Electronic Commerce Architecture Project
LAB ONE: Introduction to XML by Lawrence Leff

An XML document has two required parts. The first is the definition of what data should be in the document. The second is the document itself. An optional third part shows how the data is to be displayed on the browser (or printed).

Defining the contents of an XML document is usually done using a notation called DTD. When XML was first invented, that was used. However, there is another way that this can be done. That is XML Schemas, a newer standard from the World Wide Web Consortium. When we switch from using DTDs to Schemas, the first line of the document changes. The remainder of the document remains the same, regardless of which definition method is used.

The third part is optional. It defines how the data is to be viewed on the browser. Cascading Style sheets (CSS) or XML Style Sheet (XSL). The first is simpler. The second is more recent, complicated and is less supported by software. For example, Netscape supports Cascading Style Sheets but does not support XML Style Sheets. Internet Explorer Five supports both CSS and XSL. See the Excellent Legal XML Note on CSS and XSL Style sheets by Mr. Chambers.

The definition and style sheets are prepared by those defining the type of document. For example, e-procurement companies such as Intelisys developed standards for purchase orders and items. Or trade groups such as the Legal XML Group or the IMS project prepared standards for their respective industries (court filing and distance education).

Then, "everyone" will prepare XML documents that adhere to the standard. In many cases, this preparation will be done by various automated tools. For example, a person ordering products via an e-commerce business system would not write the XML for the purchase orders with a text editor. Their purchasing system would do this for them and forward it to a receiving computer, perhaps running a different brand of purchasing software.

However, raw XML is very easy to understand with a little bit of training or by reading tutorials such as this one. I believe that the proverbial "secretary," with a little training, could learn to read and prepare raw XML files, given the will to learn.

Using DTDs to define a XML Document

All XML documents always begin with the following text. Treat it as "boiler plate."

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

An XML document using DTD's can include the definition of the data type directly in the file or reference it from an external location.

However, as standards are put in production use, the data definition will have to be referenced separately. The DTD would be on one web page. Any XML document that confirms to that standard would reference it.

An XML document consists of a series of start tags and corresponding end tags. An example is:

```
<tableofcontents>
<item>Item One Information</item>
<item>Item Two Information</item>
<item>Item Three Information</item>
</tableofcontents>
```

We say that the **item** is inside the **tableofcontents** tag. We also say that each </item> matches the corresponding <item>. These must be correctly nested. We must close an element inside something before we close the outer element. In our example, each ITEM must end before the end tag for **tableofcontents** comes.

Thus

```
<tableofcontents>
<item>Item One Information </tableofcontents> More information
```

after the Item One `</item>`
would be illegal.

Each item might have information about it. For example, each item might have an **identifier** and a **title**.

Obviously we can just put the information in the text part of the tag. Say something like this:

```
<tableofcontents>
<item>Identifier=item1,title="Pigs"</item>
<item>Identifier=item2,title="Giraffes"</item>
<item>Identifier=item3,title="Elephants"</item>
</tableofcontents>
```

However, XML provides two more structured ways of associating such information. This allows the program reading the received XML to easily pull the information out using a software utility called an XML parser. (In later labs, we will learn about two types of XML parsers.)

We could create tags for **Identifier** and **title**. These would appear between <item> and </item>.
This would lead to XML that looks as follows:

```
<tableofcontents>
<item><Identifier>item1</Identifier><title>Pigs</title></item>
<item><Identifier>item2</Identifier><title>Giraffes</title></item>
<item><Identifier>item3</Identifier><title>Elephants</title></item>
</tableofcontents>
```

Or we could make **Identifier** and **title** attributes which means that the XML would look as follows:

```
<tableofcontents>
<item Identifier='item1' title='Pigs'></item>
<item Identifier='item2' title='Giraffes'/>
<item Identifier='item3' title='Elephants'/>
</tableofcontents>
```

Note that the **item** tags don't have any information in the tag. There is nothing between <item> and </item>. We call this an empty element. One can simply have nothing between the tag and its end as we

did in the item1-Pigs line above. Or one can use a short cut of putting a slash just before the closing right bracket as done in the other two lines containing **item**s.

There is no technical reason to use subelements instead of attributes or vice versa. Both are equally easy for a program to parse. It is simply a personal aesthetic value judgment.

As mentioned earlier, these XML fragments need some boiler plate at the front. In particular, this boiler plate needs to point the parser to the DTD describing the information.

The general form for an XML is the following: **<?xml version="1.0" encoding='UTF-8'?>**
<!DOCTYPE *rootname* **SYSTEM** *DTD-url***>**
<rootname...
interior tags
</rootname>

where *rootname* must be the name of the opening tag. *DTD-url* gives the file location of the URL--this can be a full name or the name of a file in the same directory as the XML.

This is exemplified by complete XML file for the first example:

```
<?xml version="1.0" encoding='UTF-8'?>
<!DOCTYPE tableofcontents SYSTEM "lab1.dtd">
<tableofcontents>
<item>Item One Information</item>
<item>Item Two Information</item>
<item>Item Three Information</item>
</tableofcontents>
```

The DTD would contain a definition of the form:
<!ELEMENT tag-name (stuff-definition)>
stuff-definition says which elements can be inside the element or if the element should contain text.

In the simplest case, *stuff-definition* contains a list of the elements that can appear between <*tag-name*> and </*tag-name*>. (Note that the root element also must be defined with an **ELEMENT** clause.) Thus, if we had the line:

```
<!ELEMENT tableofcontents (item)>
```

That would mean that the table of contents could have only one **item**.

When the definer of the DTD wishes to give the user options, choices, and the ability to elements to appear several times, there is some special syntax to be used. (I use the phrase "user" to mean the person or program preparing the XML document.)

- To indicate that the user can choose one of several tags, the DTD has a vertical bar (|).
- To indicate that a tag can occur one or more times, use a plus sign (+) in the DTD.
- When a tag an occur as many times as the user wants, or even not occur at all, the DTD has a star (*****).
- And lastly, when a tag can occur zero or one times but not more than once, the DTD has a question mark (**?**).

We can also group several items with parentheses.

Bottom level tags can be declared with **#PCDATA**, **EMPTY** or **ANY**. The first means that the tag contains only text, but no subtags. **Any** means that the tag can contain any subtags. **EMPTY** means that the tag contains no subtags, not even a space or carriage return. Thus, we write:
<!ELEMENT item (#PCDATA)>

When, the XML contains a tag with an attribute of the form
```
<tag-name attribute-name='value' />
```

We need to declare each attribute in the DTD as follows:
```
<!ATTLIST tag-name value CDATA #IMPLIED>
```

The "implied" means that the person preparing the document may omit the attribute. Use #REQUIRED in the DTD if you want to ensure that the tag will always have that attribute.

Laboratory procedure

You will find your validation work easiest if one puts your XML into a home page directory that is readable by the world. It will allow you to easily use validators such as those provided by: the Brown Scholarly Technology Group, World Wide WEB Consortium XML Schema Validation software, and the Microsoft validation page.

The first lab procedure will have you set up some simple XML files and use the validation procedures.

To do this on the Architecture account, do these steps:

1. (You can also keep the materials on your Athena account and use UNIX editors and file manipulation.)
2. Log into your Architecture Account from the School of Architecture and Planning computers.
3. Using NotePad (under Accessories), create a file **INDEX.HTM** in the **PUBLIC_HTML** directory. This directory will be in your Z drive.
   Make it contain:
   ```
   <Title<>
   <>Hello</P>
   ```

   Enter the following URL:
   http://architecture.mit.edu/~*user-id*/ where *user-id* is your architecture server login. This will be the same as your userid.

   For example, my home page is:
   http://architecture.mit.edu/leff
   since my Athena user id is *leff*.

In general, if you create a file called *xxxx*.html in the **public_html** subdirectory of your account, you should be able to access it as
http://architecture.mit.edu/*~user-id*/*xxxx*.html

4. *Now, we practice with DTDs* Create a file called **lab1.xml** in your **public_html** directory containing the following:

```
<?xml version="1.0" encoding='UTF-8'?>
<!DOCTYPE tableofcontents SYSTEM "lab1.dtd">
<tableofcontents>
<item>Item One Information</item>
<item>Item Two Information</item>
<item>Item Three Information</item>
</tableofcontents>
```

5. Create a file called **lab1.dtd** containing the following:

```
<?xml version="1.0" encoding="UTF-8" ?>
<!ELEMENT tableofcontents (item*)>
<!ELEMENT item (#PCDATA)>
```

6. Start up Explorer and go to the following URL (Bookmark it as you will be using it often.)
   http://www.stg.brown.edu/service/xmlvalid/
7. In the URI section of the form, enter the URL of the lab file that you just created:
   http://architecture.mit.edu/*~user-id*/**lab1.xml**
   Click on **Validate**.
8. You should get the message "Document validates OK"
   Now change the lab1.xml file so the third "item" and its end tag have initial caps. That line should now look like:
   `<Item>Item Three Information</Item>`
9. Click on the back button in the browser and click on Validate again.
   Observe what happens.
   Restore "item" to its proper capitalization.
10. *Now, we will experiment with subitems.*
    Change **lab1.xml** so it looks as follows:

```
<?xml version="1.0" encoding='UTF-8'?>
<!DOCTYPE tableofcontents SYSTEM "lab1.dtd">
<tableofcontents>
<item><Identifier>item1</Identifier><title>Pigs</title></item>
<item><Identifier>item2</Identifier><title>Giraffes</title></item>
<item><Identifier>item3</Identifier><title>Elephants</title></item>
</tableofcontents>
```

and change **lab1.dtd** so it looks like

```
<?xml version="1.0" encoding="UTF-8" ?>
<!ELEMENT tableofcontents (item*)>
<!ELEMENT item (Identifier,title)>
<!ELEMENT Identifier (#PCDATA)>
<!ELEMENT title (#PCDATA)>
```

11. Go again to http://www.stg.brown.edu/service/xmlvalid/
12. In the URI section of the form, you still have the URL of the lab file that you just created:
    http://architecture.mit.edu/*~user-id*/**lab1.xml**
    Click on **Validate**.
13. You should get the message "Document validates OK"
14. Now, change the second item line so it has two title's about giraffe's. Let it look like this:
    `<item><Identifier>item2</Identifier><title>Giraffes</title><title>More About Giraffe's</title></item>`
15. Validate your model again. What happens?
    Try fixing your DTD so an item can have one or more title's. Check to ensure that the DTD enforces the rule that each item have at least one title.
16. Edit **lab1.xml** so it reads as follows:

```
<?xml version="1.0" encoding='UTF-8'?>
<!DOCTYPE tableofcontents SYSTEM "lab1.dtd">
<tableofcontents>
<item Identifier='item1' title='Pigs'></item>
<item Identifier='item2' title='Giraffes'/>
<item Identifier='item3' title='Elephants'/>
</tableofcontents>
```

17. Edit the lab1 DTD file so it looks below:

```
<?xml version="1.0" encoding="UTF-8" ?>
<!ELEMENT tableofcontents (item*)>
<!ELEMENT item EMPTY>
<!ATTLIST item Identifier CDATA #IMPLIED>
<!ATTLIST item title CDATA #IMPLIED>
```

18. Validate your file. You should get a warning about more than one attlist for the item but no errors. You can ignore this warning.
19. Change the "title" attribute name so it is incorrect. For example, make it look as follows:
    `<item Identifier='item2' Title='Giraffes'/>`
20. Revalidate your xml--observe the two errors.
21. *Now, we will prepare some XML according to the IMS Global Learning Consortium Metadata Specification. First, we will look at the DTD and sample file from the IMS Project Home page.*
    Open up the following URL in your browser:
    http://www.imsproject.org
22. Select **Specifications** from the menu just below their banner title. Then select **Metadata**.
23. Now create a file **simpleims.xml** in your home page directory that looks as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE record SYSTEM "http://www.imsproject.org/xml/IMS_MDCORv1p1.dtd">
<record>
<general>
<title>
<langstring lang="en">Introduction to XML</langstring>
</title>
<catalogentry>
<catalogue>http:</catalogue>
<entry><langstring lang="en">web.mit.edu/leff/www/lab1.html</langstring></entry>
</catalogentry>
</general>
<lifecycle>
```

```
<contribute>
<role>
<langstring lang="en">Author</langstring>
</role>
<centity><vcard>BEGIN:vCardFN:Laurence L. LeffN:LeffEND:vCard</vcard></centity>
</contribute>
</lifecycle>
</record>
```

24. Validate it with the Brown University validator
25. Now, read the DTD at and then edit the XML to add Ariadne Goal and Dan Greenwood as Contributors.
26. Revalidate your XML:
    http://www.imsproject.org/xml/IMS_MDCORv1p1.dtd
27. Reading the DTD for the Rights tag, add XML that shows this lab is copyright MIT, year 2000, but there is no cost for the item. You should be able to do this WITHOUT looking at the example in ims.org
28. Revalidate your XML.

Bibliography

Graham, Ian S., and Liam Quin, *XML Specification Guide.* ISBN 0471327530. John Wiley & Sons, 1999.

Homer, Alex. “XML.” *IE5 Programmer's Reference*. ISBN 1861001576. Wrox Press, 1999.

Laurence Leff, "Legal XML. Unofficial Note: A Tutorial on XML in the Court Filing Context.” (Number UN_100XX_2000__02_28.htm)
http://www.wiu.edu/users/mflll/UN_100XX_2000__02_28.htm. In Submission.

www.xml.com

Extensible Markup Language (XML) 1.0. www.w3.org/TR/REC-xml. W3C Recommendation.