Introduction

In this chapter, we start to practice working with XML using XML documents, schemas, and stylesheets. An XML document organizes data and information in a structured, hierarchical format. An XML schema provides standards and rules for the structure of a given XML document. An XML schema also enables data transfer. An XSL (XML stylesheet) allows unique presentations of the material found within an XML document.

In the first chapter, *Introduction to XML*, you learned what XML is, why it is useful, and how it is used. So, now you want to create your very own XML documents. In this chapter, we will show you the basic components used to create an XML document. This chapter is the foundation for all subsequent chapters--it is a little lengthy, but don't be intimidated. We will take you through the fundamentals of XML documents.

This chapter is divided into three parts:

* XML Document
* XML Schema
* XML Stylesheets (XSL)

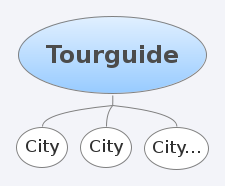
As you learned in the previous chapter, the XML Schema and Stylesheet are essentially specialized XML Documents. Within each of these three parts we will examine the layout and components required to create the document. There are links at the end of the XML document, schema, and stylesheet sections that show you how to create the documents using an XML editor. At the bottom of the page there is a link to Exercises for this chapter and a link to the Answers.

The first thing you will need before starting to create XML documents is a problem--something you want to solve by using XML to store and share data or information. You need some entity you can collect information about and then access in a variety of formats. So, we created one for you.

To develop an XML document and schema, start with a data model depicting the reality of the actual data that is exchanged. Once a high fidelity model has been created, the data model can be readily converted to an XML document and schema. In this chapter, we start with a very simple situation and in successive chapters extend the complexity to teach you more features of XML.

Our starting point is a single entity, CITY, which is shown in the following figure. While our focus is on this single entity, to map CITY to an XML schema, we need to have an entity that contains CITY. In this case, we have created TOURGUIDE. Think of a TOURGUIDE as containing many cities, and in this case TOURGUIDE has no attributes nor an identifier. It is just a container for data about cities.

**Exhibit 1: Data model - Tourguide**

[](https://en.wikibooks.org/wiki/File:Data_Model_-_Tourguide.png)

XML document

An XML document is a file containing XML code and syntax. XML documents have an .xml file extension.

We will examine the features & components of the XML document.

* Prologue (XML Declaration)
* Elements
* Attributes
* Rules to follow
* Well-formed & Valid XML documents

Below is a sample XML document using our TourGuide model. We will refer to it as we describe the parts of an XML document.

**Exhibit 2: XML document for city entity**

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

**<tourGuide** xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'

xsi:noNamespaceSchemaLocation='city.xsd'**>**

**<city>**

**<cityName>**Belmopan**</cityName>**

**<adminUnit>**Cayo**</adminUnit>**

**<country>**Belize**</country>**

**<population>**11100**</population>**

**<area>**5**</area>**

**<elevation>**130**</elevation>**

**<longitude>**88.44**</longitude>**

**<latitude>**17.27**</latitude>**

**<description>**Belmopan is the capital of Belize**</description>**

**<history>**Belmopan was established following the devastation of the

former capital, Belize City, by Hurricane Hattie in 1965. High

ground and open space influenced the choice and ground-breaking

began in 1966. By 1970 most government offices and operations had

already moved to the new location.

**</history>**

**</city>**

**<city>**

**<cityName>**Kuala Lumpur**</cityName>**

**<adminUnit>**Selangor**</adminUnit>**

**<country>**Malaysia**</country>**

**<population>**1448600**</population>**

**<area>**243**</area>**

**<elevation>**111**</elevation>**

**<longitude>**101.71**</longitude>**

**<latitude>**3.16**</latitude>**

**<description>**Kuala Lumpur is the capital of Malaysia and the largest

city in the nation**</description>**

**<history>**The city was founded in 1857 by Chinese tin miners and

preceded Klang. In 1880 the British government transferred their

headquarters from Klang to Kuala Lumpur, and in 1896 it became the

capital of Malaysia.

**</history>**

**</city>**

**<city>**

**<cityName>**Winnipeg**</cityName>**

**<adminUnit>**St. Boniface**</adminUnit>**

**<country>**Canada**</country>**

**<population>**618512**</population>**

**<area>**124**</area>**

**<elevation>**40**</elevation>**

**<longitude>**97.14**</longitude>**

**<latitude>**49.54**</latitude>**

**<description>**Winnipeg has two seasons. Winter and Construction.**</description>**

**<history>**The city was founded by people at the forks (Fort Garry)

trading in pelts with the Hudson Bay Company. Ironically,

The Bay was bought by America.

**</history>**

**</city>**

**</tourGuide>**

Prologue (XML declaration)

The XML document starts off with the prologue. The prologue informs both a reader and the computer of certain specifications that make the document XML compliant. The first line is the XML declaration (and the only line in this basic XML document).

**Exhibit 3: XML document - prologue**

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

**xml**   =   this is an XML document  
**version="1.0"**   =   the XML version (XML 1.0 is the W3C-recommended version)  
**encoding="UTF-8"**   =   the character encoding used in the document - UTF 8 corresponds to 8-bit encoded Unicode characters (i.e. the standard way to encode international documents) - [Unicode](http://www.unicode.org/) provides a unique number for every character.  
Another potential attribute of the XML declaration:  
**standalone="yes"**   =   the dependency of the document ('yes' indicates that the document does not require another document to complete content)

Elements

The majority of what you see in the XML document consists of XML **elements**. Elements are identified by their **tags** that **open** with < or </ and **close** with > or />. The **start tag** looks like this: <element attribute="value">, with a left angle bracket (<) followed by the **element type name**, optional **attributes**, and finally a right angle bracket (>). The **end tag** looks like this: </element>, similar to the start tag, but with a slash (/) between the left angle bracket and the element type name, and no attributes.

When there's nothing between a start tag and an end tag, XML allows you to combine them into an **empty element tag**, which can include everything a start tag can: <img src="Belize.gif" />. This one tag must be closed with a slash and right angle bracket (/>), so that it can be distinguished from a start tag.

The XML document is designed around a major theme, an umbrella concept covering all other items and subjects; this theme is analyzed to determine its component parts, creating categories and subcategories. The major theme and its component parts are described by elements. In our sample XML document, 'tourGuide' is the major theme; 'city' is a category; 'population' is a subcategory of 'city'; and the hierarchy may be carried even further: 'males' and 'females' could be subcategories of 'population'. Elements follow several rules of syntax that will be described in the Rules to Follow section.

We left out the attributes within the <tourGuide> start tag — that part will be explained in the XML Schema section.

**Exhibit 4: Elements of the city entity XML document**

**<tourGuide>**

**<city>**

**<cityName>**Belmopan**</cityName>**

**<adminUnit>**Cayo**</adminUnit>**

**<country>**Belize**</country>**

**<population>**11100**</population>**

**<area>**5**</area>**

**<elevation>**130**</elevation>**

**<longitude>**88.44**</longitude>**

**<latitude>**17.27**</latitude>**

**<description>**Belmopan is the capital of Belize**</description>**

**<history>**Belmopan was established following the devastation of the

former capital, Belize City, by Hurricane Hattie in 1965. High

ground and open space influenced the choice and ground-breaking

began in 1966. By 1970 most government offices and operations had

already moved to the new location.

**</history>**

**</city>**

**</tourGuide>**

**Element hierarchy**

* **root element**  -   This is the XML document's major theme element. Every document must have exactly one and only one root element. All other elements are contained within this one root element. The root element follows the XML declaration. In our example, <tourGuide> is the root element.
* **parent element**  -   This is any element that contains other elements, the child elements. In our example, <city> is a parent element.
* **child element**  -   This is any element that is contained within another element, the parent element. In our example, <population> is a child element of <city>.
* **sibling element**  -   These are elements that share the same parent element. In our example, <cityName>, <adminUnit>, <country>, <population>, <area>, <elevation>, <longitude>, <latitude>, <description>, and <history> are all sibling elements.

Attributes

Attributes aid in modifying the content of a given element by providing additional or required information. They are contained within the element's opening tag. In our sample XML document code we could have taken advantage of attributes to specify the unit of measure used to determine the area and the elevation (it could be feet, yards, meters, kilometers, etc.); in this case, we could have called the attribute 'measureUnit' and defined it within the opening tag of 'area' and 'elevation'.

<adminUnit **class="state"**>Cayo</adminUnit>

<adminUnit **class="region"**>Selangor</adminUnit>

The above attribute example can also be written as:

1. using child elements

**<adminUnit>**

**<class>**state**</class>**

**<name>**Cayo**</name>**

**</adminUnit>**

**<adminUnit>**

**<class>**region**</class>**

**<name>**Selangor**</name>**

**</adminUnit>**

2. using an empty element

<adminUnit class="state" name="Cayo" />

<adminUnit class="region" name="Selangor" />

Attributes can be used to:

* provide more information that is not defined in the data
* define a characteristic of the element (size, color, style)
* ensure the inclusion of information about an element in all instances

Attributes can, however, be a bit more difficult to manipulate and they have some constraints. Consider using a child element if you need more freedom.

Rules to follow

These rules are designed to aid the computer reading your XML document.

* The first line of an XML document must be the XML declaration (the prologue).
* The main theme of the XML document is established in the root element and all other elements must be contained within the opening and closing tags of this root element.
* Every element must have an opening tag and a closing tag - **no exceptions**

(e.g. <element>data stuff</element>).

* Tags must be nested in a particular order

=> the parent element's opening and closing tags must contain all of its child elements' tags; in this way, you close first the tag that was opened last:

<parentElement>

<childElement1>data</childElement1>

<childElement2>

<subChildElementA>data</subChildElementA>

<subChildElementB>data</subChildElementB>

</childElement2>

<childElement3>data</childElement3>

</parentElement>

* Attribute values should have quotation marks around them and no spaces.
* Empty tags or empty elements must have a space and a slash (/) at the end of the tag.
* Comments in the XML language begin with "<!--" and end with "-->".

**XML Element Naming Convention**

Any name can be used but the idea is to make names meaningful to those who might read the document.

* XML elements may only start with either a letter or an underscore character.
* The name must not start with the string "xml" which is reserved for the XML specification.
* The name may not contain spaces.
* The ":" should not be used in element names because it is reserved to be used for namespaces (This will be covered in more detail in a later chapter).
* The name may contain a mixture of letters, numbers, or other characters.

XML documents often have a corresponding database. The database will contain fields which correspond to elements in the XML document. A good practice is to use the naming rules of your database for the elements in the XML documents.

#### ONLINE Validator

<http://www.stg.brown.edu/service/xmlvalid/>

## Well-formed and valid XML

**Well-formed XML**  -  An XML document that correctly abides by the rules of XML syntax.

**Valid XML**  -  An XML document that adheres to the rules of an XML schema (which we will discuss shortly). To be valid an XML document must first be well-formed.

A Valid XML Document must be Well-formed. But, a Well-formed XML Document might not be valid - in other words, a well-formed XML document, that meets the criteria for XML syntax, might not meet the criteria for the XML schema, and will therefore be invalid.

For example, think of the situation where your XML document contains the following (for this schema):

<city>

<cityName>Boston</cityName>

<country>United States</country>

<adminUnit>Massachusetts</adminUnit>

:

:

:

</city>

Notice that the elements do not appear in the correct sequence according to the schema (cityName, adminUnit, country). The XML document can be validated (using validation software) against its declared schema – the validation software would then catch the out of sequence error.

### Using an XML Editor

Check chapter [XML Editor](https://en.wikibooks.org/wiki/XML_-_Managing_Data_Exchange/XML_Editor) for instructions on how to start an XML editor. Once you have followed the steps to get started you can copy the code in the sample XML document and paste it into the XML editor. Then check your results. Is the XML document well-formed? Is the XML document valid? (you will need to have copied and pasted the schema in order to validate - we will look at schemas next)

1. Museum information

* Create an XML document to describe a museum and populate it with data for two museums with at least 5 different dates in the calendar for one museum and 7 different dates for the other museum.

2. Currency codes for tourist guides

* Help the guides by creating a well-formed XML document describing currency codes (e.g. EUR for Euros). The currency code element should consist of three characters. Along that the country's name (e.g. Germany) and the name of the currency (e.g. Euros) should be included.

3. The spa finder

* The Spa Finder lists many spas, each spa serves many activities, and each activity is offered at many days and times by different practitioners.

\* the information found for each spa - NAME,OWNER, PHONE, CITY, STATE, ADDRESS, STARTED IN, TYPE OF SPA

\* the information found for each service - SERVICE, DESCRIPTION, PRICE

\* the information found for each offering of a service - DAYS, TIMES, PRACTITIONER, FLOOR, ROOM

* Extra- you could use enumeration for the SERVICE, listing some pre-determined services (ie. European Facial, Avocado Facial, Hot Stone Massage, Reflexology, Accupressure, Yoga, TaiChi).
* In the XML document - fill in the data for 3 spas with at least 3 services each and at least 2 offerings of each service.

4. Movie Database

* Create an XML document to depict a relationship between movie genres, movies, and actors/actresses.
* Choose two genres (drama, horror, action, etc.), two movies in each, and several actors/actresses from each.

5. Telephone directory

* A business has a telephone directory that records the first and last name, telephone number and email address of everyone working in the firm. Departments are the main organizing unit of the firm so the telephone directory is typically displayed in department order and shows for each department the contact phone and fax number and email address.

6. Credit card statement

Create a XML along with schema for a credit card statement

Write a DTD and an XML Schema for bank.xml (Added in the same folder) satisfying the following requirements:

* There are two account types: checking and savings accounts
* The account id is unique in 'accounts'
* The customer id is unique in 'customers'
* 'c\_id' refers to customers and 'ac\_id' refers to accounts
* The account balance must be greater than -5000
* Use inheritance for checking and savings accounts by deriving from a common account type