

# Introduction to DTD

The purpose of a 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.

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

## 1 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.....

## 2 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>Tove</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)>
```

---

## 3 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

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

### 3.1 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
- Tags
- Attributes
- Entities
- PCDATA
- CDATA

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

## 3.2 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".

## 3.3 Tags

Tags are used **to markup elements**.

A starting tag like `<element_name>` marks up the beginning of an element, and an ending tag like `</element_name>` marks up the end of an element.

Examples:

```
body element marked up with body tags:
<body>body text in between</body>.

message element marked up with message tags:
<message>some message in between</message>
```

## 3.4 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:

```

```

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 "/".

## 3.5 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;	'

### 3.6 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.

### 3.7 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

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

### 3.8 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)>
```

### 3.9 Empty elements

Empty elements are declared with the category keyword EMPTY:

```
<!ELEMENT element-name EMPTY>

example:
<!ELEMENT br EMPTY>
XML example:
<br />
```

### 3.10 Elements with only character data

Elements with only character data are declared with #PCDATA inside parentheses:

```
<!ELEMENT element-name (#PCDATA)>

example:
<!ELEMENT from (#PCDATA)>
```

### 3.11 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>
```

### 3.12 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)>
```

### 3.13 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.

### 3.14 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.

### 3.15 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.

### 3.16 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.

### 3.17 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.

### 3.18 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.

## 3.19 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	Explanation
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 have the following values:

Value	Explanation
value	The default value of the attribute
#REQUIRED	The attribute value must be included in the element
#IMPLIED	The attribute does not have to be included
#FIXED value	The attribute value is fixed

## 3.20 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.

### 3.21 #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.

### 3.22 #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.

### 3.23 #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.

### 3.24 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

Entities are variables used to define shortcuts to common text.

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

## 3.25 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>
```

## 3.26 External Entity Declaration

```
Syntax:  
<!ENTITY entity-name SYSTEM "URI/URL">  
  
DTD Example:  
<!ENTITY writer  
  SYSTEM "http://www.w3schools.com/entities/entities.xml">  
<!ENTITY copyright  
  SYSTEM "http://www.w3schools.com/entities/entities.dtd">  
XML example:  
<author>&writer;&copyright;</author>
```

# DTD Validation

Internet Explorer 5.0 can validate your XML against a DTD.

## 3.27 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)
document.write("<br>Error Reason: ")
document.write(xmlDoc.parseError.reason)
document.write("<br>Error Line: ")
document.write(xmlDoc.parseError.line)
```

[Try it Yourself](#) or just [look at the XML file](#)

## 3.28 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](#)

## 3.29 A general XML Validator

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

## 3.30 The parseError Object

You can read more about the `parseError` object in our XML [DOM tutorial](#).

# DTD - Examples from the Net

## 3.31 TV Schedule DTD

By David Moisan. Copied from his Web: <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>  
  
>
```

## 3.32 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">  
  
>
```

### 3.33 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)>  
  
>
```