Document Type Definition

Table of Contents

[Internal and External DTD Declarations 2](#_Toc100513682)

[Building Blocks 3](#_Toc100513683)

[Elements 4](#_Toc100513684)

[Attributes 6](#_Toc100513685)

[Entities 9](#_Toc100513686)

[XML Parsers 10](#_Toc100513687)

[DOM Parsers 10](#_Toc100513688)

[SAX Parser 11](#_Toc100513689)

[Applications of XML 11](#_Toc100513690)

As we discussed before, a **document type definition** (DTD) lists the grammar rules for an XML document. XML files that are well-formed and also abide by these rules are said to be **valid**.

A DTD will list:

* What names can be used for elements
* Where specific elements may occur
* How the elements fit together

## Internal and External DTD Declarations

DTDs can be declared inside an XML file, called an **internal DTD declaration**, or in an external file, called an **external DTD declaration**.

The following is an internal DTD declaration:

<?xml version="1.0"?>  
<!DOCTYPE note[  
 <!ELEMENT note (to, from, heading, body)>  
 <!ELEMENT to (#PCDATA)>  
 <!ELEMENT from (#PCDATA)>  
 <!ELEMENT body (#PCDATA)>  
 ]>  
<note>  
 <to>Ash</to>  
 <from>Professor Oak</from>  
 <body>Gotta catch em all</body>  
</note>

XML

The following is an external DTD declaration:

<?xml version="1.0"?>  
<!DOCTYPE note SYSTEM "note.dtd">  
<note>  
 <to>Ash</to>  
 <from>Professor Oak</from>  
 <body>Gotta catch em all</body>  
</note>

XML

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

DTD

## Building Blocks

From the point of view of a DTD, all XML files are made up of the following building blocks:

* Elements
* Attributes
* Entities
* PCDATA
* CDATA

Among these, only PCDATA and CDATA should be new.

**PCDATA** stands for parsed character data. If we have an element that has the type PCDATA, this means that the content inside the tags of the element should be parsed by the parser. That content may contain other elements, so the element needs to be expanded and the content needs to be treated as markup.

**CDATA** refers to character data. If we have an element that has the type CDATA, this means that the content inside the tags of the element should not be parsed by the parser and should be treated as though it were plaintext.

We will now be looking at examples for the different types of building blocks.

### Elements

<!ELEMENT someelement (childelement1, childelement2)>

DTD

An **element** declaration is followed by the **name** of the element. Inside the **parentheses**, we can provide the names of the **child elements** that will be allowed. Note that this also means that the specified child elements are **required** and they must appear in this **specific order**.

With the above declaration, the child elements can only appear **once**. If we want to allow a specific child element to appear **multiple times**, we can use the + symbol.

<!ELEMENT someelement (childelement1, childelement2+)>

DTD

This still means that the child element must appear **at least once**. We can allow a child element to appear **zero or more times** by using the \* symbol, or **zero or one time** by using the ? symbol.

<!ELEMENT someelement (childelement1, childelement2\*)>

<!ELEMENT someelement (childelement1, childelement2?)>

DTD

We can declare an **either/or statement** using the | symbol. This will allow any one of the child elements to appear.

<!ELEMENT someelement (childelement1, (childelement2 | childelement3))>

DTD

Finally, we can allow **any combination** of the specified child elements using the | symbol between child elements and the \* symbol outside the parentheses.

<!ELEMENT someelement (#PCDATA | childelement1 | childelement2)\*>

DTD

If we do not want to bother with specifying exactly what the child elements will be, we can simply say that the child element is of type PCDATA, to indicate that it should be parsed, but we do not care what the content is.

<!ELEMENT someelement (#PCDATA)>

DTD

If we wish to create an **empty element**, we can do so by using the keyword EMPTY.

<!ELEMENT someelement EMPTY>

If we do not wish to specify what the type of data inside an element will be, we can use the keyword ANY.

<!ELEMENT someelement ANY>

DTD

### Attributes

Suppose we have some element already declared. We can then specify **attributes** for this element by mentioning the element name, the attribute name, the attribute data type and finally a default value.

<payment method="bkash">

XML

<!ATTLIST payment method CDATA "cash on delivery">

DTD

The attribute **data type** can be any one of the following:

|  |  |
| --- | --- |
| **Type** | **Description** |
| CDATA | Value is character data |
| (en1 | en2 | …) | Value must be one from enumerated list |
| ID | Value is a unique ID |
| Xml: | Value is a predefined XML value |

To use an enumerated list:

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

DTD

<payment type="check"/> <!-- Valid -->  
<payment type="cash"/> <!-- Valid -->

XML

The **default value** can be any 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 |

To use a default value:

<!ATTLIST square width CDATA "0">

DTD

<square width="100"/>

XML

To use a required value:

<!ATTLIST person number CDATA #REQUIRED>

DTD

<person number="5677"/> <!-- Valid -->  
<person/> <!-- Invalid -->

XML

To use an implied value:

<!ATTLIST contact fax CDATA #IMPLIED>

DTD

<contact fax = "555-667788"/> <!-- Valid -->  
<contact/> <!-- Valid -->

XML

To use a fixed value:

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

DTD

<sender company="Microsoft" /> <!-- Valid -->  
<sender company="W3Schools"> <!-- Invalid -->

XML

### Entities

**Entities** are used as shortcuts for special sets of characters. These can be of three categories:

* **General or Internal Entity Declaration** – These are used as substitutes for commonly-used segments of XML code. For examples, since we cannot use the < symbol, we use &lt; instead.
* **Parameter Entity Declaration** – These are entities that are both declared and used inside the same DTD file.

<!ENTITY %acceptable "(#PCDATA | data)\*">  
<!ELEMENT example %acceptable;>

DTD

* **External Entity Declaration** – These are entities that are declared in a DTD file but used in an XML file.

<!ENTITY favquotes SYSTEM "http://www.example.com/favstocks.xml">

DTD

<section>  
 <heading>Current Favorite Stock Picks</heading>  
 &favquotes;  
</section>

XML

## XML Parsers

An **XML Parser** pre-processes XML documents before handing over the results to an application. There are two major categories of XML parsers, DOM and SAX. Based on these, some famous parsers include JDOM, DOM4J, StAX, etc.

### DOM Parsers

**Document Object Model** (DOM) is the official W3C Standard for representing XML documents in a platform and language-neutral way. It is a collection of nodes that are organized in a hierarchical structure to allow developers to look for specific information in the structure.

Parsing an XML document with a DOM parser returns a **tree** containing all the elements of the document. The DOM provides lots of functions to examine the contents and the structure of the document.

There are several advantages to using DOM parsers. Firstly, the tree is **persistent in memory**, which allows the application to make changes to the data and structure. It can also navigate up and down the tree at any time. DOM is simpler to use.

### SAX Parser

**Simple API for XML** (SAX) is an **event-based** parser for XML documents. Unlike a DOM parser, a SAX parser does not create a tree. Instead, **events** are associated with each opening and closing tag, tag body, etc.

SAX however, has **lower memory-requirements**, since the developer can choose which tags to process. The client application can also get information when the SAX parser **notifies** the client that information is available, meaning SAX is a **PUSH API**.

SAX is useful when the developer needs to work with only some of the data in a document, but it is **difficult to code** with a SAX parser and also difficult to access **multiple data** in the same document simultaneously.

## Applications of XML

XML documents are used for **configuration** in many different frameworks and platforms. They can be used on the **server-side** and then converted to JSON for a web client. This allows validation through XML schemas.

XML is also used in SVG files and in WS-\* SOA enterprise systems.