Week 5 Lecture: DTDs - Part 1

Screencast of Recorded Lecture (2022):

- https://sjeccd-edu.zoom.us/rec/share/5kPAGv-HCZvGanSVrhe-B_5XHC1cYpe4fUgsMhKjRSvGYiy7Ft0QOCxrZfcybAVD.UIGJ6riqeVSr6_JU?startTime=1665101092000 (https://sjeccd-edu.zoom.us/rec/share/5kPAGv-HCZvGanSVrhe-B_5XHC1cYpe4fUgsMhKjRSvGYiy7Ft0QOCxrZfcybAVD.UIGJ6riqeVSr6_JU?startTime=1665101092000)

Screencast of Recorded Lecture (2021):

- https://sjeccd-edu.zoom.us/rec/play/C0uzp7q-m7FVj9AgpE8RoKNlUjArxJr_nB4Z-knw3-Mppxhjg9XfT92U3dk9yk-rLyVzty5CbAasrE4-.GSr2nXv9BvStgng5 (https://sjeccd-edu.zoom.us/rec/play/C0uzp7q-m7FVj9AgpE8RoKNlUjArxJr_nB4Z-knw3-Mppxhjg9XfT92U3dk9yk-rLyVzty5CbAasrE4-.GSr2nXv9BvStgng5)

Both this week and next week I will be talking about Document Type Definitions (DTD). I'm going to post both weeks at once, but take several days to digest the material. This is important stuff, but it can be really tricky to fully understand.

First, let's talk about why we need a DTD in the first place. What exactly is a DTD? I like to think of a DTD as a "rulebook" describing the rules that this particular XML file is going to follow.

When documents are shared among business partners it is useful for XML documents to conform to a known, predictable structure. You could, of course, write some custom code in a programming language like PHP or Python to check to see if the structure of a document matches the required parameters. However, it is far more convenient to check structure using the validation tools that form part of a validating XML processor.

In XML, a Document Type Definition (DTD) defines the allowed structure of a class of XML documents. In other words, a DTD declares the rules that are going to be in play on this XML document. A validating XML processor can use the DTD to confirm that a document conforms to the relevant DTD. Using XML-based validation cuts down on the need to write custom code.

There are two types of XML processors. A non-validating XML processor checks that documents conform to the syntax rules of XML, but doesn't check for any specific structure of elements or attributes. A validating XML processor checks that the document is well formed and also checks that the document conforms to a defined structure.

An example of a non-validating XML processor would be your web browser. *Firefox* or *Internet Explorer* or *Chrome* -- they will check to see if the code is valid, but they won't check to see if you're following the rules you've established for the XML file.

The purpose of a Document Type Definition is to define the legal building blocks of an XML document. It's the rulebook for the XML -- defining what and how this XML file is supposed to be used.

The DTD defines the document structure with a list of legal elements. A DTD can be declared ***inline*** in an XML document or as an ***external*** reference. Generally, it is preferred to use an external DTD, especially if you are going to be using the same set of rules for several (or many!) XML files.

That said, let's start with how to declare an internal DTD:

Internal DTDs

Internal Document Type Definition example:

If the DTD is included in the XML source file (internal), it should be wrapped in a DOCTYPE definition with the following syntax:

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

Look at this example:

```
<?xml version="1.0"?>
<!DOCTYPE   note  [
       <!ELEMENT  note  (to, from, heading, body)>
       <!ELEMENT  to    (#PCDATA)>
       <!ELEMENT  from  (#PCDATA)>
       <!ELEMENT  heading  (#PCDATA)>
       <!ELEMENT  body     (#PCDATA)>
]>
<note>
       <to>Joe</to>
       <from>Tom</from>
       <heading>Reminder</heading>
       <body>Don't forget your book</body>
</note>
```

The DTD above is interpreted like this:

!DOCTYPE note (in line 2) defines that this is a document of the type **note.**
!ELEMENT note (in line 3) defines the **note** element as having four elements: "to, from, heading, body".
!ELEMENT to (in line 4) defines the **to** element to be of the type "#PCDATA".
!ELEMENT from (in line 5) defines the **from** element to be of the type "#PCDATA".
!ELEMENT heading (in line 6) defines the **heading** element to be of the type "#PCDATA".
!ELEMENT body (in line 7) defines the **body** element to be of the type "#PCDATA".

PCDATA means parsed character data. You can think of character data as the text found between the start tag and the end tag of an XML element.

External DTDs

If the DTD is external to the XML source file, it should be wrapped in a DOCTYPE definition with the following syntax:

```
   <!DOCTYPE  root-element   SYSTEM  "name of the DTD"   "filename">
```

or

```
   <!DOCTYPE  root-element   PUBLIC  "name of the DTD"   "filename">
```

Where **"name of the DTD"** follows the following rules:

```
   "prefix//owner_of_the_DTD//description_of_the_DTD//ISO 639_language_identifier"
```

And where **"filename"** is either a relative or absolute URL.

There are 2 types of External DTDs:

- private DTD (indicated by the word **SYSTEM**)

        - Private DTDs are designed to only be applicable locally -- that is, to a set of XML files that the general public likely won't have access to.
        - SYSTEM DTDs are the most commonly used type of Doctype.

    - public DTD (indicated by the word **PUBLIC**)
        - Public DTDs are designed to be accessed globally -- that is, they are a set of commonly used rules that everyone who wants to build this type of XML file ought to use.

        - Examples of a DOCTYPE declaration using PUBLIC DTDs:
            - XHTML:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN" "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
```

            - In this Doctype:
                - `html` is the root
                - `PUBLIC` is the type of DTD
                - `-` is the prefix
                - `W3C` is the owner
                - `DTD XHTML 1.1` is the description
                - `EN` (English) is the human language
                - `http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd` is the URL
                  Note that this is a fully-qualified (absolute) URL.

            - Bills produced by US Congress (e.g., https://www.congress.gov/116/bills/hr4630/BILLS-116hr4630ih.xml (https://www.congress.gov/116/bills/hr4630/BILLS-116hr4630ih.xml)):

```
<!DOCTYPE bill PUBLIC "-//US Congress//DTDs/bill.dtd//EN" "bill.dtd">
```

            - In this Doctype:
                - `bill` is the root
                - `PUBLIC` is the type of DTD
                - `-` is the prefix
                - `US Congress` is the owner
                - `DTDs/bill.dtd` is the description
                - `EN` (English) is the human language
                - `bill.dtd` is the URL
                  Note that "bill.dtd" is a relative URL.
                  In this case, it means that the DTD file is located in the same directory that the XML file is located.

OK, now look at this example:

```
<?xml version="1.0"?>
<!DOCTYPE note  SYSTEM "-//me//a note//EN" "note.dtd">
<note>
   <to>Joe</to>
   <from>Tom</from>
   <heading>Reminder</heading>
   <body>Don't  forget your book</body>
</note>
```

And this is what the file "note.dtd" containing the DTD would look like.

```
<!ELEMENT   note     (to, from, heading,  body)>
<!ELEMENT   to       (#PCDATA)>
<!ELEMENT   from     (#PCDATA)>
<!ELEMENT   heading  (#PCDATA)>
<!ELEMENT   body     (#PCDATA)>
```

Again, in an XHTML document, you may have seen the following DTD:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN" "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
```

In this case, **html** is the root element; it is a public DTD; it's owned by the W3C; it's description is "DTD XHTML 1.1"; it's using the human language English, and the location of the DTD is on the w3.org's website.

So why use a Document Type Definition (DTD)?

- With a DTD, each XML file can carry a description of its own format with it.
- With a DTD, independent groups of people can agree to use a common DTD for interchanging data.
- Applications can use a standard DTD to verify that the data you receive from the outside world is valid.
- You can use a DTD to verify your own XML data.

Now let's look at the XML building blocks, as seen from a DTD.

The main building blocks of XML documents are elements (tags).

In the example, one such is:

```
<note> ... </note>
```

Seen from a DTD point of view, all XML documents are composed of the following building blocks:

- **Elements**
- **Tags**
- **Attributes**
- **Entities**
- **PCDATA**
- **CDATA**

Let's look briefly at each of these building blocks.

Elements

Elements are the main building blocks of XML documents. In our example, the elements would be "note", "heading", "to", "from", and "body". Elements can contain text, other elements, or be empty.

Tags

Tags are used to markup elements. A starting tag like **<element_name>** marks up the beginning of an element, and an ending tag like **</element_name>** marks up the end of an element.

In this example, **<to>** is the starting tag for the element **to**, **</to>** is the ending tag for the element **to**, and **Joe** is the content inside.

```
<to>Joe</to>
```

Attributes

Attributes provide extra information about elements. Attributes are always placed inside the starting tag of an element. Attributes are always defined by the name, and must contain a value.

A tag with an attribute will have this syntax:

```
<tag attribute="value"> ... </tag>
```

The following "p" element has additional information about the paragraph:

```
  <p style="text-align: center;"> paragraph that will be centered </p>
```

The name of the element is "p". The name of the attribute is "style". The value of the attribute is "text-align: center;"

In this example, the "img" element has additional information about a source file:

```
    <img src="computer.gif" />
```

The name of the element is "img". The name of the attribute is "src". The value of the attribute is "computer.gif". Since the element itself is empty, it is closed by a "/".

Entities

Entities are variables used to define common text. Entity references are references to entities. The following are entities predefined in XML:

Entities

Entity Reference	Character
&lt;	<
&gt;	>
&amp;	&
&quot;	"
&apos;	'

PCDATA

PCDATA means **P**arsed **C**haracter **DATA**. You can think of character data as the text found between the start tag and the end tag of an XML document. PCDATA is text that will be parsed by a parser. **Tags inside the text will be treated as markup and entities will be expanded**.

CDATA

CDATA is **C**haracter **DATA** that will *not* be parsed by a parser. **Tags inside the text will not be treated as markup and entities will not be expanded**.

CDATA content is declared using this to begin the CDATA:

```
<![CDATA[
```

and this to end it:

```
]>
```

Just to be clear: consider the <strong> tag from HTML. When a <strong> tag is parsed, by default, it will make the information inside the <strong> tag appear bold.

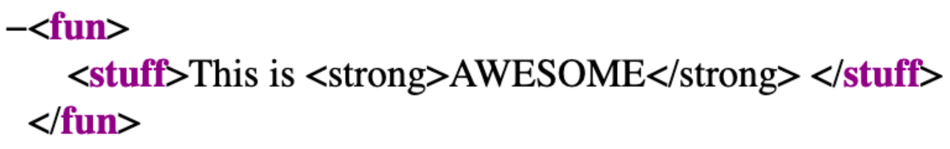
If we declare the element "strong" using PCDATA, then anything inside <strong> will render (process) as bold.

BUT, if we declare the element "strong" using CDATA, it will be treated as the text <strong>, and not only will the content *not* be bolded, the content will include the <strong> code. Be careful using CDATA; make sure you don't want the content to be parsed.

Here's an example illustrating this CDATA vs. #PCDATA issue; in this XML file, the <strong>AWESOME</strong> content would NOT be parsed with the "strong" element:

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE fun
[
<!ELEMENT fun (stuff)>
<!ELEMENT stuff (#PCDATA)>
]>
<fun>
<stuff>This is <![CDATA[ <strong>AWESOME</strong> ]]> </stuff>
</fun>
```

How it's parsed:



```
–<fun>
   <stuff>This is <strong>AWESOME</strong> </stuff>
 </fun>
```

Note that while the "fun" and the "stuff" elements are bolded and purple, the "strong" element isn't parsed at all, and is acting just like regular text.

OK, now let's take a closer look at these building blocks:

Elements

In a DTD, XML elements are declared with a DTD element declaration.

Declaring an Element

An element declaration has the following syntax:

```
<!ELEMENT   element-name  category>
```

or

```
<!ELEMENT   element-name   (element-content)>
```

Empty elements

Empty elements are declared with the category keyword EMPTY, as follows:
<!ELEMENT element-name EMPTY>

DTD example:

```
<!ELEMENT  br  EMPTY>
```

XML example:

```
<br />
```

Elements with only character data

Elements with only character data are declared with #PCDATA inside parentheses, as shown:
<!ELEMENT element-name (#PCDATA)>

DTD example:

```
<!ELEMENT  from   (#PCDATA)>
```

XML example:

```
<from>Tom</from>
```

Elements with any contents

Elements declared with the category keyword ANY can contain any combination of data that can be parsed, as shown:
<!ELEMENT element-name ANY>

DTD example:

```
<!ELEMENT  foo   ANY>
```

XML example #1:

```
<foo>Holy cow! This is great</foo>
```

XML example #2 (assuming "bold" and "underline" are also declared and defined):

```
<foo> <bold>Holy cow!</bold> <underline>This is great</underline></foo>
```

Elements with children (sequences)

Elements with one or more children are defined with the name of the children elements inside parentheses, as shown:

```
<!ELEMENT   element-name
   (child-element-name)>
```

or

```
<!ELEMENT   element-name
   (child-element-name,child-element-name,…)>
```

DTD example:

```
<!ELEMENT  starwars   (name,episode,year)>
```

XML example:

```
<starwars>
    <name>Star Wars: A New Hope</name>
    <episode>4</episode>
    <year>1977</year>
</starwars>
```

When children are declared in a sequence separated by commas, the children must appear in the same sequence in the document. In a full declaration, the children must also be declared, and the children can also have children. The full declaration of the "note" element in our example at the beginning of the lecture would be:

```
<!ELEMENT   note     (to, from, heading,  body)>
<!ELEMENT   to       (#PCDATA)>
<!ELEMENT   from     (#PCDATA)>
<!ELEMENT   heading  (#PCDATA)>
<!ELEMENT   body     (#PCDATA)>
```

Declaring only one occurrence of the same element

If you want to only allow one occurrence of an element, use the code **<!ELEMENT element-name (child-name)>**

Example:

```
<!ELEMENT  note   (message)>
```

The example declaration above declares that the child element "**message**" can only occur one time inside the "**note**" element.

Declaring minimum one occurrence of the same

element.

Use the + sign to declare that the child element must occur one or more times inside the parent element:
<!ELEMENT element-name (child-name+)>

Example:

```
<!ELEMENT  note   (message+)>
```

The example declaration above declares that the child element "**message**" must occur one or more times inside the "**note**" element.

Declaring zero or more occurrences of the same element.

Use the * sign to indicate that the child element can occur zero or more times inside the parent element:
<!ELEMENT element-name (child-name*)>

Example:

```
<!ELEMENT  note   (message*)>
```

The * sign in the above example declares that the child element "**message**" can occur zero or more times inside the "**note**" element.

Declaring zero or one occurrences of the same element

Use the ? sign to indicate that the child element can occur either zero times or just once inside the parent element:
<!ELEMENT element-name (child-name?)>

Example:

```
<!ELEMENT  note   (message?)>
```

The ? in the above example declares that the child element "**message**" can occur zero or one times inside the "**note**" element.

Declaring either/or content

The | sign (pipe sign -- found on most US keyboards by holding shift and pressing the key between the backspace & enter keys) indicates that the child element can be one of the listed elements (separated by the pipe):
<!ELEMENT element-name (child1|child2|child3)>

The example below declares that the "**note**" element must contain a "**to**" element, a "**from**" element, a "**header**" element, and either a "**message**" or a "**body**" element.

```
<!ELEMENT  note (to, from, header, (message|body))>
```

Declaring mixed content

The example below declares that the "note" element can contain parsed character data and any number of "to", "from", "header", and/or "message" elements>

Example:

```
<!ELEMENT  note   (#PCDATA|to|from|header|message)*>
```

Let's walk through a slightly more complex example:

"Star Wars" DTD Example

Suppose I have the following as my XML file:

```
<?xml version="1.0" encoding="utf-8"?>
<starwars>
  <movie>
    <title>Star Wars: A New Hope</title>
    <episode>4</episode>
```

```
     <year>1977</year>
     <director>
        <firstname>George</firstname>
        <lastname>Lucas</lastname>
     </director>
   </movie>
   <movie>
      <title>Empire Strikes Back</title>
      <episode>5</episode>
      <year>1980</year>
      <director>
         <firstname>Irvin</firstname>
         <lastname>Kershner</lastname>
      </director>
   </movie>
   <movie>
      <title>Return of the Jedi</title>
      <episode>6</episode>
      <year>1983</year>
      <director>
        <firstname>Richard</firstname>
        <lastname>Marquand</lastname>
      </director>
   </movie>
 </starwars>
```

In this case, we have the following elements:

- starwars
  - movie
    - title
    - episode
    - year
    - director
      - firstname
      - lastname

So in the XML, the elements that need to be declared are **starwars**, **movie**, **title**, **episode**, **year**, **director**, **firstname**, and **lastname**.

The order in which they are declared is important. Also, since there are multiple instances of the movie element, we need to indicate that as well.

Let's go through a potential DTD.

Since we're talking about **internal** DTDs, we would put the doctype in the XML file:

```
<!DOCTYPE starwars [
```

Next, we want to declare the elements in the XML, and the order in which they are to be listed:

```
<!ELEMENT starwars (movie*)>
```

This indicates that the **starwars** element has only one child element -- **movie** -- and that there are potentially more than one instance of the **movie** element (use the ***** sign to indicate that the child element can occur zero or more times inside the parent element; use the **+** sign to declare that the child element must occur one or more times inside the parent element; use the **?** sign to indicate that the child element can occur either zero times or just once inside the parent element).

```
<!ELEMENT movie (title, episode, year, director)>
```

This indicates that the **title**, **episode**, **year**, and **director** elements are sub-elements of the **movie** element.

```
<!ELEMENT title (#PCDATA)>
<!ELEMENT episode (#PCDATA)>
<!ELEMENT year (#PCDATA)>
```

These indicate that the **title**, **episode**, and **year** elements contain parsed character content -- that is, content that we want the browser/parser to parse.

```
<!ELEMENT director (firstname,lastname)>
```

This indicates that the **director** element has two child elements -- **firstname** and **lastname**.

```
<!ELEMENT firstname (#PCDATA)>
<!ELEMENT lastname (#PCDATA)>
```

These indicate that the **firstname** and **lastname** elements contain parsed character content -- that is, content that we want the browser/parser to parse.

So the entire DTD might look like:

```
<!ELEMENT starwars (movie*)>
<!ELEMENT movie (title, episode, year, director)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT episode (#PCDATA)>
<!ELEMENT year (#PCDATA)>
<!ELEMENT director (firstname,lastname)>
<!ELEMENT firstname (#PCDATA)>
<!ELEMENT lastname (#PCDATA)>
```

Full Example of Internal DTD with XML:

If it's an internal DTD, the entire XML code might look like:

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE starwars [
<!ELEMENT starwars (movie*)>
<!ELEMENT movie (title, episode, year, director)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT episode (#PCDATA)>
<!ELEMENT year (#PCDATA)>
<!ELEMENT director (firstname,lastname)>
<!ELEMENT firstname (#PCDATA)>
<!ELEMENT lastname (#PCDATA)>
]>

<starwars>
 <movie>
 <title>Star Wars: A New Hope</title>
 <episode>4</episode>
 <year>1977</year>
 <director>
 <firstname>George</firstname>
 <lastname>Lucas</lastname>
 </director>
 </movie>
 <movie>
 <title>Empire Strikes Back</title>
 <episode>5</episode>
 <year>1980</year>
 <director>
 <firstname>Irvin</firstname>
 <lastname>Kershner</lastname>
 </director>
 </movie>
 <movie>
 <title>Return of the Jedi</title>
 <episode>6</episode>
 <year>1983</year>
 <director>
 <firstname>Richard</firstname>
 <lastname>Marquand</lastname>
 </director>
 </movie>
</starwars>
```

I think that's enough about DTDs for the moment...