XML Tutorial

« W3Schools Home Next Chapter »

XML stands for e**X**tensible **M**arkup **L**anguage.

XML is designed to transport and store data.

XML is important to know, and very easy to learn.

XML Document Example

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

Introduction to XML

« Previous Next Chapter »

XML was designed to transport and store data.

HTML was designed to display data.

What You Should Already Know

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

- HTML
- JavaScript

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

What is XML?

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

The Difference Between XML and HTML

XML is not a replacement for HTML.

XML and HTML were designed with different goals:

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

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

XML Does not DO Anything

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

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

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

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

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

XML is Just Plain Text

XML is nothing special. It is just plain text. Software that can handle plain text can also handle XML.

However, XML-aware applications can handle the XML tags specially. The functional meaning of the tags depends on the nature of the application.

With XML You Invent Your Own Tags

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

That is because the XML language has no predefined tags.

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

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

XML is Not a Replacement for HTML

XML is a complement to HTML.

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

My best description of XML is this:

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

XML is a W3C Recommendation

XML became a W3C Recommendation 10. February 1998.

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

XML is Everywhere

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

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

XML is everywhere. It is the most common tool for data transmissions between all sorts of applications, and is becoming more and more popular in the area of storing and describing information.

XML Sample 1

```
<?xml version="1.0"?>
<dining-room>
  <manufacturer>The Wood Shop</manufacturer>
  <table type="round" wood="maple">
      <price>$199.99</price>
  </table>
  <chair wood="maple">
     <quantity>6</quantity>
      <price>$39.99</price>
  </chair>
</dining-room>
```

XML Sample 2

```
<?xml version="1.0"?>
<SONG>
    <TITLE>Mambo</TITLE>
    <COMPOSER>Enrique Garcia</COMPOSER>
    <PRODUCER>Enrique Garcia</PRODUCER>
    <PUBLISHER>Sony Music Entertainment</PUBLISHER>
    <LENGTH>3:46</LENGTH>
    <YEAR>1991</YEAR>
    <ARTIST>Azucar Moreno</ARTIST>
</SONG>
```

XML Sample 3

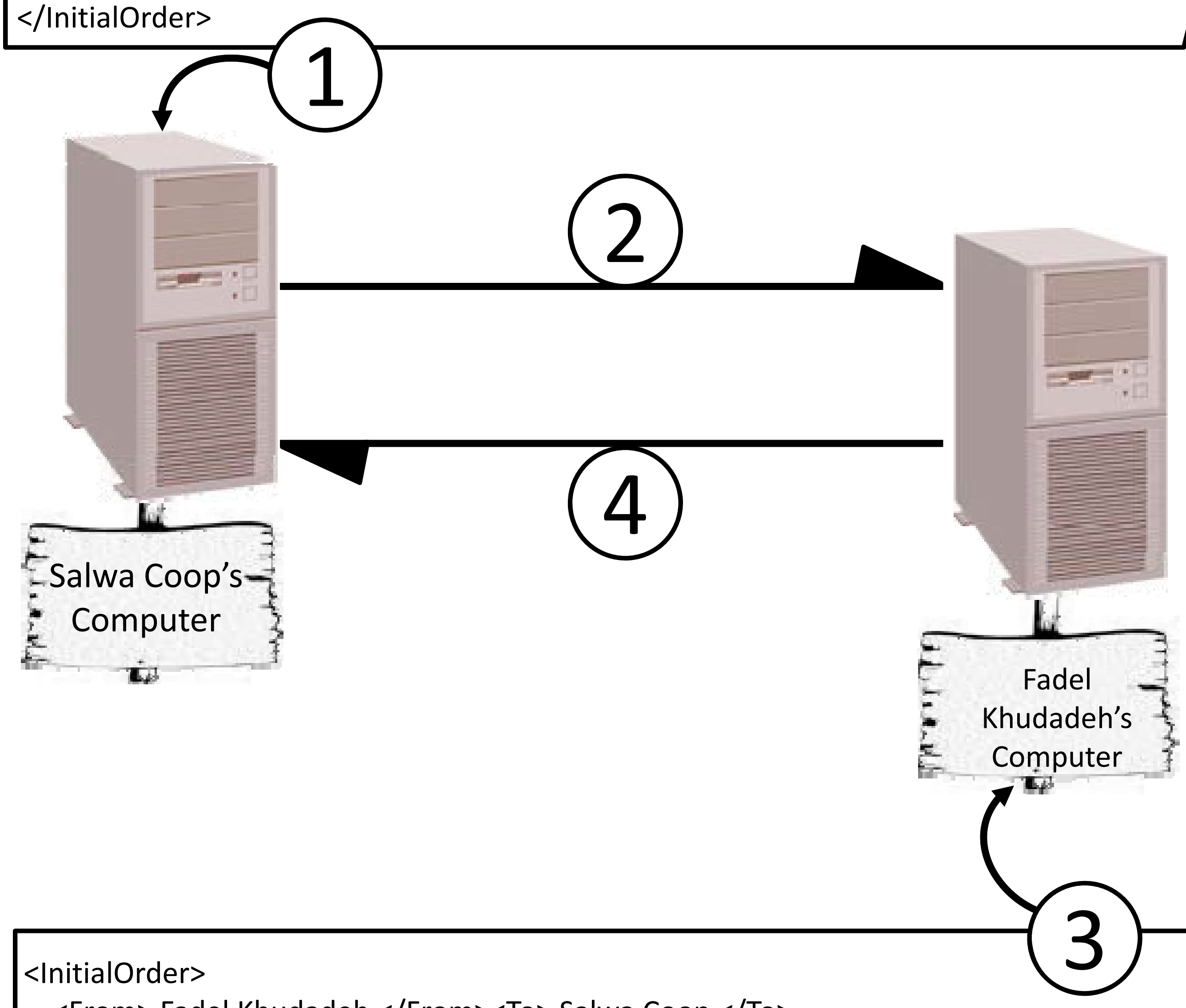
```
<?xml version="1.0" encoding="UTF-8"?>
<DOCUMENT>
      <GREETING>Hello from XML</GREETING>
      <MESSAGE>Welcome to Programing XML in Java</MESSAGE>
</DOCUMENT>
```

XML Sample 4

```
<?xml version="1.0"?>
<SCHOOL>
    <CLASS type="seminar">
         <CLASS_TITLE>XML In The Real World</CLASS_TITLE>
         <CLASS_NUMBER>6.031</CLASS_NUMBER>
         <SUBJECT>XML</SUBJECT>
         <START_DATE>6/1/2002</START_DATE>
         <STUDENTS>
               <STUDENT status="attending">
                    <FIRST_NAME>Edward</FIRST_NAME>
                    <LAST_NAME>Samson</LAST_NAME>
               </STUDENT>
               <STUDENT status="withdrawn">
                    <FIRST_NAME>Ernestine</FIRST_NAME>
                    <LAST_NAME>Johnson</LAST_NAME>
               </STUDENT>
         </STUDENTS>
     </CLASS>
</SCHOOL>
```

XML For Communication

```
<InitialOrder>
   <From> Salwa Coop </From><To>  Fadel Khudadeh </To>
    <Items>
      <Item><name>Pepsi</name><Qty>10</Qty><UnitPrice>??</UnitPrice></Item>
      <Item><name>Choco.</name><Qty>8</Qty><UnitPrice>??</UnitPrice></Item>
    </Items>
 </InitialOrder>
```



```
<InitialOrder>
   <From> Fadel Khudadeh </From><To> Salwa Coop </To>
    <Items>
      <Item><name>Pepsi</name><Qty>10</Qty><UnitPrice>.095</UnitPrice></Item>
      <Item><name>Choco.</name><Qty>8</Qty><UnitPrice>.660</UnitPrice></Item>
    </Items>
 </InitialOrder>
```

How Can XML be Used?

« Previous Next Chapter »

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

XML Separates Data from HTML

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

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

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

You will learn more about this in a later chapter of this tutorial.

XML Simplifies Data Sharing

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

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

This makes it much easier to create data that different applications can share.

XML Simplifies Data Transport

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

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

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

XML Simplifies Platform Changes

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

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

XML Makes Your Data More Available

Since XML is independent of hardware, software and application, XML can make your data more available and useful.

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

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

XML is Used to Create New Internet Languages

A lot of new Internet languages are created with XML.

Here are some examples:

- XHTML the latest version of HTML
- WSDL for describing available web services

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

If Developers Have Sense

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

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

We can only pray that all the software vendors will agree.

XML Tree

« Previous   Next Chapter »

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

An Example XML Document

XML documents use a self-describing and simple syntax:

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

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

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

```
<note>
```

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

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

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

```
</note>
```

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

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

XML Documents Form a Tree Structure

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

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

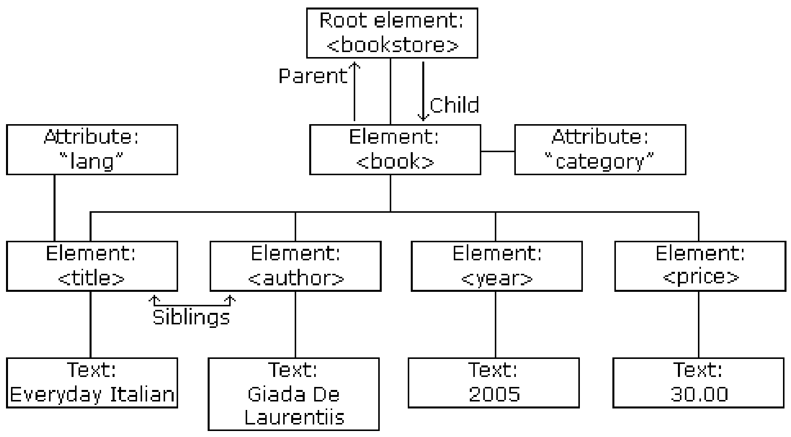
All elements can have sub elements (child elements):

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

The terms parent, child, and sibling are used to describe the relationships between elements. Parent elements have children. Children on the same level are called siblings (brothers or sisters).

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

Example:



The image above represents one book in the XML below:

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

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

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

XML Syntax Rules

« Previous　　Next Chapter »

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

All XML Elements Must Have a Closing Tag

In HTML, you will often see elements that don't have a closing tag:

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

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

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

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

XML Tags are Case Sensitive

XML elements are defined using XML tags.

XML tags are case sensitive. With XML, the tag <Letter> is different from the tag <letter>.

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

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

Note: "Opening and closing tags" are often referred to as "Start and end tags". Use whatever you prefer. It is exactly the same thing.

XML Elements Must be Properly Nested

In HTML, you might see improperly nested elements:

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

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

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

In the example above, "Properly nested" simply means that since the <i> element is opened inside the <b> element, it must be closed inside the <b> element.

XML Documents Must Have a Root Element

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

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

XML Attribute Values Must be Quoted

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

In XML the attribute value must always be quoted. Study the two XML documents below. The first one is incorrect, the second is correct:

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

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

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

Entity References

Some characters have a special meaning in XML.

If you place a character like "<" inside an XML element, it will generate an error because the parser interprets it as the start of a new element.

This will generate an XML error:

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

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

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

There are 5 predefined entity references in XML:

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

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

Comments in XML

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

<!-- This is a comment -->

White-space is Preserved in XML

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

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

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

XML Stores New Line as LF

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

XML Elements

« Previous Next Chapter »

An XML document contains XML Elements.

What is an XML Element?

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

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

```
<bookstore>
  <book category="CHILDREN">
    <title>Harry Potter</title>
    <author>J K. Rowling</author>
    <year>2005</year>
    <price>29.99</price>
  </book>
  <book category="WEB">
    <title>Learning XML</title>
    <author>Erik T. Ray</author>
    <year>2003</year>
    <price>39.95</price>
  </book>
</bookstore>
```

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

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

XML Naming Rules

XML elements must follow these naming rules:

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

Any name can be used, no words are reserved.

Best Naming Practices

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

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

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

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

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

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

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

XML Elements are Extensible

XML elements can be extended to carry more information.

Look at the following XML example:

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

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

MESSAGE

To: Tove
From: Jani

Don't forget me this weekend!

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

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

Should the application break or crash?

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

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

XML Attributes

« Previous Next Chapter »

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

Attributes provide additional information about elements.

XML Attributes

From HTML you will remember this: <img src="computer.gif">. The "src" attribute provides additional information about the <img> element.

In HTML (and in XML) attributes provide additional information about elements:

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

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

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

XML Attributes Must be Quoted

Attribute values must always be enclosed in quotes, but either single or double quotes can be used. For a person's sex, the person tag can be written like this:

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

or like this:

```
<person sex='female'>
```

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

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

or you can use character entities:

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

XML Elements vs. Attributes

Take a look at these examples:

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

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

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

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

My Favorite Way

The following three XML documents contain exactly the same information:

A date attribute is used in the first example:

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

A date element is used in the second example:

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

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

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

Avoid XML Attributes?

Some of the problems with using attributes are:

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

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

Don't end up like this:

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

XML Attributes for Metadata

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

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

The ID above is just an identifier, to identify the different notes. It is not a part of the note itself.

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

DTD Tutorial

The purpose of a DTD (Document Type Definition) is to define the legal building blocks of an XML document.

A DTD defines the document structure with a list of legal elements and attributes.

DTD Newspaper Example

```
<!DOCTYPE NEWSPAPER [

<!ELEMENT NEWSPAPER (ARTICLE+)>
<!ELEMENT ARTICLE (HEADLINE,BYLINE,LEAD,BODY,NOTES)>
<!ELEMENT HEADLINE (#PCDATA)>
<!ELEMENT BYLINE (#PCDATA)>
<!ELEMENT LEAD (#PCDATA)>
<!ELEMENT BODY (#PCDATA)>
<!ELEMENT NOTES (#PCDATA)>

<!ATTLIST ARTICLE AUTHOR CDATA #REQUIRED>
<!ATTLIST ARTICLE EDITOR CDATA #IMPLIED>
<!ATTLIST ARTICLE DATE CDATA #IMPLIED>
<!ATTLIST ARTICLE EDITION CDATA #IMPLIED>

]>
```

« W3Schools Home　　Next Chapter »

Introduction to DTD

« Previous　　　　Next Chapter »

A Document Type Definition (DTD) defines the legal building blocks of an XML document. It defines the document structure with a list of legal elements and attributes.

A DTD can be declared inline inside an XML document, or as an external reference.

Internal DTD Declaration

If the DTD is declared inside the XML file, it should be wrapped in a DOCTYPE definition with the following syntax:

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

Example XML document with an internal DTD:

```
<?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>
```

Open the XML file above in your browser (select "view source" or "view page source" to view the DTD)

The DTD above is interpreted like this:

- **!DOCTYPE note** defines that the root element of this document is note
- **!ELEMENT note** defines that the note element contains four elements: "to,from,heading,body"
- **!ELEMENT to** defines the to element to be of type "#PCDATA"
- **!ELEMENT from** defines the from element to be of type "#PCDATA"
- **!ELEMENT heading** defines the heading element to be of type "#PCDATA"
- **!ELEMENT body** defines the body element to be of type "#PCDATA"

External DTD Declaration

If the DTD is declared in an external 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, and select view source):

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

And this is the file "note.dtd" which contains 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 a DTD, each of your XML files can carry a description of its own format.

With a DTD, independent groups of people can agree to use a standard 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

The main building blocks of both XML and HTML documents are elements.

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 building blocks:

- Elements
- Attributes
- Entities
- PCDATA
- CDATA

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>some text</body>

<message>some text</message>
```

Attributes

Attributes provide **extra information about elements**.

Attributes are always placed inside the opening 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

Some characters have a special meaning in XML, like the less than sign (<) that defines the start of an XML tag.

Most of you know the HTML entity: "&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. **The text will be examined by the parser for entities and markup**.

Tags inside the text will be treated as markup and entities will be expanded.

However, parsed character data should not contain any &, <, or > characters; these need to be represented by the &amp; &lt; and &gt; entities, respectively.

CDATA

CDATA 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 Chapter »

In a DTD, elements are declared with an ELEMENT declaration.

Declaring Elements

In a DTD, XML elements are declared with an element declaration with 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 Parsed Character Data

Elements with only parsed 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 declared with the name of the children elements inside parentheses:

```
<!ELEMENT element-name (child1)>
or
<!ELEMENT element-name (child1,child2,...)>

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 is:

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

Declaring Only One Occurrence of an Element

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

Example:

<!ELEMENT note (message)>
```

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

Declaring Minimum One Occurrence of an 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 an 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 an 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 time 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 data, "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>

DTD example:

<!ATTLIST payment type CDATA "check">

XML example:

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

The **attribute-type** can be one of the following:

Type	Description
CDATA	The value is character data
(*en1*|*en2*|..)	The value must be one from an enumerated list
ID	The value is a unique id
IDREF	The value is the id of another element
IDREFS	The value is a list of other ids
NMTOKEN	The value is a valid XML name
NMTOKENS	The value is a list of valid XML names
ENTITY	The value is an entity
ENTITIES	The value is a list of entities
NOTATION	The value is a name of a notation
xml:	The value is a predefined xml value

The **default-value** can be one of the following:

Value	Explanation
value	The default value of the attribute
#REQUIRED	The attribute is required
#IMPLIED	The attribute is not required
#FIXED *value*	The attribute value is fixed

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.

#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.

#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.

#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>
```

Example

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

XML example:
<payment type="check" />
or
<payment type="cash" />
```

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

XML Elements vs. Attributes

« Previous   Next Chapter »

In XML, there are no rules about when to use attributes, and when to use child elements.

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 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 DTD

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 - Entities

« Previous Next Chapter »

Entities are variables used to define shortcuts to standard text or special characters.

- Entity references are references to entities
- Entities can be declared internal or external

An Internal Entity Declaration

Syntax

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

Example

```
DTD Example:

<!ENTITY writer "Donald Duck.">
<!ENTITY copyright "Copyright W3Schools.">

XML example:

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

Note: An entity has three parts: an ampersand (&), an entity name, and a semicolon (;).

An External Entity Declaration

Syntax

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

Example

```
DTD Example:

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

XML example:

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

DTD - Examples from the internet

« Previous   Next Chapter »

TV Schedule DTD

By David Moisan. Copied from http://www.davidmoisan.org/

```
<!DOCTYPE TVSCHEDULE [

<!ELEMENT TVSCHEDULE (CHANNEL+)>
<!ELEMENT CHANNEL (BANNER,DAY+)>
<!ELEMENT BANNER (#PCDATA)>
<!ELEMENT DAY (DATE,(HOLIDAY|PROGRAMSLOT+)+)>
<!ELEMENT HOLIDAY (#PCDATA)>
<!ELEMENT DATE (#PCDATA)>
<!ELEMENT PROGRAMSLOT (TIME,TITLE,DESCRIPTION?)>
<!ELEMENT TIME (#PCDATA)>
<!ELEMENT TITLE (#PCDATA)>
<!ELEMENT DESCRIPTION (#PCDATA)>

<!ATTLIST TVSCHEDULE NAME CDATA #REQUIRED>
<!ATTLIST CHANNEL CHAN CDATA #REQUIRED>
<!ATTLIST PROGRAMSLOT VTR CDATA #IMPLIED>
<!ATTLIST TITLE RATING CDATA #IMPLIED>
<!ATTLIST TITLE LANGUAGE CDATA #IMPLIED>
]>
```

Newspaper Article DTD

Copied from http://www.vervet.com/

```
<!DOCTYPE NEWSPAPER [

<!ELEMENT NEWSPAPER (ARTICLE+)>
<!ELEMENT ARTICLE (HEADLINE,BYLINE,LEAD,BODY,NOTES)>
<!ELEMENT HEADLINE (#PCDATA)>
<!ELEMENT BYLINE (#PCDATA)>
<!ELEMENT LEAD (#PCDATA)>
<!ELEMENT BODY (#PCDATA)>
<!ELEMENT NOTES (#PCDATA)>

<!ATTLIST ARTICLE AUTHOR CDATA #REQUIRED>
<!ATTLIST ARTICLE EDITOR CDATA #IMPLIED>
<!ATTLIST ARTICLE DATE CDATA #IMPLIED>
<!ATTLIST ARTICLE EDITION CDATA #IMPLIED>

<!ENTITY NEWSPAPER "Vervet Logic Times">
<!ENTITY PUBLISHER "Vervet Logic Press">
<!ENTITY COPYRIGHT "Copyright 1998 Vervet Logic Press">

]>
```

Product Catalog DTD

Copied from http://www.vervet.com/

```
<!DOCTYPE CATALOG [

<!ENTITY AUTHOR "John Doe">
<!ENTITY COMPANY "JD Power Tools, Inc.">
<!ENTITY EMAIL "jd@jd-tools.com">

<!ELEMENT CATALOG (PRODUCT+)>
<!ELEMENT PRODUCT
(SPECIFICATIONS+,OPTIONS?,PRICE+,NOTES?)>
<!ATTLIST PRODUCT
NAME CDATA #IMPLIED
CATEGORY (HandTool|Table|Shop-Professional) "HandTool"
PARTNUM CDATA #IMPLIED
PLANT (Pittsburgh|Milwaukee|Chicago) "Chicago"
INVENTORY (InStock|Backordered|Discontinued) "InStock">

<!ELEMENT SPECIFICATIONS (#PCDATA)>
<!ATTLIST SPECIFICATIONS
WEIGHT CDATA #IMPLIED
POWER CDATA #IMPLIED>

<!ELEMENT OPTIONS (#PCDATA)>
<!ATTLIST OPTIONS
FINISH (Metal|Polished|Matte) "Matte"
ADAPTER (Included|Optional|NotApplicable) "Included"
CASE (HardShell|Soft|NotApplicable) "HardShell">

<!ELEMENT PRICE (#PCDATA)>
<!ATTLIST PRICE
MSRP CDATA #IMPLIED
WHOLESALE CDATA #IMPLIED
STREET CDATA #IMPLIED
SHIPPING CDATA #IMPLIED>

<!ELEMENT NOTES (#PCDATA)>

]>
```

You Have Learned DTD, Now What?

« Previous Next Chapter »

DTD Summary

This tutorial has taught you how to describe the structure of an XML document.

You have learned how to use a DTD to define the legal elements of an XML document, and how a DTD can be declared inside your XML document, or as an external reference.

You have learned how to declare the legal elements, attributes, entities, and CDATA sections for XML documents.

You have also seen how to validate an XML document against a DTD.

Now You Know DTD, What's Next?

The next step is to learn about XML Schema.

XML Schema is used to define the legal elements of an XML document, just like a DTD. We think that very soon XML Schemas will be used in most Web applications as a replacement for DTDs.

XML Schema is an XML-based alternative to DTD.

Unlike DTD, XML Schemas has support for data types and namespaces.

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