

One of Many Books in the Need to Know Series from BrainMass

EVERYTHING YOU **NEED** TO KNOW ABOUT

XML

Document Type Definitions (DTDs)

**Ideal for
Undergrad
XML Students**

By Rao V.N.



www.brainmass.com

Everything You Need to Know About XML Document Type Definitions (DTDs)

By Rao V.N.

© BrainMass Inc. 2012

What Is This Book?

Everything You Need to Know publications are the best way to get a quick but detailed overview of a specific topic. Within the pages of this book, you'll find exactly what you need in order to understand the key concepts of this topic. You'll find yourself completely prepared for the next stage of your learning as it relates to this topic. Within every book, we include a collection of the most important terms and their definitions so that whenever you're in need of a refresher, you can easily refer back and remind yourself of what you're dealing with.

This book describes creation of Extensible Markup Language (XML) documents and Document Type Definitions (DTD) for the XML documents. This book provides an introduction to fundamental concepts related to creation of XML documents as well as step-by-step demonstration for creating document type definitions for XML documents.

This book is ideal for under-graduate students seeking an understanding of the way document type definitions are created for XML document. This book is also suitable for under-graduate students who are working with XML web applications and have a basic understanding of XML. Basic knowledge of XML is essential for understanding the topics covered in this book.

Table of Contents

Introduction.....	5
Everything You Need to Know	6
1. Introduction to Extensible Markup Language (XML)	7
2. Introduction to Document Type Definition.....	11
Glossary	24
References.....	26
Additional Resources	27
About the Author	28

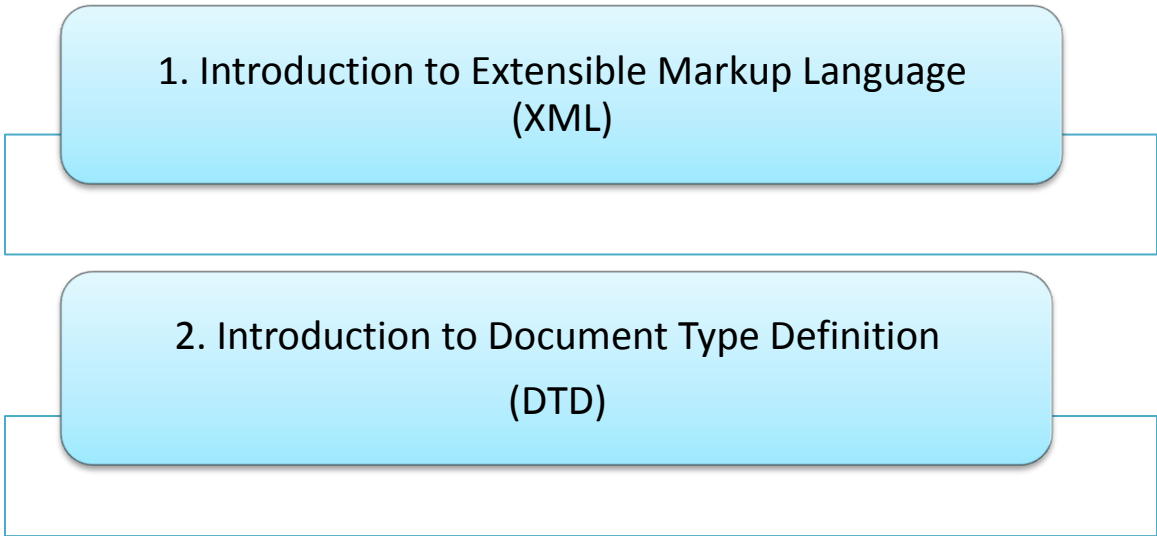
Introduction

XML is a flexible technology and an industry-wide standard used for describing information structure in documents. XML is not only emerging as an alternative to Electronic Document Interchange (EDI) but is also likely to play an important role with the emergence of the future generations of web technologies. With increasing popularity of electronic commerce, especially business-to-business e-commerce, the importance of XML will increase. To validate XML documents, we need either a document type definition or a schema. This book deals extensively with the concepts related to creation of a document type definition. Solved examples with a detailed step-by-step explanation presents readers with a clear understanding of the steps involved in creating XML documents with document type definitions (DTD). At the end of this book readers will be able to create and work with XML documents as well as document type definitions (DTDs), both internal as well as external.

Everything You Need to Know

This book includes two chapters titled Introduction to Extensible Markup Language and Introduction to Document Type Definition. The first chapter provides an introduction to XML with four subsections, each of which introduces basic concepts of XML. The first subsection in this chapter states the advantages of XML and the reasons for the popularity of XML. The second subsection provides a description of the structure of an XML document. The third and fourth subsections describe the criteria for checking the well-formedness and validity of XML files. At the end of first chapter, the reader should be able to create simple XML files and should be able to work with document type definitions (DTD) for XML files.

The second chapter begins with a description of the document type definition for XML documents. This chapter includes two subsections describing internal and external document type definitions. Each of these two subsections has two solved examples providing the readers with a practical demonstration of the theoretical concepts covered in each of the subsections. At the end of this chapter, the reader should be able to create internal and external document type definitions for XML documents.



1. Introduction to Extensible Markup Language
(XML)

2. Introduction to Document Type Definition
(DTD)

1. Introduction to Extensible Markup Language (XML)

XML stands for Extensible Markup Language. The main difference between the other popular markup language - Hypertext Markup Language (HTML) and XML is that HTML is used to present data in a particular format, whereas the key purpose of XML is to describe data. The first edition of Extensible Markup Language (XML) was accepted by W3C as a formal recommendation in 1998 and the second edition was accepted a formal recommendation in 2006 (McKinnon and McKinnon 2003).

1.1 Advantages of XML

XML has several advantages, some of which are listed below (Carey 2004; McKinnon and McKinnon 2003):

1. **XML is platform-independent.** The platform-independent nature of XML makes it popular format for users as they are not concerned with the platform while transferring information.
2. **XML is Unicode-based.** XML is Unicode-based rather than ASCII-based; hence XML can describe data from languages that can be represented using Unicode including all major languages such as English, Spanish, French, Italian, Mandarin, and Cantonese.
3. **XML is application-independent.** The application-independent nature of XML makes it easy to transfer XML files between applications developed using different languages. For instance, XML files can be transferred from an application developed using Java to another application developed using Visual Basic.NET.

1.2 Uses of XML

XML is gaining increasing popularity and prominence for several reasons including (Hunter et al., 2007; McKinnon and McKinnon 2003):

1. XML is platform-independent and language-independent making it an ideal format for exchanging data. For instance, XML could be used to reduce load on a webserver by keeping information on the client for a

period of time and sending information to the client in one single instance.

2. XML is now the preferred format for exchanging information in Business-to-Business (B2B) and Business-to-Commerce (B2C) e-commerce transactions.
3. Content Management Systems (CMS) allow website owners to update their website without knowledge of underlying operations or any web design/programming experience. XML is making great strides in integrating CMS solutions.
4. XML is increasingly becoming the language of choice to implement the middle tier of client/server interfaces due to the absence of any formatting instructions, making it ideal for data exchange.
5. XML is also increasingly used in databases due to its structured yet unformatted nature allowing manipulation by multiple database applications. XML is likely to be the universal language for representing data.

1.3 Structure of an XML document

An XML document consists of two key parts (Carey 2004):

- 1) Prolog
- 2) Body

The **prolog** is the first part of XML document consisting of information about the XML document. The first part in the prolog is an **XML declaration**. The syntax of XML declaration is given below:

```
<?xml version="number" encoding="encoding-type" standalone="yes/no"?>
```

Example of XML declaration:

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

The current XML version number is 1.1, the other value is the previous edition "1.0". Version 1.1 is not accepted yet by many browsers, so we will use version 1.0 in this book. The encoding attribute identifies the character code used in the XML document. In the above example the encoding format is the UTF-8 character set, which is the default character set for English language (McKinnon and McKinnon 2003). The last attribute is the standalone attribute, used to identify if the XML document is dependent on

any external files. In most cases the XML document is not dependent on external files, so the value is set to "yes".

Note: All the three attributes in the XML declaration are optional.

The prolog may also contain **comments** by the user. The purpose of comments is to improve the readability of the XML code and make it comprehensible. XML parser ignores the comments in an XML document. Comments in XML are quite similar to comments in HTML. Comments are stated between the <!-- and -->. An example is given below:

```
<!-- This is an example of a comment -->
```

The **body** of the XML typically consists of **elements** and **attributes**.

Elements are the basic units in XML documents containing data content. There are three types of elements: simple elements, complex elements, and empty elements. Simple elements are also referred to as closed elements; they contain only data content. For example, studentname is an element containing the name of the student and is represented as:

```
<studentname> Jack Martin </studentname>
```

Element names are case sensitive and must start with an alphabet or an underscore character.

Complex element is a collection of simple or complex elements. A complex element is usually a parent element that has child elements within it.

For instance: Address could be a complex element consisting of street number, street name, city name, and country name:

```
<address>
  <streetnumber>234 Bulford Avenue</streetnumber>
  <streetname>12th Street</streetname>
  <cityname>Toronto</cityname>
  <countryname>Canada</countryname>
</address>
```

Every XML document has a **root element**. There can be only one root element in an XML document. The root element encapsulates the other elements in the XML document.

XML documents may also contain empty elements or open elements. Single tags with a forward slash represent empty elements. For example, photo may be an empty tag in which case it is represented as <photo/> in the XML document.

Attributes are often used to describe the properties or characteristics of an element. For instance, in the case of student we can define student "idno" either as an element or an attribute. If we want to declare "idno" as an attribute then the syntax is:

```
<Student idno="123">  
....other element declarations are given here  
</Student>
```

An element can have any number of attributes. Single or double quotes enclose attribute values. Attribute names follow the same naming criteria as those of elements.

1.4 Well-Formed XML Documents

An XML document that has no syntax errors and meets all the specifications laid down by the formal recommendations accepted by W3C is considered a well-formed XML document (Carey 2004).

An XML parser is an application that interprets the code in an XML document and verifies if it meets the formal recommendation specifications laid down by W3C. Most web browsers including Microsoft Internet Explorer and Mozilla Firefox have integrated XML parsers. A parser can identify whether a document is well-formed or not. XML parsers are also referred to as XML processors.

1.5 Valid XML Documents

An XML document is valid if the XML document has either a Document Type Definition (DTD) or a Schema. The purpose of the Document Type Definition (DTD) or the Schema is to lay out rules for the XML document. In this book, we focus on Document Type Definition (DTD), and how to create valid XML documents using DTDs.

2. Introduction to Document Type Definition

A document type declaration (DTD) states the rules to be followed while defining the content and structure of the XML document. Because we can store any information in an XML file, it is necessary that we validate the content and structure of the data to maintain consistency and integrity of the data in the XML document.

2.1 Purpose of DTDs

The key purpose of a Document Type Definition (DTD) is to (Carey 2004):

- a) Ensure the presence of all relevant elements and attributes in the document.
- b) Ensure the presence of an appropriate structure in the document.
- c) Define default values for various elements and attributes in the document.

2.2 Advantages of DTDs

The key advantage of using DTDs is that DTDs can help validate the content in an XML document. The purpose of validating content in an XML document is to ensure that the receiving application can validate the structure of the XML document based on the guidelines stated in the DTD. The DTD ensures that the XML document follows the rules hitherto agreed upon by the sending and receiving applications. This adds credibility and consistency to the information exchange process between the two applications.

2.3 Types of DTDs

There are two types of document type definitions:

- 1) Internal Document Type Definitions
- 2) External Document Type Definitions

Internal Document Type Definitions are present within the XML file itself, while external Document Type Definitions are in a separate file with an extension ".dtd". These external ".dtd" files are linked to the XML document.

The key advantage of an external document type definition as compared to an internal document type definition is that one external document type definition can be used to validate multiple XML documents (McKinnon and McKinnon 2003).

2.3.1 Internal Document Type Definition

In the example given below, we will create a simple XML file and create an internal DTD for the XML file, and finally validate the XML file against the internal DTD.

First we need to create an XML document and check if it is well formed. While there are several commercial tools available including Altova XMLSpy, for this book we shall use the W3C XML validator available at the link given below to check for both well-formedness as well as validity.

http://www.w3schools.com/xml/xml_validator.asp

Solved Example 1:

Create an XML file "student.xml" and store information including the idno, first name, last name, gender, and gpa of the students. The XML file should contain at least two student elements. Check if the XML file is well-formed or not. Create an internal DTD to validate the XML file. Check if the XML file is valid or not.

Solution:

Step 1: First we shall create an XML file named "student.xml". While there are several XML editors available, we will use "notepad" to create this file as it is widely available.

Student.xml

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Students>
  <Student>
    <idno>123</idno>
    <firstname>Scott</firstname>
    <lastname>Smith</lastname>
    <gender>male</gender>
    <gpa>2.5</gpa>
  </Student>
```

```
<Student>
  <idno>124</idno>
  <firstname>Mary</firstname>
  <lastname>Cuthbert</lastname>
  <gpa>3.5</gpa>
</Student>
</Students>
```

Explanation:

The first line of code is the XML declaration. In this declaration we have stated that the version of XML used is 1.0. The encoding is UTF-8, which is default for English, and it is standalone because there are no external links.

Step 2:

We will check if the XML document is well-formed or not using the W3C XML validator. Open any browser and open the following URL:

http://www.w3schools.com/xml/xml_validator.asp

Go to the section titled "Syntax-Check Your XML", and place the XML code shown in step one in the text area of the section. Click on validate.

A message "No Errors Found" is displayed. If you are getting any errors, check if you have made any syntax errors while creating the XML file.

The XML document is well-formed.

Step 3:

As the XML file is well formed, we will check if the XML file is valid or not.

For this we need to design a DTD, which in this case is an internal DTD.

Modify the XML file created in step one by adding the internal DTD after the XML declaration (new code is italicized):

```
<?xml version="1.0" encoding="UTF" standalone="yes"?>
<!-- Internal Document Type Definition starts below-->
<!DOCTYPE Students [
  <!ELEMENT Students (Student)*>
  <!ELEMENT Student (idno,firstname,lastname,gender,gpa)>
  <!ELEMENT idno (#PCDATA)>
```

```
<!-- Internal document type definition ends here-->
<Students>
  <Student>
    <idno>123</idno>
    <firstname>Scott</firstname>
    <lastname>Smith</lastname>
    <gender>male</gender>
    <gpa>2.5</gpa>
  </Student>

  <Student>
    <idno>124</idno>
    <firstname>Mary</firstname>
    <lastname>Cuthbert</lastname>
    <gpa>3.5</gpa>
  </Student>
</Students>
```

Explanation:

The italicized code shows the comments as well as the document type definition. The comments indicate the start and end of the internal document type definition.

The **DOCTYPE declaration** is referred to as the document type declaration and is different from the document type definition. There can be only one DOCTYPE declaration in an XML file. In the case of internal document type definitions, the DOCTYPE has the following syntax (Carey, 2004):

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

In this example, the root element is "Students".

Next we need to define the name of the element, and if it a complex element. We also need to define the order of the elements within this

complex element. In this case Student is a complex element consisting of idno, firstname, lastname, gender, and gpa. So we have to define the order in which these elements appear in the XML document. If the order of elements in the XML document is different from the order stated in the document type definition, the document would not be validated.

After this the **content-model** of the element is stated. The content-model indicates the nature of the content stored in the element. Element content can be one of the following five types (Carey 2004; McKinnon and McKinnon 2003):

- a) ANY
- b) Character Data
- c) Elements
- d) Empty
- e) Mixed

The ANY content type means that there are no restrictions regarding the content of the element. The empty content type states that the content of the element must be empty; for instance, images or videos could use this content type. For elements that contain only string (collection of characters) type values, the element type is represented by the keyword (`#PCDATA`). In this example, because all the element values are strings, the content-model used is (`#PCDATA`).

The element content type indicates that the element is a complex element. For instance Student is a complex element, as it has other elements including idno, firstname, lastname, gender, and gpa within its content. The last content type "mixed" allows both character as well as element content in the element.

We have also used a **modifying symbol** "*" in the document type declaration. This modifying symbol stands for allowing zero or more items. This means that there can be a minimum of zero student elements and a maximum of any number in the "students" root element. There are two other modifying symbols: "?" indicates that there can be a minimum of zero and a maximum of one item within the element, and "+", which indicates that there can be a minimum of one item and a maximum of any number of items within an element (Carey 2004).

Step 4:

We will validate the XML file against the internal document type definition. Using the same W3C XML validator used in Step 2, go to the section titled "Validate Your XML Against a DTD" and paste the code shown in step three in the text area.

Click on Validate and a message titled "No Errors Found" will be displayed. If you get any error message check your DTD with the code given in Step 3.

In the first solved example we had used only entities. In the second example, we shall look at how attributes are represented in document type definitions.

Solved Example 2:

Create an XML file "products.xml" and store information including the product id, product name, product quantity, and price of the product. Product id is an attribute while the other fields are represented as entities in the XML file. The XML file should contain at least two product elements. Check if the XML file is well-formed or not. Create an internal DTD to validate the XML file. Check if the XML file is valid or not.

Step 1:

First we shall create an XML file named "Products.xml". While there are several XML editors available, we will use "notepad" to create this file as it is widely available.

Products.xml

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Products>
  <Product product_id="101">
    <product_name> Dell Inspiron</product_name>
    <product_quantity>100</product_quantity>
    <product_price>780.00</product_price>
  </Product>
  <Product product_id="102">
    <product_name>Asus 1250T</product_name>
    <product_quantity>53</product_quantity>
    <product_price>570.00</product_price>
  </Product>
</Products>
```

Explanation:

The first line of code is the XML declaration. In this declaration we have stated that the version of XML used is 1.0. The encoding is UTF-8, which is default for English and it is standalone because there are no external links.

Step 2:

Now, we will check if the XML document is well-formed or not using the W3C XML validator. Open any browser and open the following URL:

http://www.w3schools.com/xml/xml_validator.asp

Go to the section titled "Syntax-Check Your XML", and place the XML code shown in step one in the text area of the section. Click on validate.

A message "No Errors Found" is displayed. If you are getting any errors, check if you have made any syntax errors while creating the XML file.

The XML document is well-formed.

Step 3:

The XML file is well formed, we will check if the XML file is valid or not.

For this we need to design a DTD, which in this case is an internal DTD.

Modify the XML file created in Step 1 by adding the internal DTD after the XML declaration (new code is italicized):

```
<?xml version="1.0" encoding="UTF" standalone="yes"?>
<!-- Internal Document Type Definition starts below-->
<!DOCTYPE Products [
<!ELEMENT Products (Product)+>
<!ELEMENT Product
(product_id,product_name,product_quantity,product_price)>
<!ATTLIST Product product_id ID #REQUIRED>
<!ELEMENT product_name (#PCDATA)>
<!ELEMENT product_quantity (#PCDATA)>
<!ELEMENT product_price (#PCDATA)>
]>
<!-- Internal document type definition ends here-->

<Products>
  <Product product_id="101">
    <product_name> Dell Inspiron</product_name>
    <product_quantity>100</product_quantity>
    <product_price>780.00</product_price>
  </Product>
  <Product product_id="102">
    <product_name>Asus 1250T</product_name>
    <product_quantity>53</product_quantity>
    <product_price>570.00</product_price>
  </Product>
</Products>
```

Explanation:

The italicized code shows the comments as well as the document type definition. The comments indicate the start and end of the internal document type definition.

The **DOCTYPE declaration** is referred to as the document type declaration and is different from the document type definition. There can be only one DOCTYPE declaration in an XML file. In the case of internal document type definitions, the DOCTYPE has the following syntax (Carey, 2004):

```
<!DOCTYPE root-element  
[  
    declarations/statements  

```

In this example, the root element is "Products".

Following this we have the Product element, which has an attribute product_id. In the document type definition, we can accomplish the same through the use of ATTLIST. The ATTLIST keyword allows the user to mention the set of attributes.

The attribute value can be any of the three types, namely, string, enumerated, or tokenized types (Ray 2003). If the content of an attribute is a string (collection of one or more characters), we can use the keyword CDATA to indicate that the attribute value is a string.

For instance if product_id is a string, we could have declared:

```
<!ATTLIST Product product_id CDATA #REQUIRED>
```

The second type of attribute value is enumerated type. If an attribute value is limited to a certain number of attributes, we can use enumerated type to handle such attributes. For instance let us assume that "gender" is an attribute, the attribute declaration would be:

```
<!ATTLIST Student student_gender (male|female) #REQUIRED>
```

The last type of attribute values is a tokenized type. Tokenized types are text strings following a certain set of guidelines for format and content. For instance, if we want to declare product_id as a unique field, indicating that

product_id can have only unique values. The attribute list declaration in the document type definition would be:

```
<!ATTLIST Product product_id ID #REQUIRED>
```

We have used the above declaration for declaring the product_id attribute in the document type definition. The other tokens we can include in the tokenized type include IDREF, IDREFS, ENTITY, and ENTITIES.

The ATTLIST keyword is followed by the name of the element. The name of the element in this case is Product. This is followed by the name of the attribute, which in this case is product_id. This is followed by the Attribute Default. The Attribute Default can take of the three values, namely, #REQUIRED, #IMPLIED, or #FIXED "default" (Ray 2003). When the attribute default is specified as #REQUIRED, it means that the attribute must appear with every occurrence of the element. If the attribute default is specified as #IMPLIED, it means that specifying the attribute value is optional. In the case of #FIXED "default", it means that while the value of the attribute is optional, if the value is provided then it must be same as the default value mentioned (Carey 2004).

We have also used a **modifying symbol** "+" in the document type declaration. This modifying symbol stands for allowing a minimum of one item and a maximum of any number of items within an element (Carey 2004).

Step 4:

We will validate the XML file against the internal document type definition. Using the same W3C XML validator used in Step 2, go to the section titled "Validate Your XML Against a DTD" and paste the code shown in Step 3 in the text area.

Click on Validate and a message titled "No Errors Found" will be displayed. If you get any error message check your DTD with the code given in step three.

In the first solved example we had used only entities. In the second example, we shall look at how attributes are represented in document type definitions.

2.3.2 External Document Type Definition

An external document type definition is different from an internal document definition because the external document type can be applied to multiple XML documents, and resides in a separate external file with an extension ".dtd". We shall repeat solved example one, but this time around we will apply an external document type definition to validate the XML file.

Solved Example 3:

Create an external DTD to validate the XML file "students.xml" created in solved example 1. Check if the XML file is valid or not.

Solution:

Step 1:

The XML file is already created in the first example. Remove the internal document type definition embedded in the XML file and place it in a file named "students.dtd":

```
<!ELEMENT Students (Student)*>
<!ELEMENT Student (idno,firstname,lastname,gender,gpa)>
<!ELEMENT idno (#PCDATA)>
<!ELEMENT firstname (#PCDATA)>
<!ELEMENT lastname (#PCDATA)>
<!ELEMENT gender (#PCDATA)>
<!ELEMENT gpa (#PCDATA)>
```

Step 2:

Modify the DOCTYPE declaration in the XML file. The XML file would now appear as shown below:

```
<?xml version="1.0" encoding="UTF" standalone="yes"?>
<!-- External Document Type Definition starts below-->
<!DOCTYPE Students SYSTEM "students.dtd">
<!-- External document type definition ends here-->
<Students>
  <Student>
    <idno>123</idno>
    <firstname>Scott</firstname>
```

```
        <lastname>Smith</lastname>
        <gender>male</gender>
        <gpa>2.5</gpa>
    </Student>

    <Student>
        <idno>124</idno>
        <firstname>Mary</firstname>
        <lastname>Cuthbert</lastname>
        <gpa>3.5</gpa>
    </Student>
</Students>
```

Explanation:

The document type definition itself is not changed. The only change is in the DOCTYPE document type declaration. The DOCTYPE declaration is:

```
<!DOCTYPE Students SYSTEM "students.dtd">
```

The syntax for a DOCTYPE declaration for an external document type definition is the DOCTYPE keyword followed by the name of the root element, which in this case is "Students". The "SYSTEM" keyword follows indicating that the external document type definition is found in the same machine, followed by the path of the external DTD. In this case both the XML file and the DTD file are in the same folder, so the name of the DTD file is specified directly.

Solved Example 4:

Create an external DTD to validate the XML file "products.xml" created in solved example 2. Check if the XML file is valid or not.

Solution:**Step 1:**

The XML file is already created in the first example. Remove the internal document type definition embedded in the XML file and place it in a file named "products.dtd":

```
<!ELEMENT Products (Product)+>
<!ELEMENT Product
(product_id,product_name,product_quantity,product_price)>
<!ATTLIST Product product_id ID #REQUIRED>
<!ELEMENT product_name (#PCDATA)>
<!ELEMENT product_quantity (#PCDATA)>
<!ELEMENT product_price (#PCDATA)>
```

Step 2:

Modify the DOCTYPE declaration in the XML file. The XML file would appear as shown below:

```
<?xml version="1.0" encoding="UTF" standalone="yes"?>
<!-- External Document Type Definition starts below-->
<!DOCTYPE Products SYSTEM "products.dtd">
<!-- External document type definition ends here-->
<Products>
  <Product product_id="101">
    <product_name> Dell Inspiron</product_name>
    <product_quantity>100</product_quantity>
    <product_price>780.00</product_price>
  </Product>
  <Product product_id="102">
    <product_name>Asus 1250T</product_name>
    <product_quantity>53</product_quantity>
    <product_price>570.00</product_price>
  </Product>
</Products>
```

Glossary

Attributes - Attributes describe the characteristics or features of an element. For example, product element attributes could include the product id, product name, product price, etc.

Comments - Comments are used to improve the readability of code. In XML comments are declared within `<!-- -->` tags. XML comment style is the same as HTML comments.

Content Model – The content-model of an element indicates the nature of the content stored in an element. There are five different types of XML content-model.

Document Type Declaration - Document type declaration associates an XML file with an internal or external document type definition. The keyword DOCTYPE is used to indicate a document type declaration.

Document Type Definition - Document type definition or DTD states the guidelines to be followed in the definition of XML document content as well as structure. The document type definition can be either made externally or internally with the XML document.

Elements - Elements are used to represent the data content within an XML document. There are two types of elements, namely, closed and open elements.

Extensible Markup Language - Extensible Markup Language (XML) is a markup language designed to store and transport data in a platform-independent and application-independent manner.

Modifying Symbol - Modifying symbols are used in a document type definition to indicate the number of occurrences of an element in an XML document.

Root Element - also known as document element, root element encloses all the other elements in an XML document. There can be only one root element in an XML document.

Well-Formedness – An XML document is well-formed if the XML document meets the XML syntax requirements.

Validity – An XML document is valid if it is well-formed and has either a document type definition (DTD) or a schema.

References

Carey, P. 2004. New Perspectives on XML: Comprehensive. Boston: Course Technology.

McKinnon, A., McKinnon, L. 2003. XML: Web Warrior Series. Boston: Course Technology.

Ray, E.,T. 2003. Learning XML, New York: O'Reilly.

Additional Resources

Deitel, H.M., Deitel, P.J., Nieto, T.R., Lin, T., and Sadhu, P. 2000. XML How to Program. New Jersey: Pearson.

Ethier, K., and Houser, A. 2001. XML Weekend Crash Course. Indianapolis: Wiley.

Hunter, D., Rafter, J., Fawcett, J., Vlist, E.V., Ayers, D., Duckett, J., Watt, A., and McKinnon, L. 2007. Beginning XML. Indianapolis: Wiley.

Westermann, E. 2002. Learn XML in a Weekend. Ohio: Course Technology.

XML Tutorial. 2011. <http://www.w3schools.com/xml/> [Accessed 02 January 2012]

About the Author

Rao V.N. has completed his Masters in Computer Applications (MCA) from Osmania University in India as well as a Masters in Business Administration (MBA) from IUKB, Switzerland. He holds a certificate on Effective Online Tutoring from The Department of Continuing Education, Oxford University. He holds several professional certifications including: Oracle Certified Associate (OCA), Microsoft Certified Trainer (MCT), Sun Certified Java Professional (SCJP) and Cisco Certified Network Associate (CCNA). He has over 10 years of experience in teaching programming and database related courses at both the graduate and under-graduate level.