources.

## XPointer - XML Pointer Language

The XML Pointer Language (XPointer) supports addressing into the internal structures of XML documents, such as elements, attributes, and content.

## DTD - Document Type Definition

A DTD can be used to define the legal building blocks of an XML document. [Read more in our DTD tutorial.](#)

## **Namespaces**

XML namespaces defines a method for defining element and attribute names used in XML by associating them with URI references.

## **XSD - XML Schema**

Schemas are powerful alternatives to DTDs. Schemas are written in XML, and support namespaces and data types. [Read more in our Schema tutorial.](#)

## **XDR - XML Data Reduced**

XDR is a reduced version of XML Schema. Support for XDR was shipped with Internet Explorer 5.0 when XML Schema was still a working draft. Microsoft has committed full support for XML Schema as soon as the specification becomes a W3C Recommendation.

## **DOM - Document Object Model**

The DOM defines interfaces, properties and methods to manipulate XML documents. [Read more in our DOM tutorial.](#)

## **XQL - XML Query Language**

The XML Query Language supports query facilities to extract data from XML documents.

## **SAX - Simple API for XML**

SAX is another interface to read and manipulate XML documents.

## **8.83 W3C Recommendations**

The World Wide Web Consortium (W3C) was founded in 1994 to lead the Web by developing common WWW protocols like HTML, CSS and XML.

The most important work done by the W3C is the development of Web specifications (called "Recommendations") that describe communication protocols (like HTML and XML) and other building blocks of the Web.

Read more about the status of each XML standard at our [W3C School](#).

# XML Editors

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

## 8.84 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.

## 8.85 Why Not Notepad?

Many "hardcore" web developers use Notepad to edit both HTML and XML documents because Notepad is free and simple to use, and 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. You will create many errors, and as your XML documents grow larger you will lose control.

## 8.86 Why an XML Editor?

Today XML is an important technology, and everyday we can see XML playing a more and more critical role in new web development.

New 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... and much more

To be able to write XML documents for all your new development projects, you will need an intelligent editor to help you write error free XML documents.

## 8.87 XML Editors

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

A good 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

## **8.88 XMLSPY**

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

- Easy to use
- Syntax coloring
- Automatic tag completion
- Automatic well-formed check
- Easy switching between text view and grid view
- Built in DTD and / or Schema validation
- Built in graphical XML Schema designer
- Powerful conversion utilities
- Database import and export
- Built in templates for most XML document types
- Built in XPath analyzer
- Full SOAP and WSDL capabilities
- Powerful project management

[Read more about XMLSPY](#)

# XML Examples

**Note:** To view an XML document in Netscape 6: Open the XML file and then right-click in the XML file and select "View Page Source".

**Note:** Not all of the examples below work in Netscape.

## 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](#)

## Viewing XML files with a dtd

[View note.xml with an internal dtd](#)  
[View note.xml with an external dtd](#)

## The Microsoft XML parser

[View a simple XML file \(xml\\_note.xml\)](#)  
[Loading the same file into the parser](#)  
[Traversing the node tree of the file](#)  
[Loading the same file into HTML](#)

## Displaying using JavaScript

[View a simple XML file \(xml\\_note.xml\)](#)  
[Format the same file with JavaScript](#)

## XML and CSS

[View an XML CD catalog](#)  
[View the corresponding CSS file](#)  
[Display the CD catalog formatted with the CSS file](#)

## XML and XSL

[View an XML food menu](#)  
[View the corresponding XSL stylesheet](#)  
[Display the food menu styled with the XSL stylesheet \(IE6\)](#)  
[Display the food menu styled with the XSL stylesheet \(IE5\)](#)

## Data Binding

[View an XML CD catalog](#)  
[Bind the CD catalog to an HTML table](#)  
[Add <thead>, <tfoot>, <tbody> elements](#)

## **Database Output**

[View XML output from a database](#)

## **Displayed as HTML**

[View an XML CD catalog](#)

[See how the CD catalog can be displayed inside HTML elements](#)

[See how the CD catalog can be displayed inside an HTML table](#)

[See how to navigate the CD catalog](#)

[A simple CD catalog application](#)

## **Requesting XML data from a server**

[Request XML from a server using JavaScript](#)

[Request XML from a server using VBScript](#)

[Send a request to the server](#)

[Communicating with a server using XML](#)

## **XML Behaviors**

[XML Behaviors](#)