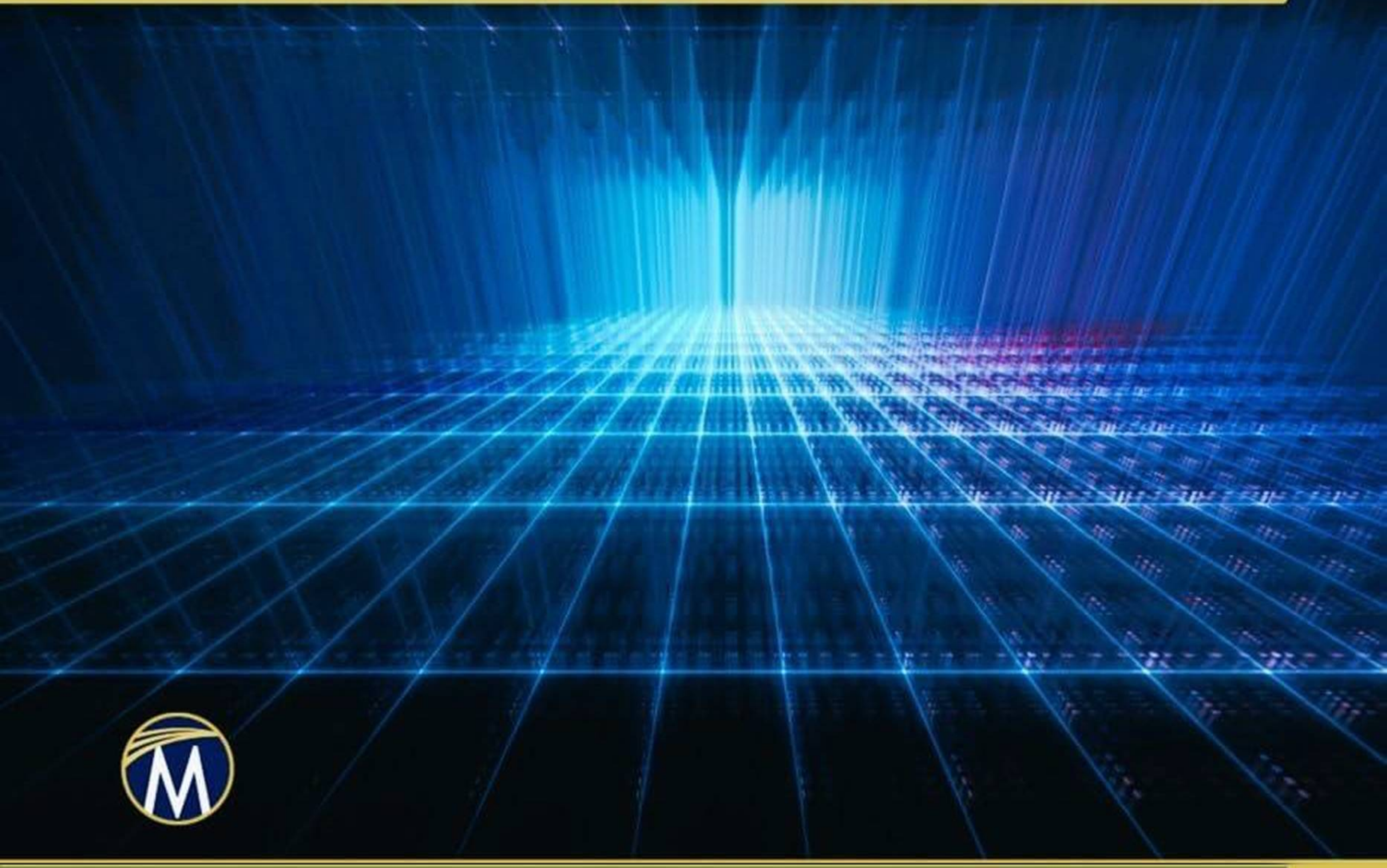
XML BASICS



XML stands for Extensible Markup Language.

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.

What is XML?

● XML stands for E**X**tensible **M**arkup **L**anguage

● 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**

● XML is a W3C (World Wide Web Consortium)Recommendation

XML is a W3C Recommendation

The Extensible Markup Language (XML) became a W3C Recommendation 10. February 1998. You can read more about XML standards in our W3C tutorial.

W3C was started in 1994 to lead the Web to its full potential by developing common protocols that promote its evolution and ensure its interoperability.

What it Is

- W3C Stands for the **World Wide Web Consortium**
- W3C was created in **October 1994**
- W3C was created by **Tim Berners-Lee**
- W3C was created by the **Inventor of the Web**
- W3C is organized as a **Member Organization**
- W3C is working to **Standardize the Web**
- W3C creates and maintains **WWW Standards**
- W3C Standards are called **W3C Recommendations**

How it Started

The World Wide Web (WWW) began as a project at the European Organization for Nuclear Research (CERN), where Tim Berners-Lee developed a vision of the World Wide Web.
Tim Berners-Lee - the inventor of the World Wide Web - is now the Director of the World Wide Web Consortium (W3C).
W3C was created in 1994 as a collaboration between the Massachusetts Institute of Technology (MIT) and

the European Organization for Nuclear Research (CERN), with support from the U.S. Defense Advanced Research Project Agency (DARPA) and the European Commission.

Standardizing the Web

W3C is working to make the Web accessible to all users (despite differences in culture, education, ability, resources, and physical limitations)
W3C also coordinates its work with many other standards organizations such as the Internet Engineering Task Force, the Wireless Application Protocols (WAP) Forum and the Unicode Consortium.
W3C is hosted by three universities:

- Massachusetts Institute of Technology in the U.S.
- The French National Research Institute in Europe
- Keio University in Japan

W3C Members

Because the Web is so important (both in scope and in investment) that no single organization should have control over its future, W3C functions as a member organization. Some well known members are:

- IBM
- Microsoft
- America Online
- Apple
- Adobe
- Macromedia
- Sun Microsystems

The Full List of Member Organisations includes a variety of software vendors, content providers, corporate users, telecommunications companies, academic institutions, research laboratories, standards bodies, and governments.

W3C Recommendations

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.
Each W3C Recommendation is developed by a work group consisting of members and invited experts. The

group obtains its input from companies and other organizations, and creates a Working Draft and finally a Proposed Recommendation. In general the Recommendation is submitted to the W3C membership and director, for a formal approval as a W3C Recommendation. The specification approval process is described in the next chapter.

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.

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.

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.

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.**

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.

How can XML be Used?

Previous Next

It is important to understand that XML was designed to store, carry, and ex-

change data. XML was not designed to display data.

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.

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.

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.

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.

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.

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.

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.

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 Rules

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.

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?

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.

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>
```

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>

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>
```

XML Attribute Values Must 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.

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.

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.

Comments in XML The syntax for writing comments in XML is similar to that of HTML. <!-- This is a comment -->

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 Attributes

Previous Next

XML elements can have attributes in the start tag, just like HTML.
Attributes are used to provide additional information about elements.

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: <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>

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">
```

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.

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>
```

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 (this is not how XML should be used):

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

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.

DTD Tutorial

Previous Next DTD Tutorial

The purpose of a DTD (Document Type Definition) is to define the legal building blocks of an XML document. It defines the document structure with a list of legal elements.

Introduction to DTD

Previous Next **A Document Type Definition defines the legal building blocks of an XML document. It defines the document structure with a list of legal elements.**
A DTD can be declared inline in your XML document, or as an external reference.

Internal DOCTYPE declaration

If the DTD is included in your XML source file, it should be wrapped in a DOCTYPE definition with the following syntax:

```
<!DOCTYPE root-element [element-declarations]>
```

Example XML document with a DTD: (Open it in IE5, and select view source):

```
<?xml version="1.0"?>

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

]>
<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
```

```
<body>Don't forget me this weekend</body>
</note>
```

The DTD above is interpreted like this:

!DOCTYPE note (in line 2) defines that this is a document of the type **note. !ELEMENT note** (in line 3) defines the **note** element as having four elements: "to,from,heading,body".
!ELEMENT to (in line 4) defines the **to** element to be of the type "#PCDATA". **!ELEMENT from** (in line 5) defines the **from** element to be of the type "#PCDATA" and so on.....

External DOCTYPE declaration

If the DTD is external to your XML source file, it should be wrapped in a DOCTYPE definition with the following syntax:

```
<!DOCTYPE root-element SYSTEM "filename">
```

This is the same XML document as above, but with an external DTD: (Open it in IE5, and select view source)

```
<?xml version="1.0"?>
<!DOCTYPE note SYSTEM "note.dtd">
<note>
```

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

And this is a copy of the file "note.dtd" containing the DTD:

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

Why use a DTD?

With DTD, each of your XML files can carry a description of its own format with it. With a DTD, independent groups of people can agree to use a common DTD for interchanging data. Your application can use a

standard DTD to verify that the data you receive from the outside world is valid.
You can also use a DTD to verify your own data.

DTD - XML building blocks

Previous Next

The main building blocks of both XML and HTML documents are tags like <body>....</body>.

The building blocks of XML documents

Seen from a DTD point of view, all XML documents (and HTML documents) are made up by the following simple building blocks:

- Elements
- Attributes
- Entities
- PCDATA
- CDATA

The following is a brief explanation of each of the building blocks:

Elements

Elements are the **main building blocks** of both XML and HTML documents.
Examples of HTML elements are "body" and "table". Examples of XML elements could be "note" and "message". Elements can contain text, other elements, or be empty. Examples of empty HTML elements are "hr", "br" and "img".
Examples:

<body>body text in between</body>

<message>some message in between</message>

Attributes

Attributes provide **extra information about elements**.
Attributes are always placed inside the starting tag of an element. Attributes always come in name/value pairs. The following "img" element has additional information about a source file: `<img src="computer.gif" />`
The name of the element is "img". The name of the attribute is "src". The value of the attribute is "computer.gif". Since the element itself is empty it is closed by a " /".

Entities

Entities are variables used to **define common text**. Entity references are references to entities. Most of you will know the HTML entity reference: "&nbsp;". This "no-breaking-space" entity is used in HTML to insert an extra space in a document. Entities are expanded when a document is parsed by an XML parser.
The following entities are predefined in XML:
Entity References Character

&lt; <

&gt; >

&amp; &

&quot; "

&apos; '

PCDATA

PCDATA means parsed character data.
Think of character data as the text found between the start tag and the end tag of an XML element. **PCDATA is text that will be parsed by a parser**. Tags inside the text will be treated as markup and entities will be expanded.

CDATA

CDATA also means character data.
CDATA is text that will NOT be parsed by a parser. Tags inside the text will NOT be treated as markup and entities will not be expanded.

DTD - Elements

Previous Next

In a DTD, XML elements are declared with a DTD element declaration.

Declaring an Element

In the DTD, XML elements are declared with an element declaration. An element declaration has the following syntax:

<!ELEMENT element-name category>

or

<!ELEMENT element-name (element-content)>

Empty elements

Empty elements are declared with the category keyword EMPTY: <!ELEMENT element-name EMPTY>

example:

<!ELEMENT br EMPTY>

XML example:

<br />

Elements with only character data Elements with only character data are declared with #PCDATA inside parentheses: <!ELEMENT element-name (#PCDATA)>

example:

<!ELEMENT from (#PCDATA)>

Elements with any contents

Elements declared with the category keyword ANY, can contain any combination of parsable data: <!
ELEMENT element-name ANY>

example:

<!ELEMENT note ANY>

Elements with children (sequences)

Elements with one or more children are defined with the name of the children elements inside parentheses:

<!ELEMENT element-name

(child-element-name)>

or

<!ELEMENT element-name

(child-element-name,child-element-name,.....)>

example:

<!ELEMENT note (to,from,heading,body)>

When children are declared in a sequence separated by commas, the children must appear in the same sequence in the document. In a full declaration, the children must also be declared, and the children can also have children. The full declaration of the "note" element will be: <!ELEMENT note (to,from, heading,body)>

<!ELEMENT to (#PCDATA)>

<!ELEMENT from (#PCDATA)>

<!ELEMENT heading (#PCDATA)>

<!ELEMENT body (#PCDATA)>

Declaring only one occurrence of the same element

<!ELEMENT element-name (child-name)>

example:

<!ELEMENT note (message)>

The example declaration above declares that the child element message must occur once, and only once inside the "note" element.

Declaring minimum one occurrence of the same element

<!ELEMENT element-name (child-name+)>

example:

<!ELEMENT note (message+)>

The + sign in the example above declares that the child element message must occur one or more times inside the "note" element.

Declaring zero or more occurrences of the same element

<!ELEMENT element-name (child-name*)>

example:

<!ELEMENT note (message*)>

The * sign in the example above declares that the child element message can occur zero or more times inside the "note" element.

Declaring zero or one occurrences of the same element

<!ELEMENT element-name (child-name?)>

example:

<!ELEMENT note (message?)>

The ? sign in the example above declares that the child element message can occur zero or one times inside the "note" element.

Declaring either/or content

example:

<!ELEMENT note (to,from,header,(message|body))>

The example above declares that the "note" element must contain a "to" element, a "from" element, a "header" element, and either a "message" or a "body" element.

Declaring mixed content

example:

<!ELEMENT note (#PCDATA|to|from|header|message)*>

The example above declares that the "note" element can contain zero or more occurrences of parsed character, "to", "from", "header", or "message" elements.

DTD - Attributes

In a DTD, Attributes are declared with an ATTLIST declaration.

Declaring Attributes

An attribute declaration has the following syntax:

```
<!ATTLIST element-name attribute-name attribute-type default-value>
```

example:

DTD example:

```
<!ATTLIST payment type CDATA "check">
```

XML example:

```
<payment type="check" />
```

The **attribute-type** can have the following values: **Value**

CDATA

(en1|en2|..) ID

IDREF

IDREFS

NMTOKEN

NMTOKENS ENTITY

ENTITIES

NOTATION xml:

Explanation

The value is character data

The value must be one from an enumerated list The value is a unique id

The value is the id of another element The value is a list of other ids

The value is a valid XML name

The value is a list of valid XML names

The value is an entity

The value is a list of entities

The value is a name of a notation

The value is a predefined xml value

The **default-value** can have the following values: **Value**
value
#REQUIRED #IMPLIED
#FIXED value

Explanation
The default value of the attribute
The attribute value must be included in the element The attribute does not have to be included The attribute value is fixed

Specifying a Default attribute value

DTD:

```
<!ELEMENT square EMPTY>
<!ATTLIST square width CDATA "0">
```

Valid XML:

```
<square width="100" />
```

In the example above, the "square" element is defined to be an empty element with a "width" attribute of type CDATA. If no width is specified, it has a default value of 0.

#IMPLIED

Syntax

```
<!ATTLIST element-name attribute-name attribute-type #IMPLIED>
```

Example

DTD:

```
<!ATTLIST contact fax CDATA #IMPLIED>
```

Valid XML:

```
<contact fax="555-667788" />
```

Valid XML:

```
<contact />
```

Use the #IMPLIED keyword if you don't want to force the author to include an attribute, and you don't have an option for a default value.

#REQUIRED

Syntax

<!ATTLIST element-name attribute_name attribute-type #REQUIRED>

Example

DTD:

<!ATTLIST person number CDATA #REQUIRED>

Valid XML:

<person number="5677" />

Invalid XML:

<person />

Use the #REQUIRED keyword if you don't have an option for a default value, but still want to force the attribute to be present.

#FIXED

Syntax

<!ATTLIST element-name attribute-name attribute-type #FIXED

"value">

Example

DTD:

<!ATTLIST sender company CDATA #FIXED "Microsoft">

Valid XML:

<sender company="Microsoft" />

Invalid XML:

<sender company="W3Schools" />

Use the #FIXED keyword when you want an attribute to have a fixed value without allowing the author to change it. If an author includes another value, the XML parser will return an error.

Enumerated attribute values

Syntax:

```
<!ATTLIST element-name
attribute-name (en1|en2|..) default-value>
```

DTD example:

```
<!ATTLIST payment type (check|cash) "cash">
```

XML example:

```
<payment type="check" />
```

or

```
<payment type="cash" />
```

Use enumerated attribute values when you want the attribute values to be one of a fixed set of legal values.

DTD - Entities

◂ Previous Next ▸**Entities are variables used to define shortcuts to common text.**

- Entity references are references to entities.

- Entities can be declared internal, or external

Internal Entity Declaration

Syntax:

<!ENTITY entity-name "entity-value">

DTD Example:

<!ENTITY writer "Donald Duck.">

<!ENTITY copyright "Copyright W3Schools."> XML example:

<author>&writer;&copyright;</author>

External Entity Declaration Syntax:

<!ENTITY entity-name SYSTEM "URI/URL">

DTD Example:

<!ENTITY writer

SYSTEM "http://www.w3schools.com/dtd/entities.dtd"> <!ENTITY copyright

SYSTEM "http://www.w3schools.com/dtd/entities.dtd"> XML example:

<author>&writer;&copyright;</author>

DTD Validation

◂ Previous Next ▸ **Internet Explorer 5.0 can validate your XML against a DTD.**

Validating with the XML Parser

If you try to open an XML document, the XML Parser might generate an error. By accessing the parseError object, the exact error code, the error text, and even the line that caused the error can be retrieved:

Note: The load( ) method is used for files, while the loadXML( ) method is used for strings. var xmlDoc = new ActiveXObject("Microsoft.XMLDOM") xmlDoc.async="false"

xmlDoc.validateOnParse="true"

xmlDoc.load("note_dtd_error.xml")

document.write("<br>Error Code: ")

document.write(xmlDoc.parseError.errorCode) documen-

t.write("<br>Error Reason: ") document.write(xmlDoc.parseError.rea-

son) document.write("<br>Error Line: ")

document.write(xmlDoc.parseError.line) Try it Yourself or just look at the XML file

Turning Validation off

Validation can be turned off by setting the XML parser's validateOnParse="false". var xmlDoc =

new ActiveXObject("Microsoft.XMLDOM") xmlDoc.async="false"

xmlDoc.validateOnParse="false"

xmlDoc.load("note_dtd_error.xml")

document.write("<br>Error Code: ")

document.write(xmlDoc.parseError.errorCode) document.write("<br>Error Reason: ") document.write(xmlDoc.parseError.reason) document.write("<br>Error Line: ")

document.write(xmlDoc.parseError.line) Try it Yourself

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 Namespaces

◂ Previous Next ▸ **XML Namespaces provide a method to avoid element name conflicts.**

Name Conflicts

Since element names in XML are not predefined, a name conflict will occur when two different documents use the same element names.

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.

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 there will be no name conflict 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.

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, we have added an xmlns attribute to the <table> tag to give the prefix a qualified name associated with a namespace.

The XML Namespace (xmlns) Attribute

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

xmlns:*namespace-prefix*="*namespaceURI*"

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/.

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 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>
```

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/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">
<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 Data Island

Previous Next**With Internet Explorer, the unofficial <xml> tag can be used to create an XML data island.**

XML Data Embedded in HTML

An XML data island is XML data embedded into an HTML page.

Here is how it works; assume we have the following XML document ("note.xml"):

```
<?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>
```

Then, in an HTML document, you can embed the XML file above with the <xml> tag. The id attribute of the <xml> tag defines an ID for the data island, and the src attribute points to the XML file to embed:

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

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

However, the embedded XML data is, up to this point, not visible for the user.
The next step is to format and display the data in the data island by binding it to HTML elements.

Bind Data Island to HTML Elements

In the next example, we will embed an XML file called "note.xml" into an HTML file. The HTML file looks like this:

```
<html>
<body>

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

<table border="1" datasrc="#note">
<td><span datafld="to"></span></td> <td><span datafld="from"></
```

```
span></td> <td><span datafld="heading"></span></td> <td><span
datafld="body"></span></td> </tr>
</table>


</body>
</html>
```

Example explained:

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

<td> tags cannot be bound to data, so we are using <span> tags. The <span> tag allows the datafld attribute to refer to the XML element to be displayed. In this case, it is datafld="to" for the <to> element and datafld="from" for the <from> element in the XML file. As the XML is read, additional rows are created for each <CD> element.

If you are running IE 5.0 or higher, you can try it yourself.

Also try this example, demonstrating <thead>, <tbody>, and <tfoot>.

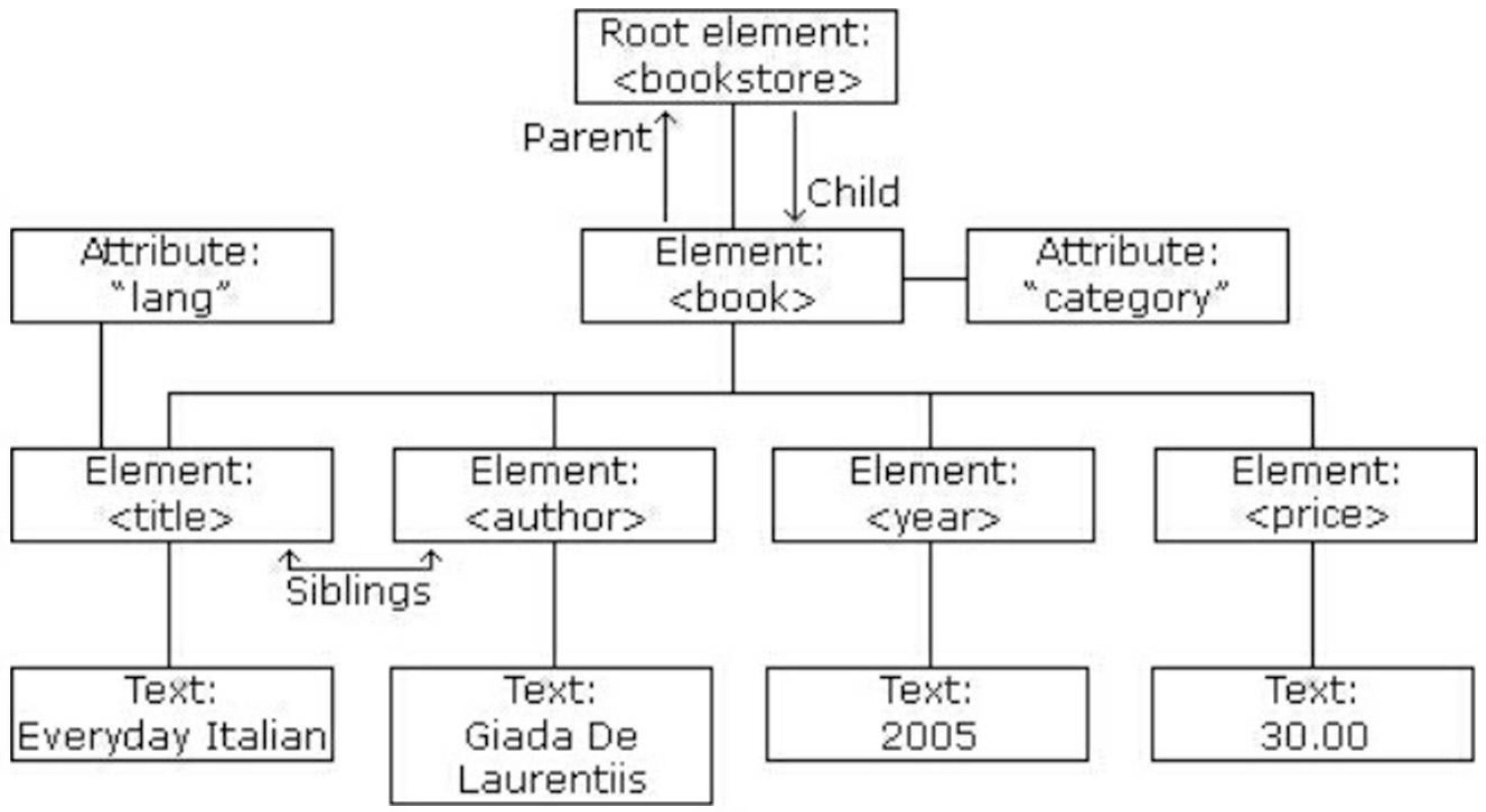
XML DOM Tutorial

◂ Previous | Next ▸

XML DOM Tutorial

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

The DOM presents an XML document as a tree-structure (a node tree), with the elements, attributes, and text defined as nodes.



XML DOM Introduction

Previous Next **The XML Document Object Model defines a standard way for accessing and manipulating XML documents.**

What You Should Already Know

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

- HTML / XHTML
- JavaScript
- XML

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

What is the DOM?

"The W3C Document Object Model (DOM) is a platform and language-neutral interface that allows programs and scripts to dynamically access and update the content, structure, and style of a document."
The W3C DOM provides a standard set of objects for HTML and XML documents, and a standard interface for accessing and manipulating them.

The W3C DOM is separated into different parts (Core, XML, and HTML) and different levels (DOM Level 1/2/3):

● Core DOM - defines a standard set of objects for any structured document ● XML DOM - defines a standard set of objects for XML documents ● HTML DOM - defines a standard set of objects for HTML documents

You can read more about the W3C DOM specifications / levels in our W3C tutorial.

What is the XML DOM?

● The XML DOM is the Document Object Model for XML

● The XML DOM is platform- and language-independent

● The XML DOM defines a standard set of objects for XML ● The XML DOM defines a standard way to access XML documents ● The XML DOM defines a standard way to manipulate XML documents ● The XML DOM is a W3C standard

The DOM views XML documents as a tree-structure. All elements; their containing text and their attributes, can be accessed through the DOM tree. Their contents can be modified or deleted, and new elements can be created. The elements, their text, and their attributes are all known as nodes.

XML DOM - The Attr Object

◂ Previous Next ▸**The Attr object represents an attribute of an Element object.**

The Attr object

The Attr object represents an attribute of an Element object. The allowable values for attributes are usually defined in a DTD.
Because the Attr object is also a Node, it inherits the Node object's properties and methods. However, an attribute does not have a parent node and is not considered to be a child node of an element, and will return null for many of the Node properties.
IE: Internet Explorer, **F:** Firefox, **O:** Opera, **W3C:** World Wide Web Consortium (Internet Standard)

Attr Object Properties

Property baseURI isId

localName

name

namespaceURI nodeName

nodeType nodeValue

ownerDocument

ownerElement

prefix

schemaTypeInfo

specified

textContent

text

value xml

Description IE F O W3C Returns the absolute base URI of the attribute No 1 No Yes Returns true if the attribute is known to be of type ID, otherwise it returns false

Returns the local part of the name of the attribute

Returns the name of the attribute

Returns the namespace URI of the attribute Returns the name of the node, depending on its type

Returns the type of the node

Sets or returns the value of the node,

depending on its type

Returns the root element (document object) for an attribute

Returns the element node the attribute is attached to

Sets or returns the namespace prefix of the attribute

Returns the type information associated with this attribute

Returns true if the attribute value is set in the document, and false if it's a default value in a DTD/Schema.

Sets or returns the textual content of an attribute

Returns the text of the attribute. IE-only property

Sets or returns the value of the attribute Returns the XML of the attribute. IE-only property

No No No Yes

No 1 9 Yes

5 1 9 Yes No 1 9 Yes

5 1 9 Yes

5 1 9 Yes 5 1 9 Yes

5 1 9 Yes

No 1 9 Yes

No 1 9 Yes

No No No Yes

5 1 9 Yes

No 1 9 Yes

5 No No No

5 1 9 Yes 5 No No No

XML DOM Examples

Syntax attrObject.baseURI

In all examples, we will use the XML file books_ns.xml, and the JavaScript function loadXMLDoc(). **Example 1**

The following code fragment returns the base URI of the "lang" attributes: xmlDoc=loadXMLDoc("books_ns.xml");

var x=xmlDoc.getElementsByTagName('title');

for(i=0;i<x.length;i++)

```
{
document.write(x.item(i).attributes[0].baseURI);
document.write("<br />");
}
```

The output of the code above will be:

```
http://www.w3schools.com/dom/books_ns.xml http://www.w3schools.com/dom/books_ns.xml
```

Syntax

```
attrObject.value
```

In all examples, we will use the XML file books.xml, and the JavaScript function loadXMLDoc(). **Example 2**

The following code fragment displays the name and value of the category attribute:

```
xmlDoc=loadXMLDoc("books.xml");
var x=xmlDoc.getElementsByTagName('book');
for(i=0;i<x.length;i++)

{
document.write(x.item(i).attributes[0].name);
document.write(" = ");
document.write(x.item(i).attributes[0].value);
document.write("<br />");
}
```

Output:

```
category = COOKING
category = CHILDREN
```

category = WEB

category = WEB

XML DOM Examples

XML DOM Parsing

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

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

- <!-

Edited with XML Spy v2006 (http://www.altova.com)

-->

```
-<note time="12:03:46"> <to>Tove</to> <from>Jani</from>
<heading>Reminder</heading> <body>Don't
forget me this weekend!</body> </note>
```

XHTML Tutorial

◂ Previous | Next ▸ XHTML Tutorial

XHTML is a stricter and cleaner version of HTML.

In this tutorial you will learn the difference between HTML and XHTML. We will also show you how this Web site was converted to XHTML.

Introduction To XHTML

◂ Previous Next ▸**XHTML is a stricter and cleaner version of HTML.**

What You Should Already Know

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

- HTML and the basics of building web pages

If you want to study HTML first, please read our HTML tutorial.

What Is XHTML?

- XHTML stands for **EX**tensible **H**yper**T**ext **M**arkup **L**anguage
- XHTML is aimed to **replace** HTML
- XHTML is almost **identical** to HTML 4.01
- XHTML is a **stricter and cleaner** version of HTML
- XHTML is HTML defined as an **XML application**
- XHTML is a W3C Recommendation

XHTML is a W3C Recommendation

XHTML 1.0 became a W3C Recommendation January 26, 2000.

W3C defines XHTML as the latest version of HTML. XHTML will gradually replace HTML. Stay updated with the latest W3C recommendations in our W3C tutorial.

All New Browsers Support XHTML XHTML is compatible with HTML 4.01. All new browsers have support for XHTML.

XHTML - Why?

Previous Next **XHTML is a combination of HTML and XML (EXtensible Markup Language). XHTML consists of all the elements in HTML 4.01 combined with the syntax of XML.**

Why XHTML?

We have reached a point where many pages on the WWW contain "bad" HTML.
The following HTML code will work fine if you view it in a browser, even if it does not follow the HTML rules:

```
<html>

<head>

<title>This is bad HTML</title>

<body>
```

<h1>Bad HTML

</body>

XML is a markup language where everything has to be marked up correctly, which results in "well-formed" documents.

XML was designed to describe data and HTML was designed to display data.

Today's market consists of different browser technologies, some browsers run Internet on computers, and some browsers run Internet on mobile phones and hand helds. The last-mentioned do not have the resources or power to interpret a "bad" markup language.

Therefore - by combining HTML and XML, and their strengths, we got a markup language that is useful now and in the future - XHTML.

XHTML pages can be read by all XML enabled devices AND while waiting for the rest of the world to upgrade to XML supported browsers, XHTML gives you the opportunity to write "well-formed" documents now, that work in all browsers and that are backward browser compatible !!!

Differences Between XHTML And HTML

Previous Next **You can prepare yourself for XHTML by starting to write strict HTML.**

How To Get Ready For XHTML

XHTML is not very different from the HTML 4.01 standard.
So, bringing your code up to the 4.01 standard is a good start. Our complete HTML 4.01 reference can help you with that.
In addition, you should start NOW to write your HTML code in lowercase letters, and NEVER skip ending tags (like </p>).
Happy coding!

The Most Important Differences:

● XHTML elements must be **properly nested** ● XHTML elements must always be **closed** ● XHTML elements must be in **lowercase** ● XHTML documents must have **one root element**

XHTML Elements Must Be Properly Nested

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

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

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

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

Note: A common mistake with nested lists, is to forget that the inside list must be within <li> and </li>

tags.

This is wrong:

```
<ul>

<li>Coffee</li>

<li>Tea


<ul>

<li>Black tea</li>

<li>Green tea</li>


</ul>

<li>Milk</li>

</ul>
```

This is correct:

```
<ul>
```

```
<li>Coffee</li>

<li>Tea


<ul>

<li>Black tea</li>

<li>Green tea</li>


</ul>

</li>

<li>Milk</li>


</ul>
```

Notice that we have inserted a </li> tag after the </ul> tag in the "correct" code example.

XHTML Elements Must Always Be Closed

Non-empty elements must have an end tag. This is wrong:

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

This is correct:

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

Empty Elements Must Also Be Closed

Empty elements must either have an end tag or the start tag must end with `/>`. This is wrong:

```
A break: <br>
A horizontal rule: <hr>
An image: <img src="happy.gif" alt="Happy face">
```

This is correct:

```
A break: <br />
```

A horizontal rule: <hr />

An image: <img src="happy.gif" alt="Happy face" />

XHTML Elements Must Be In Lower Case

The XHTML specification defines that the tag names and attributes need to be lower case. This is wrong:

```
<BODY>
<P>This is a paragraph</P>
</BODY>
```

This is correct:

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

XHTML Documents Must Have One Root Element

All XHTML elements must be nested within the <html> root element. All other elements can have sub (children) elements. Sub elements must be in pairs and correctly nested within their parent element. The basic document structure is:

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

XHTML Syntax

Previous Next **Writing XHTML demands a clean HTML syntax.**

Some More XHTML Syntax Rules:

● Attribute names must be in **lower case** ● Attribute values must be **quoted**

● Attribute minimization is **forbidden**

● The id attribute **replaces** the name attribute ● The XHTML DTD defines **mandatory** elements

Attribute Names Must Be In Lower Case

This is wrong:

<table WIDTH="100%"> This is correct:

<table width="100%">

Attribute Values Must Be Quoted

This is wrong:

<table width=100%> This is correct:

<table width="100%">

Attribute Minimization Is Forbidden

This is wrong:

<input checked>

<input readonly>

<input disabled>

```
<option selected>
<frame noresize>
```

This is correct:

```
<input checked="checked" />
<input readonly="readonly" />
<input disabled="disabled" />
<option selected="selected" />
<frame noresize="noresize" />
```

Here is a list of the minimized attributes in HTML and how they should be written in XHTML: **HTML**

compact checked declare readonly disabled selected defer

ismap

nohref

noshade nowrap multiple noresize

XHTML

compact="compact" checked="checked" declare="declare" readonly="readonly" disabled="disabled" selected="selected" defer="defer"

ismap="ismap"

nohref="nohref"
noshade="noshade" nowrap="nowrap" multiple="multiple" noresize="noresize"

The id Attribute Replaces The name Attribute

HTML 4.01 defines a name attribute for the elements a, applet, frame, iframe, img, and map. In XHTML the name attribute is deprecated. Use id instead.
This is wrong:

```
<img src="picture.gif" name="picture1" />
```

This is correct:

```
<img src="picture.gif" id="picture1" />
```

Note: To interoperate with older browsers for a while, you should use both name and id, with identical attribute values, like this:

```
<img src="picture.gif" id="picture1" name="picture1" />
```

IMPORTANT Compatibility Note:

To make your XHTML compatible with today's browsers, you should add an extra space before the "/" symbol.

The Lang Attribute

The lang attribute applies to almost every XHTML element. It specifies the language of the content within an element.

If you use the lang attribute in an element, you must add the xml:lang attribute, like this: <div lang="no" xml:lang="no">Heia Norge!</div>

Mandatory XHTML Elements

All XHTML documents must have a DOCTYPE declaration. The html, head and body elements must be present, and the title must be present inside the head element.
This is a minimum XHTML document template:

```
<!DOCTYPE Doctype goes here>

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

<title>Title goes here</title>

</head>

<body>

</body>
```

```
</html>
```

Note: The DOCTYPE declaration is not a part of the XHTML document itself. It is not an XHTML element, and it should not have a closing tag.

Note: The xmlns attribute inside the <html> tag is required in XHTML. However, the validator on w3.org does not complain when this attribute is missing in an XHTML document. This is because "xmlns=http://www.w3.org/1999/xhtml" is a fixed value and will be added to the <html> tag even if you do not include it.

You will learn more about the XHTML document type definition in the next chapter.

XHTML DTD

Previous Next**The XHTML standard defines three Document Type Definitions. The most common is the XHTML Transitional.**

<!DOCTYPE> Is Mandatory

An XHTML document consists of three main parts:

- the DOCTYPE
- the Head
- the Body

The basic document structure is:

```
<!DOCTYPE ...>
<html>
<head>
<title>... </title>
</head>
<body> ... </body>
</html>
```

The DOCTYPE declaration should always be the first line in an XHTML document.

An XHTML Example

This is a simple (minimal) XHTML document:

```
<!DOCTYPE html
PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd"> <html>
<head>
```

```
<title>simple document</title>
</head>
<body>
<p>a simple paragraph</p>
</body>
</html>
```

The DOCTYPE declaration defines the document type:

```
<!DOCTYPE html
PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
```

The rest of the document looks like HTML:

```
<html>
<head>
<title>simple document</title>
</head>
```

```
<body>
<p>a simple paragraph</p>
</body>
</html>
```

The 3 Document Type Definitions

- DTD specifies the syntax of a web page in SGML.

- DTD is used by SGML applications, such as HTML, to specify rules that apply to the markup of documents of a particular type, including a set of element and entity declarations.
- XHTML is specified in an SGML document type definition or 'DTD'.
- An XHTML DTD describes in precise, computer-readable language, the allowed syntax and grammar of XHTML markup.

There are currently 3 XHTML document types:

- STRICT
- TRANSITIONAL
- FRAMESET

XHTML 1.0 specifies three XML document types that correspond to three DTDs: Strict, Transitional, and Frameset.

XHTML 1.0 Strict

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
```

Use this when you want really clean markup, free of presentational clutter. Use this together with Cascading Style Sheets.

XHTML 1.0 Transitional

```
<!DOCTYPE html
PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

Use this when you need to take advantage of HTML's presentational features and when you want to support browsers that don't understand Cascading Style Sheets.

XHTML 1.0 Frameset

```
<!DOCTYPE html
PUBLIC "-//W3C//DTD XHTML 1.0 Frameset//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-frameset.dtd">
```

Use this when you want to use HTML Frames to partition the browser window into two or more frames.

XHTML HowTo

Previous Next

How W3Schools Was Converted To XHTML

W3Schools was converted from HTML to XHTML the weekend of 18. and 19. December 1999, by Hege Refsnes and Ståle Refsnes.

To convert a Web site from HTML to XHTML, you should be familiar with the XHTML syntax rules of the previous chapters. The following steps were executed (in the order listed below):

A DOCTYPE Definition Was Added

The following DOCTYPE declaration was added as the first line of every page:

```
<!DOCTYPE html PUBLIC
"-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

Note that we used the transitional DTD. We could have chosen the strict DTD, but found it a little too "strict", and a little too hard to conform to.

A Note About The DOCTYPE

Your pages must have a DOCTYPE declaration if you want them to validate as correct XHTML. Be aware however, that newer browsers (like Internet Explorer 6) might treat your document differently depending on the <!DOCTYPE> declaration. If the browser reads a document with a DOCTYPE, it might treat the document as "correct". Malformed XHTML might fall over and display differently than without a DOCTYPE.

Lower Case Tag And Attribute Names

Since XHTML is case sensitive, and since XHTML only accepts lower case HTML tags and attribute names, a general search and replace function was executed to replace all upper case tags with lowercase tags. The same was done for attribute names. We have always tried to use lower case names in our Web, so the replace function did not produce many real substitutions.

All Attributes Were Quoted

Since the W3C XHTML 1.0 Recommendation states that all attribute values must be quoted, every page in the web was checked to see that attributes values were properly quoted. This was a time-consuming job, and we will surely never again forget to put quotes around our attribute values.

Empty Tags: <hr> , <br> and <img>

Empty tags are not allowed in XHTML. The <hr> and <br> tags should be replaced with <hr /> and <br />. This produced a problem with Netscape that misinterpreted the <br/> tag. We don't know why, but changing it to <br /> worked fine. After that discovery, a general search and replace function was executed to swap the tags.
A few other tags (like the <img> tag) were suffering from the same problem as above. We decided not to close the <img> tags with </img>, but with /> at the end of the tag. This was done manually.

The Web Site Was Validated

After that, all pages were validated against the official W3C DTD with this link: XHTML Validator. A few more errors were found and edited manually. The most common error was missing </li> tags in lists. Should we have used a converting tool? Well, we could have used **TIDY**.
Dave Raggett's HTML TIDY is a free utility for cleaning up HTML code. It also works great on the hard-to-read markup generated by specialized HTML editors and conversion tools, and it can help you identify where you need to pay further attention on making your pages more accessible to people with disabilities.
The reason why we didn't use Tidy? We knew about XHTML when we started writing this web site. We knew that we had to use lowercase tag names and that we had to quote our attributes. So when the time came (to do the conversion), we simply had to test our pages against the W3C XHTML validator and correct the few mistakes. AND - we have learned a lot about writing "tidy" HTML code.

XHTML Validation

Previous Next **An XHTML document is validated against a Document Type Definition.**

Validate XHTML With A DTD

An XHTML document is validated against a Document Type Definition (DTD). Before an XHTML file can be properly validated, a correct DTD must be added as the first line of the file. The Strict DTD includes elements and attributes that have not been deprecated or do not appear in framesets:

!DOCTYPE html PUBLIC

```
"-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd"
```

The Transitional DTD includes everything in the strict DTD plus deprecated elements and attributes: !

```
DOCTYPE html PUBLIC
"-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd"
```

The Frameset DTD includes everything in the transitional DTD plus frames as well: !DOCTYPE html

```
PUBLIC
"-//W3C//DTD XHTML 1.0 Frameset//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-frameset.dtd"
```

This is a simple XHTML document:

```
<!DOCTYPE html
PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html>
```

```
<head>
<title>simple document</title>
</head>
<body>
<p>a simple paragraph</p>
</body>
</html>
```

XHTML Modularization

Previous Next

The XHTML modularization model defines the modules of XHTML.

Why XHTML Modularization?

XHTML is a simple, but large language. XHTML contains most of the functionality a web developer will need.
For some purposes XHTML is too large and complex, and for other purposes it is much too simple. By splitting XHTML into modules, the W3C (World Wide web Consortium) has created small and well-defined

sets of XHTML elements that can be used separately for simple devices as well as combined with other XML standards into larger and more complex applications. With modular XHTML, product and application designers can:

● Choose the elements to be supported by a device using standard XHTML building blocks. ● Add extensions to XHTML, using XML, without breaking the XHTML standard. ● Simplify XHTML for devices like hand held computers, mobile phones, TV, and home

appliances.

● Extend XHTML for complex applications by adding new XML functionality (like MathML, SVG, Voice and Multimedia).

● Define XHTML profiles like XHTML Basic (a subset of XHTML for mobile devices).

XHTML Modules

W3C has split the definition of XHTML into 28 modules: **Module name** Applet Module Base Module

Description
Defines the deprecated* applet element. Defines the base element.
Basic Forms Module

Basic Tables Module

Bi-directional Text Module Client Image Map Module Edit Module

Forms Module

Frames Module

Hypertext Module

Iframe Module

Image Module

Intrinsic Events Module Legacy Module

Link Module

List Module

Metainformation Module Name Identification Module Object Module

Presentation Module

Scripting Module

Server Image Map Module Structure Module

Style Attribute Module Style Sheet Module

Tables Module

Target Module

Text Module

Defines the basic forms elements.

Defines the basic table elements.

Defines the bdo element.

Defines browser side image map elements. Defines the editing elements del and ins. Defines all elements used in forms.

Defines the frameset elements.

Defines the a element.

Defines the iframe element.

Defines the img element.

Defines event attributes like onblur and onchange. Defines deprecated* elements and attributes. Defines the link element.

Defines the list elements ol, li, ul, dd, dt, and dl. Defines the meta element.

Defines the deprecated* name attribute. Defines the object and param elements.

Defines presentation elements like b and i. Defines the script and noscript elements. Defines server side image map elements. Defines the elements html, head, title and body. Defines the style attribute.

Defines the style element.

Defines the elements used in tables.

Defines the target attribute.

Defines text container elements like p and h1.

* Deprecated elements should not be used in XHTML.

You Have Learned XHTML, Now What?

Previous Next

XHTML Summary

This tutorial has taught you how to create stricter and cleaner HTML pages.

You have learned that all XHTML elements must be properly nested, XHTML documents must be well-formed, all tag names must be in lowercase, and that all XHTML elements must be closed. You have also learned that all XHTML documents must have a DOCTYPE declaration, and that the html, head, title, and body elements must be present.

For more information on XHTML, please look at our XHTML reference.

Now You Know XHTML, What's Next?

The next step is to learn CSS and JavaScript.

CSS

CSS is used to control the style and layout of multiple Web pages all at once.

With CSS, all formatting can be removed from the HTML document and stored in a separate file. CSS gives you total control of the layout, without messing up the document content. To learn how to create style sheets, please visit our CSS tutorial.

JavaScript

JavaScript can make your web site more dynamic.

A static web site is nice when you just want to show flat content, but a dynamic web site can react to events and allow user interaction.

JavaScript is the most popular scripting language on the internet and it works with all major browsers.

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