https://www.tutorialspoint.com/dtd/index.htm:

DTD Tutorial (www.tutorialspoint.com)

XML **D**ocument **T**ype **D**eclaration commonly known as **DTD** is a way to describe precisely the XML language. DTDs check the validity of, structure and vocabulary of an XML document against the grammatical rules of the appropriate XML language.

This tutorial will teach you basics of DTD. Tutorial contains chapters discussing all the basic components of DTD with suitable examples.

Audience

This reference has been prepared for the beginners to help them to understand the basic concepts related to DTD. This tutorial will give you enough understanding on DTD from where you can take yourself to a higher level of expertise.

Prerequisites

Before proceeding with this tutorial you should have basic knowledge of XML, HTML and Javascript.

DTD Overview

XML Document Type Declaration, commonly known as DTD, is a way to describe precisely the XML language. DTDs check the validity of structure and vocabulary of an XML document against the grammatical rules of the appropriate XML language.

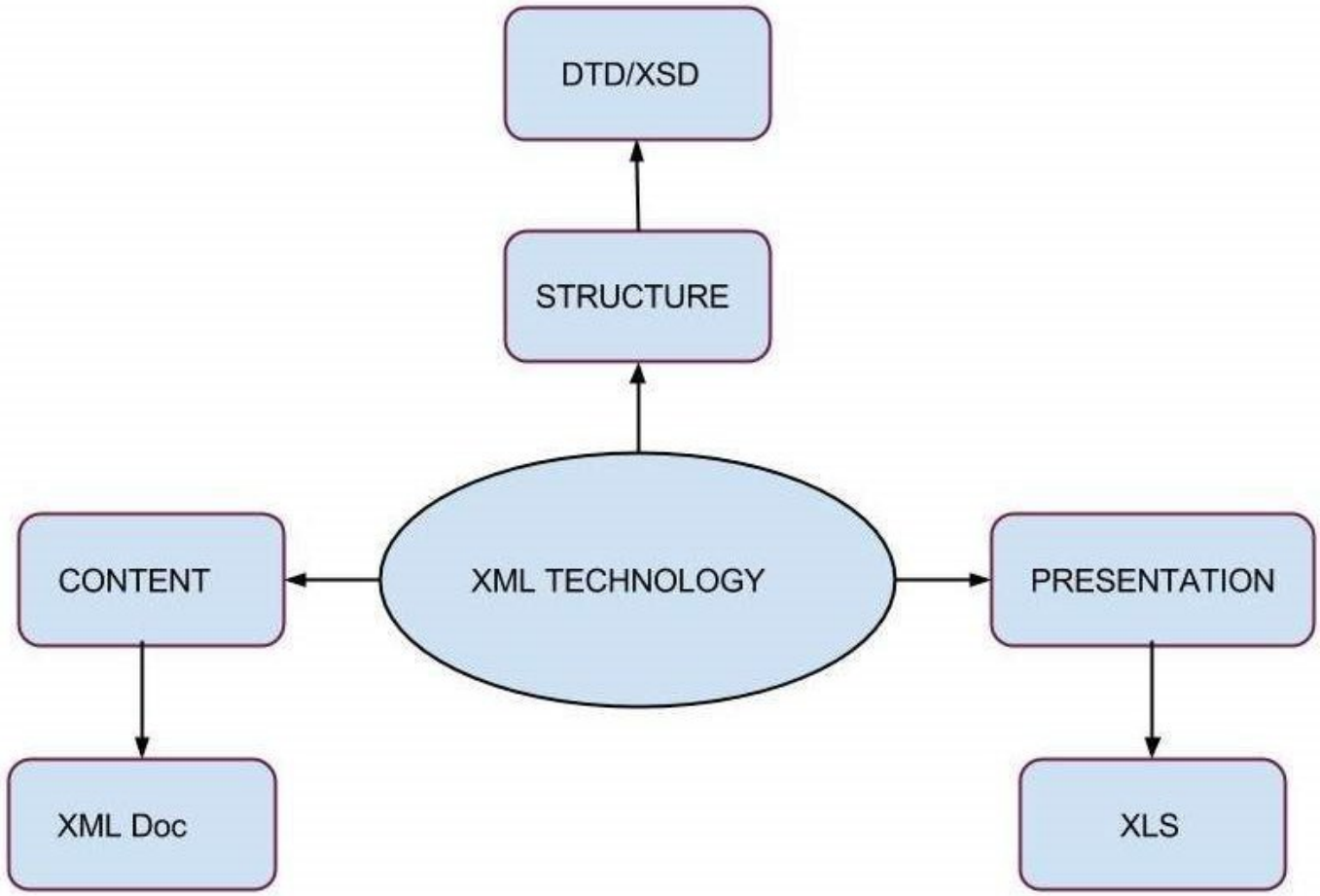
An XML document can be defined as:

- **Well-formed**: If the XML document adheres to all the general XML rules such as tags must be properly nested, opening and closing tags must be balanced, and empty tags must end with '/>', then it is called as *well-formed*.

OR

- **Valid**: An XML document said to be valid when it is not only *well-formed*, but it also conforms to available DTD that specifies which tags it uses, what attributes those tags can contain, and which tags can occur inside other tags, among other properties.

The following diagram represents that a DTD is used to structure the XML document:



Types

DTD can be classified on its declaration basis in the XML document, such as:

- Internal DTD
- External DTD

When a DTD is declared within the file it is called **Internal DTD** and if it is declared in a separate file it is called **External DTD**.

We will learn more about these in the chapter DTD Syntax

Features

Following are some important points that a DTD describes:

- the elements that can appear in an XML document.
- the order in which they can appear.
- optional and mandatory elements.

- element attributes and whether they are optional or mandatory.
- whether attributes can have default values.

Advantages of using DTD

- **Documentation -** You can define your own format for the XML files. Looking at this document a user/developer can understand the structure of the data.
- **Validation -** It gives a way to check the validity of XML files by checking whether the elements appear in the right order, mandatory elements and attributes are in place, the elements and attributes have not been inserted in an incorrect way, and so on.

Disadvantages of using DTD

- It does not support the namespaces. Namespace is a mechanism by which element and attribute names can be assigned to groups. However, in a DTD namespaces have to be defined within the DTD, which violates the purpose of using namespaces.
- It supports only the *text string data type.*
- It is not object oriented. Hence, the concept of inheritance cannot be applied on the DTDs.
- Limited possibilities to express the cardinality for elements.

An XML DTD can be either specified inside the document, or it can be kept in a separate document and then the document can be linked to the DTD document to use it.

DTD Syntax

Basic syntax of a DTD is as follows:

```
<!DOCTYPE element DTD identifier
[
   declaration1
   declaration2
   ........
]>
```

In the above syntax

- **DTD** starts with <!DOCTYPE delimiter.
- An **element** tells the parser to parse the document from the specified root element.

- **DTD identifier** is an identifier for the document type definition, which may be the path to a file on the system or URL to a file on the internet. If the DTD is pointing to external path, it is called **external subset.**

- The **square brackets [ ]** enclose an optional list of entity declarations called **internal subset**.

Internal DTD

A DTD is referred to as an internal DTD if elements are declared within the XML files. To reference it as internal DTD, *standalone* attribute in XML declaration must be set to **yes**. This means the declaration works independent of external source.

Syntax

The syntax of internal DTD is as shown:

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

where *root-element* is the name of root element and *element-declarations* is where you declare the elements.

Example

Following is a simple example of internal DTD:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<!DOCTYPE address [
   <!ELEMENT address (name,company,phone)>
   <!ELEMENT name (#PCDATA)>
   <!ELEMENT company (#PCDATA)>
   <!ELEMENT phone (#PCDATA)>
]>
<address>
   <name>Tanmay Patil</name>
   <company>TutorialsPoint</company>
   <phone>(011) 123-4567</phone>
</address>
```

Nota Bene: **PCDATA** - **P**arsed **C**haracter **Data**.

> **CDATA** - (Unparsed) **C**haracter **Data**.The term CDATA is used about text data that should not be parsed by the XML parser. Characters like "<" and "&" are illegal in XML elements.

Let us go through the above code:

Start Declaration- Begin the XML declaration with following statement

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
```

DTD- Immediately after the XML header, the *document type declaration* follows, commonly referred to as the DOCTYPE:

```
<!DOCTYPE address [
```

The DOCTYPE declaration has an exclamation mark (!) at the start of the element name. The DOCTYPE informs the parser that a DTD is associated with this XML document.

DTD Body- The DOCTYPE declaration is followed by body of the DTD, where you declare elements, attributes, entities, and notations:

```
<!ELEMENT address (name,company,phone)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT company (#PCDATA)>
<!ELEMENT phone_no (#PCDATA)>
```

Several elements are declared here that make up the vocabulary of the <name> document. <!ELEMENT name (#PCDATA)> defines the element *name* to be of type "#PCDATA". Here **#PCDATA** means **parse-able text data**.

End Declaration - Finally, the declaration section of the DTD is closed using a closing bracket and a closing angle bracket (]>). This effectively ends the definition, and thereafter, the XML document follows immediately.

Rules

- The document type declaration must appear at the start of the document (preceded only by the XML header) - it is not permitted anywhere else within the document.
- Similar to the DOCTYPE declaration, the element declarations must start with an exclamation mark.
- The Name in the document type declaration must match the element type of the root element.

External DTD

In external DTD elements are declared outside the XML file. They are accessed by specifying the system attributes which may be either the legal *.dtd* file or a valid URL. To reference it as external DTD, *standalone* attribute in the XML declaration must be set as **no**. This means, declaration includes information from the external source.

Syntax

Following is the syntax for external DTD:

```
<!DOCTYPE root-element SYSTEM "file-name">
```

where *file-name* is the file with *.dtd* extension.

Example

The following example shows external DTD usage:

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<!DOCTYPE address SYSTEM "address.dtd">
<address>
  <name>Tanmay Patil</name>
  <company>TutorialsPoint</company>
  <phone>(011) 123-4567</phone>
</address>
```

The content of the DTD file **address.dtd** are as shown:

```
<!ELEMENT address (name,company,phone)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT company (#PCDATA)>
<!ELEMENT phone (#PCDATA)>
```

Types

You can refer to an external DTD by either using **system identifiers** or **public identifiers**.

System Identifiers

A system identifier enables you to specify the location of an external file containing DTD declarations. Syntax is as follows:

```
<!DOCTYPE name SYSTEM "address.dtd" [...]>
```

As you can see it contains keyword SYSTEM and a URI reference pointing to the location of the document.

Public Identifiers

Public identifiers provide a mechanism to locate DTD resources and are written as below:

```
<!DOCTYPE name PUBLIC "-//Beginning XML//DTD Address Example//EN">
```

As you can see, it begins with keyword PUBLIC, followed by a specialized identifier. Public identifiers are used to identify an entry in a catalog. Public identifiers can follow any format, however, a commonly used format is called *Formal Public Identifiers, or FPIs.*

DTD - Components

This chapter will discuss about XML Components from DTD perspective. A DTD will basically contain declarations of the following XML components:

- Element
- Attributes
- Entities

Elements

XML elements can be defined as building blocks of an XML document. Elements can behave as a container to hold text, elements, attributes, media objects or mix of all.

Each XML document contains one or more elements, the boundaries of which are either delimited by start-tags and end-tags, or empty elements.

Example

Below is a simple example of XML elements

```
<name>Tutorials Point</name>
```

As you can see we have defined a <name> tag. There's a text between start and end tag of <name>. Elements, when used in an XML-DTD, need to be declared which will be discussed in detail in the chapter DTD Elements.

Attributes

Attributes are part of the XML elements. An element can have any number of unique attributes. Attributes give more information about the XML element or more precisely it defines a property of the element. An XML attribute is always a *name-value* pair.

Example

Below is a simple example of XML attributes:

```
<img src="flower.jpg"/>
```

Here *img* is the element name whereas *src* is an attribute name and *flower.jpg* is a value given for the attribute *src*.

If attributes are used in an XML DTD then these need to be declared which will be discussed in detail in the chapter DTD Attributes

Entities

Entities are placeholders in XML. These can be declared in the document prolog or in a DTD. Entities can be primarily categorized as:

- Built-in entities
- Character entities
- General entities
- Parameter entities

There are five built-in entities that play in well-formed XML, they are:

- ampersand: &
- Single quote: '
- Greater than: >
- Less than: <
- Double quote: "

We will study more about entity declarations in XML DTD in detail in the chapter DTD Entities.

DTD - Elements

XML elements can be defined as building blocks of an XML document. Elements can behave as a container to hold text, elements, attributes, media objects or mix of all.

A DTD element is declared with an ELEMENT declaration. When an XML file is validated by DTD, parser initially checks for the root element and then the child elements are validated.

Syntax

All DTD element declarations have this general form:

```
<!ELEMENT elementname (content)>
```

- *ELEMENT* declaration is used to indicate the parser that you are about to define an element.
- *elementname* is the element name (also called the *generic identifier*) that you are defining.

- *content* defines what content (if any) can go within the element.

Element Content Types

Content of elements declaration in a DTD can be categorized as below:

- Empty content
- Element content
- Mixed content
- Any content

Empty Content

This is a special case of element declaration. This element declaration does not contain any content. These are declared with the keyword **EMPTY**.

Syntax

Following is the syntax for empty element declaration:

```
<!ELEMENT elementname EMPTY >
```

In the above syntax:

- **ELEMENT** is the element declaration of category *EMPTY*
- **elementname** is the name of empty element.

Example

Following is a simple example demonstrating empty element declaration:

```
<?xml version="1.0"?>
<!DOCTYPE hr[
   <!ELEMENT address EMPTY>
]>
<address />
```

In this example *address* is declared as an empty element. The markup for *address* element would appear as <address />.

Element Content

In element declaration with element content, the content would be allowable elements within parentheses. We can also include more than one element.

Syntax

Following is a syntax of element declaration with element content:

```
<!ELEMENT elementname (child1, child2...)>
```

- **ELEMENT** is the element declaration tag
- **elementname** is the name of the element.
- *child1, child2..* are the elements and each element must have its own definition within the DTD.

Example

Below example demonstrates a simple example for element declaration with element content:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<!DOCTYPE address [
   <!ELEMENT address (name,company,phone)>
   <!ELEMENT name (#PCDATA)>
   <!ELEMENT company (#PCDATA)>
   <!ELEMENT phone (#PCDATA)>
]>
<address>
   <name>Tanmay Patil</name>
   <company>TutorialsPoint</company>
   <phone>(011) 123-4567</phone>
</address>
```

In the above example, *address* is the parent element and *name*, *company* and *phone_no* are its child elements.

List of Operators and Syntax Rules

Below table shows the list of operators and syntax rules which can be applied in defining child elements:

Operator	Syntax	Description	Example
+	<!ELEMENT element-name (child1+)>	It indicates that child element can occur one or more times inside parent element.	<!ELEMENT address (name+)> Child element *name* can occur one or more

			times inside the element name *address*.
*	<!ELEMENT element-name (child1*)>	It indicates that child element can occur zero or more times inside parent element.	<!ELEMENT address (name*)> Child element *name* can occur zero or more times inside the element name *address*.
?	<!ELEMENT element-name (child1?)>	It indicates that child element can occur zero or one time inside parent element.	<!ELEMENT address (name?)> Child element *name* can occur zero or one time inside the element name *address*.
,	<!ELEMENT element-name (child1, child2)>	It gives sequence of child elements separated by comma which must be included in the the element-name.	<!ELEMENT address (name, company)> Sequence of child elements *name*, *company*, which must occur in the same order inside the element name *address*.
|	<!ELEMENT element-name (child1 | child2)>	It allows making choices in the child element.	<!ELEMENT address (name | company)> It allows you to choose either of child elements i.e. *name* or *company*, which must occur in inside the element name *address*.

Rules

We need to follow certain rules if there is more than one element content:

- **Sequences -** Often the elements within DTD documents must appear in a distinct order. If this is the case, you define the content using a sequence. For example:

```
<!ELEMENT address (name,company,phone)>
```

The declaration indicates that the <address> element must have exactly three children - <name>, <company>, and <phone> - and that they must appear in this order.

- **Choices**: Suppose you need to allow one element or another, but not both. In such cases you must use the pipe (|) character. The pipe functions as an exclusive OR. For example:

```
<!ELEMENT address (mobile | landline)>
```

Mixed Element Content

This is the combination of (#PCDATA) and children elements. PCDATA stands for parsed character data, that is, text that is not markup. Within mixed content models, text can appear by itself or it

can be interspersed between elements. The rules for mixed content models are similar to the element content as discussed in the previous section.

Syntax

Following is a generic syntax for mixed element content:

```
<!ELEMENT elementname (#PCDATA|child1|child2)*>
```

- **ELEMENT** is the element declaration tag.
- **elementname** is the name of the element.
- **PCDATA** is the text that is not markup. #PCDATA must come first in the mixed content declaration.
- *child1, child2..* are the elements and each element must have its own definition within the DTD.
- The operator (*) must follow the mixed content declaration if children elements are included
- The (#PCDATA) and children element declarations must be separated by the (|) operator.

Example

Following is a simple example demonstrating the mixed content element declaration in a DTD.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<!DOCTYPE address [
   <!ELEMENT address (#PCDATA|name)*>
   <!ELEMENT name (#PCDATA)>

]>
<address>
   Here's a bit of text mixed up with the child element.
   <name>Tanmay Patil</name>
</address>
```

ANY Element Content

You can declare an element using the ANY keyword in the content. It is most often referred to as mixed category element. ANY is useful when you have yet to decide the allowable contents of the element.

Syntax

Following is the syntax for declaring elements with ANY content:

```
<!ELEMENT elementname ANY>
```

Here, the ANY keyword indicates that text (PCDATA) and/or any elements declared within the DTD can be used within the content of the <elementname> element. They can be used in any order any number of times. However, the ANY keyword does not allow you to include elements that are not declared within the DTD.

Example

Following is a simple example demonstrating the element declaration with ANY content:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<!DOCTYPE address [
   <!ELEMENT address ANY>
]>
<address>
  Here's a bit of sample text
</address>
```

DTD - Attributes

In this chapter we will discuss about DTD Attributes. Attribute gives more information about an element or more precisely it defines a property of an element. An XML attribute is always in the form of a name-value pair. An element can have any number of unique attributes.

Attribute declaration is very much similar to element declarations in many ways except one; instead of declaring allowable content for elements, you declare a list of allowable attributes for each element. These lists are called ATTLIST declaration.

Syntax

Basic syntax of DTD attributes declaration is as follows:

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

In the above syntax

- The DTD attributes start with <!ATTLIST keyword if the element contains the attribute.
- **element-name** specifies the name of the element to which the attribute applies.

- **attribute-name** specifies the name of the attribute which is included with the element-name.
- **attribute-type** defines the type of attributes. We will discuss more on this in the following sections.
- **attribute-value** takes a fixed value that the attributes must define. We will discuss more on this in the following sections.

Example

Below is a simple example for attribute declaration in DTD:

```
<?xml version = "1.0"?>
<!DOCTYPE address [
<!ELEMENT address ( name )>
<!ELEMENT name ( #PCDATA )>
<!ATTLIST name id CDATA #REQUIRED>
 ]>
<address>
   <name id="123">Tanmay Patil</name>
</address>
```

Let us go through the above code:

- Begin with the XML declaration with the following statement:

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

- Immediately following the XML header is the document type declaration, commonly referred to as the DOCTYPE:

  ```
  <!DOCTYPE address [
  ```

  The DOCTYPE informs the parser that a DTD is associated with this XML document. The DOCTYPE declaration has an exclamation mark (!) at the start of the element name.

- Following is the body of DTD. Here we have declared element and attribute:

  ```
  <!ELEMENT address ( name )>
  <!ELEMENT name ( #PCDATA )>
  ```

- Attribute *id* for the element *name* is defined as:

```
<!ATTLIST name id CDATA #REQUIRED>
```

Here attribute type is *CDATA* and its value is *#REQUIRED*.

Rules of Attribute Declaration

- All attributes used in an XML document must be declared in the Document Type Definition (DTD) using an Attribute-List Declaration
- Attributes may only appear in start or empty tags.
- The keyword ATTLIST must be in upper case
- No duplicate attribute names will be allowed within the attribute list for a given element.

Attribute Types

When declaring attributes, you can specify how the processor should handle the data that appears in the value. We can categorize attribute types in three main categories:

- String type
- Tokenized types
- Enumerated types

Following table provides a summary of the different attribute types:

Type	Description
CDATA	CDATA is character data (text and not markup). It is a *String Attribute Type*.
ID	It is a unique identifier of the attribute. It should not appear more than once. It is a *Tokenized Attribute Type*.
IDREF	It is used to reference an ID of another element. It is used to establish connections between elements. It is a *Tokenized Attribute Type*.
IDREFS	It is used to reference multiple ID's. It is a *Tokenized Attribute Type*.
ENTITY	It represents an external entity in the document. It is a *Tokenized Attribute Type*.
ENTITIES	It represents a list of external entities in the document. It is a *Tokenized Attribute Type*.

NMTOKEN	It is similar to CDATA and the attribute value consists of a valid XML name. It is a *Tokenized Attribute Type*.
NMTOKENS	It is similar to CDATA and the attribute value consists a list of valid XML name. It is a *Tokenized Attribute Type*.
NOTATION	An element will be referenced to a notation declared in the DTD document. It is an *Enumerated Attribute Type*.
Enumeration	It allows defining a specific list of values where one of the values must match. It is an *Enumerated Attribute Type*.

Attribute Value Declaration

Within each attribute declaration, you must specify how the value will appear in the document. You can specify if an attribute:

- can have a default value
- can have a fixed value
- is required
- is implied

Default Values

It contains the default value. The values can be enclosed in single quotes(') or double quotes(")

Syntax

Following is the syntax of value:

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

where *default-value* is the attribute value defined.

Example

Following is a simple example of attribute declaration with default value:

```
<?xml version = "1.0"?>
<!DOCTYPE address [
<!ELEMENT address ( name )>
<!ELEMENT name ( #PCDATA )>
```

```
<!ATTLIST name id CDATA "0">
 ]>
<address>
   <name id="123">
      Tanmay Patil
   </name>
</address>
```

In this example we have *name* element with attribute *id* whose default value is *0*. The default value is been enclosed within the double quotes.

FIXED Values

#FIXED keyword followed by the fixed value is used when you want to specify that the attribute value is constant and cannot be changed. A common use of fixed attributes is specifying version numbers.

Syntax

Following is the syntax of fixed values:

```
<!ATTLIST element-name attribute-name attribute-type #FIXED "value" >
```

where #FIXED is an attribute value defined.

Example

Following is a simple example of attribute declaration with FIXED value:

```
<?xml version="1.0"?>
<!DOCTYPE address [
  <!ELEMENT address (company)*>
  <!ELEMENT company (#PCDATA)>
  <!ATTLIST company name NMTOKEN #FIXED "tutorialspoint">
]>
<address>
  <company name="tutorialspoint">we are a free online teaching faculty</company>
</address>
```

In this example we have used the keyword #FIXED where it indicates that the value "tutorialspoint" is the only value for the attribute *name* of element <company>. If we try to change the attribute value then it gives an error.

Following is an invalid DTD:

```
<?xml version="1.0"?>
<!DOCTYPE address [
  <!ELEMENT address (company)*>
  <!ELEMENT company (#PCDATA)>
  <!ATTLIST company name NMTOKEN #FIXED "tutorialspoint">
]>
<address>
  <company name="abc">we are a free online teaching faculty</company>
</address>
```

REQUIRED values

Whenever you want specify that an attribute is required, use #REQUIRED keyword.

Syntax

Following is the syntax of #REQUIRED:

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

where #REQUIRED is an attribute type defined.

Example

Following is a simple example of DTD attribute declaration with #REQUIRED keyword:

```
<?xml version = "1.0"?>
<!DOCTYPE address [
<!ELEMENT address ( name )>
<!ELEMENT name ( #PCDATA )>
<!ATTLIST name id CDATA #REQUIRED>
 ]>
<address>
   <name id="123">
      Tanmay Patil
   </name>
</address>
```

In this example we have used #REQUIRED keyword to specify that the attribute *id* must be provided for the element-name *name*

IMPLIED Values

When declaring attributes you must always specify a value declaration. If the attribute you are declaring has no default value, has no fixed value, and is not required, then you must declare that the attribute as *implied*. Keyword #IMPLIED is used to specify an attribute as *implied*.

Syntax

Following is the syntax of #IMPLIED:

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

where #IMPLIED is an attribute type defined.

Example

Following is a simple example of #IMPLIED

```
<?xml version = "1.0"?>
<!DOCTYPE address [
<!ELEMENT address ( name )>
<!ELEMENT name ( #PCDATA )>
<!ATTLIST name id CDATA #IMPLIED>
 ]>
<address>
   <name />
</address>
```

In this example we have used the keyword #IMPLIED as we do not want to specify any attributes to be included in element *name*. It is optional.

DTD - Entities

Entities are used to define shortcuts to special characters within the XML documents. Entities can be primarily of four types:

- Built-in entities
- Character entities
- General entities
- Parameter entities

Entity Declaration Syntax

In general, entities can be declared ***internally*** or ***externally***. Let us understand each of these and their syntax as follows:

Internal Entity

If an entity is declared within a DTD it is called as internal entity.

Syntax

Following is the syntax for internal entity declaration:

```
<!ENTITY entity_name "entity_value">
```

In the above syntax:

- **entity_name** is the name of entity followed by its value within the double quotes or single quote.
- **entity_value** holds the value for the entity name.
- The entity value of the Internal Entity is de-referenced by adding prefix **&** to the entity name *i.e. &entity_name.*

Example

Following is a simple example for internal entity declaration:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!DOCTYPE address [
<!ELEMENT address (#PCDATA)>
<!ENTITY name "Tanmay patil">
<!ENTITY company "TutorialsPoint">
<!ENTITY phone_no "(011) 123-4567">
]>
<address>
   &name;
   &company;
   &phone_no;
</address>
```

In the above example, the respective entity names *name*, *company* and *phone_no* are replaced by their values in the XML document. The entity values are de-referenced by adding prefix **&** to the entity name.

Save this file as **sample.xml** and open it in any browser, you will notice that the entity values for *name*, *company*, *phone_no* are replaced respectively.

External Entity

If an entity is declared outside a DTD it is called as external entity. You can refer to an external Entity by either using system identifiers or public identifiers.

Syntax

Following is the syntax for External Entity declaration:

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

In the above syntax:

- **name** is the name of entity.
- **SYSTEM** is the keyword.
- **URI/URL** is the address of the external source enclosed within the double or single quotes.

Types

You can refer to an external DTD by either using:

- **System Identifiers -** A system identifier enables you to specify the location of an external file containing DTD declarations. Syntax is as follows:

```
<!DOCTYPE name SYSTEM "address.dtd" [...]>
```

As you can see it contains keyword SYSTEM and a URI reference pointing to the document's location.

- **Public Identifiers -**

Public identifiers provide a mechanism to locate DTD resources and are written as below:

```
<!DOCTYPE name PUBLIC "-//Beginning XML//DTD Address Example//EN">
```

As you can see, it begins with keyword PUBLIC, followed by a specialized identifier. Public identifiers are used to identify an entry in a catalog. Public identifiers can follow any format; however, a commonly used format is called *Formal Public Identifiers, or FPIs.*

Example

Let us understand the external entity with the following example:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
```

```
<!DOCTYPE address SYSTEM "address.dtd">


<address>
   <name>Tanmay Patil</name>
   <company>TutorialsPoint</company>
   <phone>(011) 123-4567</phone>
</address>
```

Below is the content of the DTD file *address.dtd*:

```
<!ELEMENT address (name, company, phone)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT company (#PCDATA)>
<!ELEMENT phone (#PCDATA)>
```

Built-in entities

All XML parsers must support built-in entities. In general, you can use these entity references anywhere. You can also use normal text within the XML document, such as in element contents and attribute values.

There are five built-in entities that play their role in well-formed XML, they are:

- ampersand: &
- Single quote: '
- Greater than: >
- Less than: <
- Double quote: "

Example

Following example demonstrates the built-in entity declaration:

```
<?xml version="1.0"?>
<note>
   <description>I'm a technical writer & programmer</description>
<note>
```

As you can see here the & character is replaced by & whenever the processor encounters this.

Character entities

Character Entities are used to name some of the entities which are symbolic representation of information i.e characters that are difficult or impossible to type can be substituted by Character Entities.

Example

Following example demonstrates the character entity declaration:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!DOCTYPE author[
<!ELEMENT author (#PCDATA)>
<!ENTITY writer "Tanmay patil">
<!ENTITY copyright "&#169;">
]>
<author>&writer;&copyright;</author>
```

You will notice here we have used **©** as value for copyright character. Save this file as *sample.xml* and open it in your browser and you will see that copyright is replaced by the character ©.

General entities

General entities must be declared within the DTD before they can be used within an XML document. Instead of representing only a single character, general entities can represent characters, paragraphs, and even entire documents.

Syntax

To declare a general entity, use a declaration of this general form in your DTD:

```
<!ENTITY ename "text">
```

Example

Following example demonstrates the general entity declaration:

```
<?xml version="1.0"?>
<!DOCTYPE note [
<!ENTITY source-text "tutorialspoint">
]>

<note>
&source-text;
```

```
</note>
```

Whenever an XML parser encounters a reference to *source-text* entity, it will supply the replacement text to the application at the point of the reference.

Parameter entities

The purpose of a parameter entity is to enable you to create reusable sections of replacement text.

Syntax

Following is the syntax for parameter entity declaration:

```
<!ENTITY % ename "entity_value">
```

- *entity_value* is any character that is not an '&', '%' or ' " '.

Example

Following example demonstrates the parameter entity declaration. Suppose you have element declarations as below:

```
<!ELEMENT residence (name, street, pincode, city, phone)>
<!ELEMENT apartment (name, street, pincode, city, phone)>
<!ELEMENT office (name, street, pincode, city, phone)>
<!ELEMENT shop (name, street, pincode, city, phone)>
```

Now suppose you want to add additional eleement *country*, then then you need to add it to all four declarations. Hence we can go for a parameter entity reference. Now using parameter entity reference the above example will be :

```
<!ENTITY % area "name, street, pincode, city">
<!ENTITY % contact "phone">
```

Parameter entities are dereferenced in the same way as a general entity reference, only with a percent sign instead of an ampersand:

```
<!ELEMENT residence (%area;, %contact;)>
<!ELEMENT apartment (%area;, %contact;)>
<!ELEMENT office (%area;, %contact;)>
<!ELEMENT shop (%area;, %contact;)>
```

When the parser reads these declarations, it substitutes the entity's replacement text for the entity reference.

DTD - Validation

We use DTD to describe precisely the XML document. DTDs check the validity of structure and vocabulary of an XML document against the grammatical rules of the appropriate XML language. Now to check the validity of DTD, following procedures can be used:

- **Using XML DTD validation tools -** You can use some IDEs such as XML Spy (not free) and XMLStarlet(opensource) can be used to validate XML files against DTD document.
- **Using XML DTD on-line validators -** W3C Markup Validation Service is designed to validate Web documents. Use the online validator to check the validaty of your XML DTD here
- **Write your own XML validators with XML DTD validation API -** Newer versions of JDK (above 1.4) support XML DTD validation API. You can write your own validator code to check the validity of XML DTD validation.