DTD Tutorial

« W3Schools Home

Next Chapter »

A Document Type Definition (DTD) defines the structure and the legal elements and attributes of an XML document.

A DTD can be declared inside an XML document or in an external file.

What You Should Already Know

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

- HTML
- XML

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

Why Use a DTD?

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.

Internal DTD Declaration

If the DTD is declared inside the XML file, it must be wrapped inside the <!DOCTYPE> definition:

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>
```

View XML file »

In the XML file, select "view 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 must contain 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, the <!DOCTYPE> definition must contain a reference to the DTD file:

XML document with a reference to an external DTD

```
<?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>
```

View XML file »

And here 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)>
```

« W3Schools Home

Next Chapter »

DTD - XML Building Blocks

« Previous

Next Chapter »

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 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: " ". 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 & < and > 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.

« Previous

Next Chapter »

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 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

```
<!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

```
<!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.

« Previous

Next Chapter »

DTD - Attributes

« Previous

Next Chapter »

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 attribute-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 **attribute-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 optional
#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.

« Previous

Next Chapter »

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.

« Previous

Next Chapter »

DTD - Entities

« Previous

Next Chapter »

Entities are used to define shortcuts to special characters.

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>
```

« Previous

Next Chapter »

DTD Validation

« Previous

Next Chapter »

With Internet Explorer you 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, you can retrieve the error code, the error text, or even the line that caused the error.

Note: The load() method is used for files, while the loadXML() method is used for strings.

Example

```
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);
document.write("<br />Error Reason: ");
document.write(xmlDoc.parseError.reason);
```

```
document.write("<br />Error Line: ");
document.write(xmlDoc.parseError.line);
```

Try it yourself »

Look at the XML file

Turn Validation Off

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

Example

```
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 check your xml files, you can syntax-check any XML file here.

The parseError Object

You can read more about the parseError object in our XML DOM tutorial.

« Previous

Next Chapter »

DTD Examples

« 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)>

]>
```

« Previous

Next Chapter »