**L09.XML**

**Introduction**

Thus far in CWEB 180, we've seen tools for laying out text and images on a webpage through Hypertext Markup Language (HTML). We've also seen the ability to add styles to our text and images through Cascading Style Sheets (CSS). Nothing to date has dealt with the data itself or the structure of the data.

You need to know how to structure data such that it can be used with web pages and other applications. One way to do so is to use XML **(eXtensible Markup Language**). You need to know how to create and use XML documents to store and organize data.

Learning Steps

To meet this learning outcome, you will complete the following learning steps in order:

1. Discuss the purpose of XML documents.
2. Create an XML document.
3. Create a defined structure.
4. Apply styles with XML.

**1.The purpose of XML documents**

We have already seen that HTML is used to structure and lay out the contents of a web page. But if we just want to store or transfer data of any type, we can use **eXtensible Markup Language** (**XML**). XML looks similar to HTML in that it has elements surrounded by tags, but we can define the elements so that they suit our data, since XML is designed for data storage and transfer.

The fact that we can add or modify elements (tags) so that the tags suit our data is what is meant by the term "eXtensible" in the XML name. This feature allows developers to create their own tags rather than just using tags created by others. If you are in an industry that has already created XML structures, then you can use one of the created structures rather than creating your own.

Some of the rules that we'll use with XML are similar to the rules of HTML, The rules of XML include:

* Tags must be closed properly – both one-sided and two-sided tags.
* Elements are properly nested.
* Attributes are enclosed in quotes.

There is a single root element.

* The XML document has a root element and all other element tags are nested inside the root element properly.
* XML documents must be *well-formed* and can be *validated*.

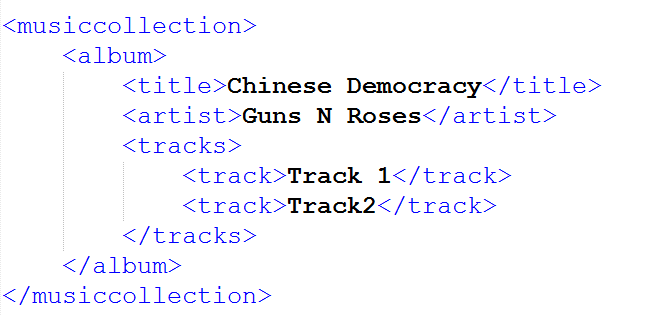
**2. Instruction: Creating an XML document**

An XML document will have an extension of **.xml** and will start off with this first line (the XML **prolog**):

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

This first line states the document will be using version 1.0 of XML along with the UTF-8 character set. Following this first line, we have the root element. The root element is a *required* element and will contain all other data elements.

In this example, the music collection consists of multiple albums that together form a complete album collection. Currently only a single album is displayed. If you want to add additional albums, insert the block of tags that correspond to the album and its properties. Only one root element is used.



The root element is <musiccollection> and is the parent element for all other elements. Every XML document must have a root element.

Within the root element, we have elements for each individual part of the data. In the example above, the album will have a title, an artist, and a series of tracks. Note that tracks contain individual track elements, so it is possible to have elements that have multiple values like tracks. The album could be any object that we need to describe. The elements within album are the individual properties that describe the album.

Note that In XML, the elements act as categories or containers for organizing and storing specific types of data. The actual data is placed inside these elements, between the opening and closing tags.Here actual data is Chinese Democracy.

This XML document will display in a browser and, at this point, doesn't have a defined structure. The document structure is defined with a document type definition (DTD). The DTD can be included in the XML file or in its own document.

The DTD is a collection of rules that defines the contents and structure of an XML file. It has the ability to define which elements are required and which are optional. In addition, the order of the defined structure must be followed.

**What are Attributes?**​

Attributes are extra details added to an XML element to give more information about it. They provide context but aren't part of the main data.​

**When to Use Attributes:**​

There’s no strict rule, but generally:​

* Use **attributes** for small details like IDs or labels that describe the element but aren’t the main data.​
* Use **elements** for the actual data content.​

**Example:**​

Imagine a library catalog:​

<book id="101">​  
<title>The Great Gatsby</title>​  
<author>F. Scott Fitzgerald</author>​  
</book>​

​  
id="101" is an **attribute** of the <book> element. It gives extra information (the book’s unique ID) but isn’t part of the title or author data.​

​

**2.1Exercise: Creating an XML document**

Suppose that you want to store information about students. In particular, you want to store the student's name, student number, and a list of the student's marks. Create an XML file that stores an example of this kind of data. Use at least one one-sided tag (with an attribute) and at least one two-sided tag.

**Exercise Answer: Creating an XML document**

**A student, Shane, has three courses. One course with the course code CWEB 180, another course with the course code COSC 190, and the third course with the course code COOS 181.**

**Write an XML document to represent this student's information, including their name, student number 123456, registered courses, and marks for each registered course. Marks for CWEB 180 is 90, COSC190 is 85 and COOS 181 are 53, respectively.**

One possible solution is as follows:

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

<student>

<name>Shane</name>

<studentNum num="123456" />

<marks>

<mark courseCode="CWEB 180">90</mark>

<mark courseCode="COSC 190">85</mark>

<mark courseCode="COOS 181">53</mark>

</marks>

</student>

Another possible solution is as follows:

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

<student>

<name>Shane</name>

<studentNum num="123456" />

<course>

<courseCode>CWEB 180</courseCode>

<mark>90</mark>

</course>

<course>

<courseCode>COSC 180</courseCode>

<mark>85</mark>

</course>

<course>

<courseCode>COOS 181</courseCode>

<mark>53</mark>

</course>

</student>

**3. Instruction: Creating a defined structure**

The rules that define the structure in an XML document are called the **Document Type Definition** or **DTD**. This declaration can be part of the XML file or can be located in its own file.

When the DTD is included in the XML file, it is located at the top of the document just underneath the XML prolog and before the XML elements.

**See the following example**:

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

<!DOCTYPE musiccollection [

<!ELEMENT musiccollection (album\*)>

<!ELEMENT album (title, artist, tracks)>

<!ELEMENT title (#PCDATA)>

<!ELEMENT artist (#PCDATA)>

<!ELEMENT tracks (track\*)>

<!ELEMENT track (#PCDATA)>

]>

<musiccollection>

<album>

<title>Chinese Democracy</title>

<artist>Guns N' Roses</artist>

<tracks>

<track>Welcome to the Jungle</track>

<track>Sweet Child O' Mine</track>

</tracks>

</album>

</musiccollection>

Looking at the DTD rules for this example, the syntax is first to define the root element (musiccollection) and then to define the elements contained within the root.

The DTD can contain elements, attributes, and entities. Only elements are used in this example.

**Instruction: Working with elements in the DTD**

Elements make up the tree structure of your data and have the following syntax:

**<!ELEMENT** elementName **content/type>**

There are several rules when working with elements including:

* elementName is the name of the element and is case-sensitive.
* Elements cannot contain spaces or < or > sign.
* The content specifies the type of content that the element can contain. The usual content is either parsed data or another element. In our previous example, the parsed data is the specific values that we want displayed (album name, artist, track names). Note that the tracks element does not contain parsed data but contains another element - track.

There are six possible values for content:

* ANY - no restrictions on content
* EMPTY - does not contain any content
* #PCDATA - contains only parsed data. Does not allow child elements. This is most commonly used.
* elements - contains other elements or child elements
* mixed - contains #PCDATA or elements

In the example above, the root element is called musiccollection. It will contain all of our data elements that pertain to our music collection.

From there, you define each of the individual elements of the music collection. The musiccollection will contain a group of albums. Each album will contain the title of the album, the artist, and a series of tracks. The tracks element will contain the individual tracks on the album. These are all elements (properties) that are used to define the characteristics of album.

Most elements will have a content type of either parsed data (#PCDATA) or another element. Parsed data refers to the actual data values that the parser processes and displays without further interpretation. In this example, it includes specific details like the album name, the artist, or the track names.

The next thing to note is the extra symbols. There is only one of the three extra symbols used in defining data in this example, which is the \*.

The three special symbols are:

* ? This symbol is used to designate that only zero or one occurrence of the element will appear.
* + This symbol is used to designate that one or more occurrences of the element will appear.
* \* This symbol is used to designate that zero or more occurrences of the element will appear.

For the above example, there are zero or more occurrences of an album existing in the musiccollection. The definition also states that there are zero or more occurrences of a track element occurring in the tracks element.

**Instruction: Working with external DTDs**

The document type definition (DTD) can also be in its own document. This document is called a DTD or document type definition. This separate document will have a **.dtd** extension.

The DTD file will have much the same content as the data definition in the XML document. In fact, you can copy the list of elements and descriptions from the top of the XML file and place it in its own document. Save it with a **.dtd** extension and include the name of the DTD file at the top of your XML file.

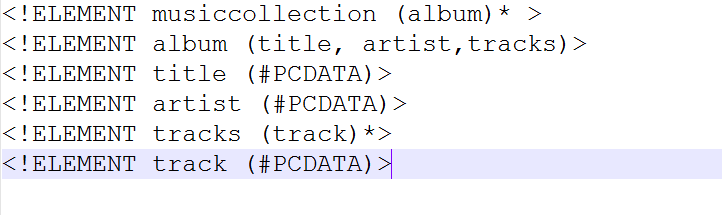
The XML document will need to include information to reference the DTD. Here is an example using the same content that we've previously used:

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

<!DOCTYPE musiccollection SYSTEM "music.dtd">

In this case, we've had to add an attribute called SYSTEM and a path to our external DTD. The SYSTEM attribute declares that we are using an external document that is on our system.

The DTD in our example is a separate file called **music.dtd** and here is its content:



The way to read this DTD file is as follows:

* Your root element is called musiccollection.
* The musiccollection contains one or more albums.
* The album contains the properties title, artist, and tracks, in that order.
* Title contains data.
* Artist contains data.
* Tracks contain another element called track.
* Track contains data.

**4.Instruction: Applying styles to XML**

Much like regular HTML, styles can be applied to an XML document to change the appearance of the data. There are several ways of doing this, including one that we've discussed earlier in the course.

Cascading style sheets can be used to format XML data. This is done in exactly that same format that we've used in HTML files.

Linking our stylesheet to the XML file is also slightly different. Here is the link to a style sheet called **music.css** that is located in a folder called **css**:

<?xml-stylesheet type="text/css" href="css/music.css" ?>

The syntax is as follows and appears just above the document type declaration in your XML file:

<?xml-stylesheet type="text/css" href="path to the CSS document you want to use to format your XML page" ?>

**Exercise**

A student, your name, has two courses, a course with course code CWEB 180, named “Web Site Development” and another course with course code CWEB 190, named “Internet Programming/Web Applications 1”. CWEB180 is registered, but CWEB190 is not.

Requirements:

1. Include the XML prolog at the appropriate location in your file. [1 mark]

2. Create elements and attributes to describe the situation. [3 marks]

3. You must have at least one one-sided element and at least one two-sided element. [2 marks]

4. Create an external DTD and attach it into your XML to verify the XML structure. [3 marks

5. Your file must contain well-formed XML. [2 marks]

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

<!DOCTYPE student SYSTEM "student.dtd">

<student>

<name>your name</name>

<course>

<courseNumber>CWEB 180</courseNumber>

<courseName>Web Site Development</courseName>

<registered status="yes" />

</course>

<course>

<courseNumber>CWEB 190</courseNumber>

<courseName>Internet Programming/Web Applications 1</courseName>

<registered status="no" />

</course>

</student>

<!ELEMENT student (name,course+)>

<!ELEMENT name (#PCDATA)>

<!ELEMENT course (courseNumber,courseName,registered)>

<!ELEMENT courseNumber (#PCDATA)>

<!ELEMENT courseName (#PCDATA)>

<!ELEMENT registered EMPTY>

<!ATTLIST registered status CDATA #REQUIRED>

**Applying styles to XML**

<?xml version="1.0" encoding="UTF-8"?>  
<?xml-stylesheet type="text/css" href="music.css" ?>  
<music>  
 <album>  
 <title>Thriller</title>  
 <artist>Michael Jackson</artist>  
 <tracks>  
 <track>Wanna Be Startin' Somethin'</track>  
 <track>Thriller</track>  
 <track>Beat It</track>  
 </tracks>  
 </album>  
</music>

**CSS**

album {  
 font-family: Arial, sans-serif;  
 background-color: #f4f4f4;  
 padding: 10px;  
 margin: 10px;  
 border: 1px solid #ccc;  
}  
  
title {  
 font-size: 20px;  
 color: blue;  
 font-weight: bold;  
}  
  
artist {  
 font-style: italic;  
 color: green;  
}  
 track {  
 margin-left: 20px;  
 list-style: square;  
}