**Introduction to XML Schema**

**Unit 5**

**XML Schema Basics** is actually *XML*

* 1. Schema is a recent standard released in 2001
  2. XML Schema offers a lot more functionality than DTDs
  3. As we saw with DTDs, while in wide use, they lack a lot of functionality and the standard is old.
  4. Part of the problem with DTDs was that they were not XML. DTDs were based on a standard that pre-dated the advent of XML.
  5. XML Schema *is* XML.
  6. Schema gives us more control over our XML documents
  7. Because Schema is XML, it will eventually replace DTDs. It’s extensible!
  8. There are many communities that still rely on DTD. For example, the archives community use a markup standard called Encoded Archival Description. The markup standard was created when only DTD’s were available in the early 90s. It since has created an XML Schema once the standard became available in 2001. Currently, the profession allows archives to use either a DTD or XML Schema to validate their XML files
  9. This week we are going to cover most of chapters 9 & 10. You will learn just enough of chapter 11 in order to get you started on your exercise.

1. XML Schema can be thought of either part of a **Simple type** or **Complex Type**.
   1. **Simple types** are types that describe the text of your xml document.
   2. **Complex types** describe the structure of your XML document.
   3. Basically, Simple types contain text, Complex types contain attributes and children
   4. *children* and *attributes*—we will talk about these briefly just to get you going on your exercise, but we will not talk about Complex Types more fully next week.
   5. In summary, define what is allowable in your XML documents by either using a **Simple Type** or a **Complex Type.**
   6. You declare the element name in your Simple Type definition
   7. Next, you must define what type of text content is allowable in the element
   8. **A Simple Type Elements**  are the building blocks of our xml documents. Schema defines them using XML language. Simple types may only contain content, and may not contain child elements.
      * + 1. Example: <element name="employee"

**Data Types,** in a simple type you must specify what kind of data will be contained in the content. Whereas with DTDs, we merely just put #PCDATA, which could basically be anything.

* 1. With each Element you define, you must define what type of content is allowable.
  2. You must define the *data type*
  3. We have had a little experience doing this already, by defining DTD content to PCDATA.
     + - 1. Example: type="xs:string"/>
  4. We will talk more about data types shortly

1. **The Schema Basics or how to Set up your XML file.** Your .xml document must declare what version of XML you are using as well as its namespace. You must also define the root element in the namespace definition.
   1. **Namespace** the schema requires that you use the Schema namespace. The Namspace defines all of the syntax and functionality allowed in the Schema standard.
   2. by entering in the **“xmlns:xs=**[**http://www.3.org/ 2001/XMLSchema**](http://www.3.org/%202001/XMLSchema) you are telling the XML parser that this is XML, and that it uses a namespace called XML Schema.
   3. We will talk about Namespaces later in the course, but it’s important to know that this is how standards are updated, and how you know which version of the standard you are using. In this case, we are using 2001.
   4. When you use the XML Schema standard, you define elements by starting with the namespace **prefix xs:.** This tells the parser that you are using reserved set of command as designated by the XMLSchema namespace.
   5. When you use the definition command **<xs:element** the parser knows that element is used in a ways already specified in the XML namespace as decided on by the internet standards body, the W3C.
   6. Then, you will tell the parser this is a *complexType* and then that it is a *Sequence*, and then begin your element and data type definitions.
   7. Each element, in fact every new line of Schema code, will begin with **xs:**
   8. **Connecting the Schema file with the XML file**
   9. In a similar way that we link our Document Type Definition at the top of you XML file, you will do the same with your XML file that is linked to your Schema file.
   10. You will need to type the *root element* of your XML document, and then enter the namespace for Schema standard
   11. Next, you type xsi:noNamespaceSchemaLocation, which tells that parser that your Schema is not on the web, but locally held on your computer.
   12. Next, you will tell the parser what Schema you are using to *validate* your document by entering the file name in quotes.
   13. Remember, your root element will always be opened in the namespace declaration and closed in the very last closing tag of the XML document. We will look at setting up your .xsd file at the end of the presentation.
2. **What are Data Types?**
   1. XML Schema provides a lot of what are known as built-in **data types.** These allow you to determine what type of text is permissible in the content of your elements. With DTDs we were very limited, but now we have all of the same functionality that a database management system or general programming language affords.
   2. Every element must have its *data type* defined
   3. *Data Types* allows you to define and limit your content
   4. Once you defined your element name, you can define the data type:
      * 1. **Example: <element name="name" type="xs:string"/>**
      1. **xs:string**
         1. Type xs: to tell the parser that you are using the namespace for the Schema 2001 standard that defines how this data type should be used.
         2. Next, type the *data type* that you want to impose on the data for that element.
            1. In the first one we are limiting our data to string data, which is the most general of all the data types. It basically can contain any character.
      2. **xs:integer**
         1. You can also define your content to only allow numeric data. This basically includes any numeric data that is not fractional, for example no decimals are allowable.
      3. **xs:date**
         1. You can limit your data to dates. The date must be entered in the XML document in a prescribe manner. It must follow this *YYYY-MM-DD*.
      4. **xs:time**
         1. You can limit your data to a *time*. This follow the formula of, hh-mm-ss.
3. **Predefining an Element’s Content, using Default and Fixed**
   1. There are 2 ways you can predefine an element’s content by using a ***fixed*** *value* or a ***default*** *value*
   2. With a ***fixed*** value, you set the element equal to a certain value.
      1. The value must be equal to what *fixed* is set equal to unless it is left blank in the XML document, then it is considered valid
      2. If an element value is left empty, the document is still valid, but if it has a different value entered it is invalid!
   3. With a ***default*** value, you set the element equal to a certain value.
      1. If the value is equal to, omitted, or another value is used, they are all considered valid. The main point of this functionality is to have a default value that will always be there just in case nothing is entered in the XML document.
4. **Deriving a Custom Simple Type**
   1. XML Schema allows you to use foundational data types to derive your own customized data types.
   2. The syntax for this usually using **restriction** and **base**, followed by another customized data entry. The use of restrictions are also know as ***facets***, where you are able to use multiple restrictions on the element you are defining. Think of searching for books under the subject “Whaling”, and limiting to the facet “Fiction”
   3. You might think of the same process of doing a faceted search in the libraries **OPAC**. You might search by author, and then limit those results by a certain date or format.
   4. For example, I can set up as a base data type, ***string*** to set the element **story** data type, **string** data only, and then further restrict that value by using the ***length*** to set the number of characters that can be entered in the content of an element to *exactly* 1024 characters.
5. **Deriving a Named Custom Type**
   1. If you going to use a custom type more than once, a **named** custom types allow you to reuse it
   2. Type <xs:simpleType, followed by the name of your custom type, not the element name, which will be defined somewhere else.
   3. You need to following by a <xs:restriction
6. **Specifying a Set of Acceptable Values using *Enumeration***
   1. I can create a custom simple type to create a specific set of allowable values using *enumeration*.
   2. Again, I have to use the command restriction followed by base which sets the element to a certain type of data, in this case it is **string** data. The user can now enter any **string** data into the element **wonder\_name**. But, I want to further limit how this element is used.
   3. I want to enumerate the allowable data that can be entered to all of the listed Wonders of the World. [read list]
   4. I must use the same syntax as before. I use restriction followed by base which is set to whatever data type I want, and then I enumerate what values are allowable using the enumeration command.
   5. You merely set enumeration equal to all of the allowable values.
   6. Each enumeration must be unique.
7. **Limiting the Length of an Element’s content**
   1. We talked about this custom data type already.
   2. By using the restriction, base, and *length* command, you can limit the number of characters entered as the content of an element.
   3. Here I have given the example of how we might limit what data gets entered for the content of the social security element. We need to ensure there is no more than 11 characters entered for the social security number.
8. **XML Schema also allows us to use Patterns**
   1. We can use patterns or Regular Expressions to create customized ways to further restrict how data is entered into our XML documents.
   2. Regular Expressions are also known sometimes as RegEx
   3. Regular Expressions are used in many programming languages.
   4. They provide a concise and flexible way to match text like word, numbers or strings of characters.
   5. In the example, I have made it possible to enforce how data is entered for a Social Security number. The caret or ^ symbol anchors the start of the string.
   6. Then, the backward slash followed by the letter d tells the parser that any digit will follow.
   7. Then, the curly brackets with a number inside of them tell the parser how many numbers must be entered. Here it is three.
   8. A dash mush be entered
   9. The same pattern is done for the rest of the social security number, ending with a dollar sign, which signifies that it is the end of the string of characters.