XML Schema Continued

Lesson 6

Review

Deriving a Named Custom Simple Types (p.126)

```
<xs:simpleType name="story_type">
    <xs:restriction base="xs:string">
        <xs:length value="1024"/>
    </xs:restriction>
</xs:simpleType>
```

References story_type

```
<xs:element name="story"
type="story_type"/>

<xs:element name="summary"
type="story_type"/>

<xs:element name="another_story"
type="story_type"/>
```

What is a Complex Type? It can contain:

- Attribute
- Child element
- Or some combination of the two

There are Four Complex Types:

1. The first is called **"text only"** and is a *complex type* element with *simple content* (allows **text** and **attributes**)
2. The second is called **"element only"** and is a *complex type* element with *complex content* (allows children and attributes)
3. The third is the **"empty element"** and is also a *complex type* element with *complex content.* It's considered complex content because simple content allows text, and, since I am defining an empty element, it **cannot allow text content. WE WILL NOT SEE THESE IN OUR HOMEWORK!**

4. Finally, the fourth *complex type* XML element is called **"mixed content."** This is because it is a *complex type element* with both *complex content* and *simple content* **(allows text, child elements, and attributes)**

Important!

Simple Types- can only define elements with only content no children

Complex Types-can define elements with children and attributes

Complex Types are divided into:

Simple Content – can only have string content and attributes

Complex Content- can have attributes and children

Deriving a Complex Type

- With complex types, there aren't any built in types to use. To use a complex type, it must be *derived*. We worked on this last week were we looked at **Simple Types**
- To derive a **"text only" content type** you can use the **`<xs:simpleContent>`** element.
- To derive an **"element only" complex type** you can use the **`<xs:complexContent>`** element.

Deriving Anonymous Complex Types

As with simple types, you can derive a complex type anonymously, or you can name it (and reuse it throughout the XML Schema). If you don't need to reuse a complex type, it's faster to create it anonymously within the element definition itself.

Note: The only difference between an anonymous type and a **named type is that a named type can be used more than once,** and can be used as the base for new complex types. An ***anonymous type can only be used for the element in which it is contained.***

1. **Deriving Named Complex Types**

Last week we looked at deriving custom simple types and deriving **named custom types.** We did not do any of these in your exercise, but we will create one today. ***Remember simple types can only define elements for content, they cannot define attributes and children.***

- Being able to derive named complex types is the heart of what chapter 11 is about.
- Being able to name your types allows you to reuse your types throughout your schema definition.
- This is a technique that programmers use all of the time when they are writing software. Rather than write a routine or algorithm every time they need to do something, they just write it once, and name it. And, then every time they need to use it, they just call or make a reference to it.
- Also, when you read about parameter entities with DTDs that too was similar to Named Complex Types. We didn't really talk about them in class, but they were a more primitive version of what Schema allows you to do with Complex Types.

- If you are going to use a complex type to define more than one element in your XML Schema, you can create a **named complex type**. Then, each time you want to use it, you can include a ***reference*** between the XML element and your new custom type.

  LOOK AT EXAMPLE 1

2. **Anonymous Complex Types with one Child**

Work through example

Defining Complex Types That Contains a Child Element [NO EXAMPLE]
One of the most common complex types is one that contains **child** elements. This complex type can also contain attributes, but it cannot contain a value of its own. It's described (even with attributes) as "element only." To define an "element only" complex type:
1. Type `<xs:complexType.`
2. Then, type `name="complex_type_name">,` where `complex_type_name` identifies your new complex type.
3. Next, you'll define the structure and order of the child elements of this parent element. You'll declare a **sequence**, an **unordered list**, or a **choice**; each of which is discussed on the next three pages.
4. Then, declare the attributes that should appear in this complex type element, if any.
5. Finally, type `</xs:complexType>` to complete the "element only" complex type definition.

- Remember: the default condition for complex types is that they derive from `xs:complexContent` that restricts `xs:anyType`. Because of this, you can **<u>omit</u> these XML Schema element** declarations.
- The child elements of a complex type are referred to as its *content model*.
- The content model of a complex type must either be a sequence, an unordered list, or a choice. These are called model groups, and they indicate the structure and order of child elements within their parent.

Requiring Child Elements to Appear in Sequence [example]

If you want a complex type element to contain child elements, in order, you have to define a **sequence of those elements**. To require child elements to appear in sequence:

1. Type `<xs:sequence`.
2. If desired, specify how many times the sequence of elements itself can appear by setting the `minOccurs` and `maxOccurs` attributes.
3. Then, type > to complete the opening tag.
4. Declare the simple type elements and/or complex type elements you want in the sequence, in the order in which they should appear.
5. Finally, type `</xs:sequence>` to complete the model group.

- A sequence defines the order in which its child elements must appear in an XML document.
- Since an XML element may only have one child, it's perfectly legitimate for a sequence to contain only one element.
- A sequence can also contain other sequences, choices or references to named groups.
- A sequence may be contained in a complex type definition, other sequences, a set of choices or in named group definitions.
- The `<xs:sequence>` element is basically equivalent to the **comma (,) in DTDs**.

Allowing Child Elements to Appear in Any Order

If you want a complex type element to contain child elements in any order, you can list those children with an all element. To allow child elements to appear in any order:

1. Type `<xs:all` to begin the unordered list of elements.
2. Optionally, you can specify how many times the unordered list itself can appear by setting the `minOccurs` and `maxOccurs` attributes.
3. Then, type > to complete the opening tag.
4. Declare the simple type elements and/or complex type elements that you want in the unordered list.
5. Finally, type `</xs:all>` to complete the model group.

Creating a Set of Choices

It's sometimes useful to declare a complex type element so that it can contain one child element (or a group of child elements) or another. You do that by creating a choice model group. To offer a choice of child elements:

1. Type `<xs:choice.`

2. Optionally, you can specify how many times the set of choices itself can appear by setting the `minOccurs` and `maxOccurs` attributes.
3. Then, type `>` to complete the opening tag.
4. Declare the simple type elements and/or complex type elements that you want to make up the set of choices.
5. Finally, type `</xs:choice>` to complete the model group.

- The default `minOccurs` and `maxOccurs` attribute values are both 1. With these defaults, only one of the elements in a set of choices can appear in a valid XML document. If the value of the `maxOccurs` attribute is greater than 1, that value determines how many of the choices may appear. Using `maxOccurs="unbounded"` is equivalent to adding an asterisk **(*)** to a set of choices in a DTD.
- A set of choices can also contain sequences, additional choice sets, or references to named groups.
- A set of choices may be contained in a complex type definition, in sequences in other sets of choices, or in named group definitions.
- The `<xs:choice>` element is basically equivalent to the **vertical bars in DTDs**.

Defining Empty Elements

Elements that can contain attributes, but have no content between the opening and closing tags are called "empty elements." Since these are complex type elements, they can (and often do), have one or more attributes. To define an **"empty element" complex type**:
1. Type `<xs:complexType`.
2. Then, type `name="complex_type_name">,` where `complex_type_name` identifies your new complex type.
3. Next, declare the attributes that should appear in this complex type element, if any.
4. Finally, type `</xs:complexType>` to complete the complex type definition.

The default condition for complex types is that they derive from `xs:complexContent` that restricts `xs:anyType`. Because of this, you can omit these XML Schema element declarations.

Defining Elements with Mixed Content

While pure database-driven content rarely has elements that contain both child elements and text, more *text-centered documents* wouldn't find it strange at all. A complex type that supports this is called "mixed content." When creating this complex type, you must declare that the content will be mixed. To create a **"mixed content" complex type**:
1. Type `<xs:complexType`.
2. Then, type `name="complex_type_name",` where complex_type_name identifies your

new complex type.
3. Next, type mixed="true"> to indicate that the element can contain elements and text, (and may even contain attributes as well).
4. Declare a *sequence*, an *unordered list or a choice* to specify the child elements and structure within the complex type.
5. Then, declare the attributes that should appear in this complex type element, if any.
6. Finally, type `</xs:complexType>` to complete the complex type definition.

- Mixed content elements are ideal for descriptive, text-based chunks of information. They are not very common in database-type applications.
- The default condition for complex types is that they derive `from xs:complexContent` that restricts `xs:anyType`.

In the exercise I merely have you create an element and then define it as an anonymous type.

Deriving Complex Types from Existing Complex Types

You can also create new complex types based on existing complex types. The new complex type begins with all the information from the existing type (Figure 11.28), and then adds or removes features. To derive a new complex type from an existing type:

1. Type <xs:complexType.
2. Then, type name="complex_type_name", where complex_type_name identifies your new complex type.
3. Type <xs:complexContent>.
4. Next, type <xs:extension to indicate that features will be added to the existing complex type. Or type <xs:restriction to indicate that features will be removed from the existing complex type.
5. Then, type base="existing_type">, where existing_type identifies the name of the existing type from which the new complex type will be derived.
6. Declare the attributes (see page 154) that should be part of the new complex type.
7. Type </xs:complexContent>.
8. Finally, type </xs:complexType> to complete the complex type definition.

- New complex types derived using restrictions must be valid subsets of the existing complex type. Some acceptable restrictions include setting default or fixed values.
- As this is an advanced topic, I have only identified the basics here.

Defining Attributes

Attributes are simple type elements since they contain neither child elements nor attributes. To **define an attribute**:
1. Within the definition of the **complex type**, type `<xs:attribute`.

2. Then, type `name="attribute_name",` where `attribute_name` is the name of the XML attribute that you are defining. The you have the following optiongs for what type of attribute to create:
To use a **base or named simple type**:
3. Type `type="simple_type"/>,` where `simple_type` is the named or base type of the attribute that you are defining.
To use an **anonymous simple type**:
3. Type `>` to complete the opening tag.
4. Then, type `<xs:simpleType>.`
5. Add any **restrictions (or facets)** you like.
6. Next, type `</xs:simpleType>` to close the simple type element.
7. Finally, type `</xs:attribute>` to close the opening tag, see Figure 11.39.
To use a **globally defined** attribute:
3. Type `ref="label`"/>, where label identifies an attribute definition that you've already globally defined.

- Attributes must be defined at the very end of the complex type to which they belong; that is, after all the elements in the complex type have been defined.

Requiring an Attribute

Unless you specify otherwise, an **attribute is always optional**. In other words, it may appear or be absent from a valid XML document. However, if you'd prefer, you **can insist that an attribute be present (or not)**, when determining if the XML document is valid.

To require that an attribute be present:

1. Within an **attribute definition**, type `use="required"` to indicate that the attribute must appear for an XML document to be considered valid.
2. You may also add `value="must_be"`, where `must_be` is the only acceptable value for the attribute. To require that an attribute not be present:

- Within an **attribute definition**, type `use="prohibited"` so that the XML document will only be considered valid if the attribute is not present.
- You could also type `use="optional"` within an attribute definition, but since that's the default condition, it's unnecessary.

Predefining an Attribute's Content

There are **two ways** to use XML Schema to **predefine what an attribute's content** should be. You can either dictate the attribute's content, or set an initial value for the attribute regardless of whether it appears or not. The former is called a fixed value; the latter is called a default value.

To dictate an attribute's content:

- Within an attribute definition, type `fixed="content",` where content determines what the value of the attribute should be for the document to be considered valid. (This only applies if the attribute appears in the XML document.)

To set an attribute's initial value:

- Within an attribute definition, type `default="content",` where content determines the value the that attribute should be set to if it is omitted from the XML document .

- The fixed attribute only sets a value if the attribute actually appears in the XML. If the attribute is omitted, then no content is set.
- If the default attribute is set and the attribute is omitted from the XML document, then the attribute's value is set to the default value.
- If you set the default attribute, the only use attribute value you can have is optional.
- You may not have values for both default and fixed in the same attribute definition.