Week 8 Lecture : Schemas - Part 2

Screencast of Recorded Lecture (2022):

- [forthcoming]

Screencast of Recorded Lecture (2021):

- **https://sjeccd-edu.zoom.us/rec/play/oejQWwE8SYLZflx5ao6jwqvh63EgpXWJTH7W6nc2FhVfLwnov38h7nngwfB98i2jqVTz8ff-W0UTjvQ9.nx8puRrquthycGuR (https://sjeccd-edu.zoom.us/rec/play/oejQWwE8SYLZflx5ao6jwqvh63EgpXWJTH7W6nc2FhVfLwnov38h7nngwfB98i2jqVTz8ff-W0UTjvQ9.nx8puRrquthycGuR)**

Let's continue on with **XML Schemas**.

Let's look at a simple XML file:

```
<?xml version="1.0"?>
<GREETING>
   Hello XML World
</GREETING>
```

Here is a very simple schema for the above XML document:

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="GREETING" type="xsd:string" />
</xsd:schema>
```

The root element of this and other schemas is **schema**. This must be in the **http://www.w3.org/2001/XMLSchema (http://www.w3.org/2001/XMLSchema)** namespace.

Elements are declared using the "element" element with a prefix of xsd or xs -- **xsd:element** (or **xs:element**).

The example above includes a single such element declaring the **GREETING** element. The **name** attribute specifies which element is being declared (GREETING in this example). This **xsd:element** element also has a **type** attribute whose value is the data type of the element. In this case, the type is **xsd:string**, a standard type for elements that can contain any amount of text in any form but not child elements. It's equivalent to a DTD content model of #PCDATA.

Simple vs. Complex Types

The W3C XML Schema language divides elements into complex and simple types.

A **simple type** element is one like GREETING that only contains text and does not have any attributes. It cannot contain any child elements. It may, however, be more limited in the kind of text it can contain. For instance, a schema can say that a simple element contains an integer, a date, or a decimal value between 3.76 and 98.24.

Complex type elements can have attributes and can have child elements. Most XML documents need a mix of both complex and simple elements.

minOccurs and maxOccurs attributes
These attributes specify the minimum and maximum number of instances of the element that may appear at that point in the document. This is done by attaching minOccurs and maxOccurs attributes to each xsd:element element.

Example (in this, the TITLE element must occur once and only once in the XML document):

```
<xsd:element name="TITLE" type="xsd:string" minOccurs="1" maxOccurs="1" />
```

Mixed Content

Schemas offer much greater control over mixed content than DTDs. In particular, schemas let you enforce the order and number of elements appearing in mixed content.

Choices

The **xsd:choice** element is the schema equivalent of the pipe symbol (|) in DTDs. When xsd:element elements are combined inside an xsd:choice, then exactly one of those elements must appear in the instance document.

Example:

```
<xsd:element name="TITLE" type="xsd:string" />
<xsd:choice>
  <xsd:element name="COMPOSER" type="complexType" />
  <xsd:element name="PRODUCER" type="complexType" />
</xsd:choice>
```

Sequences

An **xsd:sequence** element requires each member of the sequence to appear in the same order in the instance document as in the **xsd:sequence** element.

Simple Types

There are 44 built-in simple types in the W3C XML Schema language. These can be unofficially divided into seven groups:

- Numeric types
- Time types
- XML types
- String types
- The Boolean type
- The URI type
- The binary type

In summary we can make a few general statements:

- Schemas address a number of perceived limitations with DTDs, including a strange, non-XML syntax, namespace incompatibility, lack of data typing, and limited extensibility and scalability.
- A XML document can indicate the schema that applies to its non-namespace qualified elements via an **xsi:noNamespaceSchemaLocation** attribute, which is normally placed on the root element.
- An XML document can indicate the schema that applies to its namespace qualified elements via an **xsi:schemaLocation** attribute, which is normally placed on the root element.
- Schemas declare elements with **xsd:element** elements.
- The type attribute of **xsd:element** specifies the data type of that element.

- Elements with complex types can have attributes and child elements.
- Elements with simple types contain only parsed character data and do not have attributes.
- The **xsd:group**, **xsd:all**, **xsd:choice** , and **xsd:sequence** elements let you specify particular combinations of elements in an element's content model.
- The **minOccurs** and **maxOccurs** attributes of *xsd:element* determine how many of a given element are allowed in the instance document at that point. The default for each is 1. **maxOccurs** can be set to unbounded to indicate that any number of the element may appear.
- There are 44 built-in simple types, including many numeric, string, time, and XML types.
- Schemas declare attributes with **xsd:attribute** elements.

Simple Elements vs. Complex Elements

A simple element is one that **only contains** text. It cannot contain any other sub-elements or attributes. If the element has a sub-element or attribute, it's a complex element.

These are all simpleType elements. None of these contain anything other than plain text inside the tags:

- <foo>Hi there</foo>
- <h1>Title of something</h1>
- A simple paragraph
-

But, these are all complexType elements:

- <bar something="nifty" />
  - **bar** is complex because it contains the attribute **something**
- <foo><bar /></foo>
  - **foo** is complex because it contains the element **bar** inside it. (**bar** is simpleType, though!)
- green-colored text in a paragraph

- **p** is complex because it contains the attribute **style**

Defining Simple Elements

```
<xs:element name="ElementName" type="ElementDataType" />
```

You can declare that the element must use a particular data type. Some of the most commonly used data types are:

- **xs:string** – string of characters and numbers
- **xs:decimal** – string of only decimal numbers (but no spaces or other characters)
- **xs:integer** – string of any type of number (but no spaces or other characters)
- **xs:boolean** – true/false
- **xs:date** – date in the YYYY-MM-DD format
- **xs:time** – time in the HH:MM:SS format

For more on data types, visit the W3C's page on datatypes:

https://www.w3.org/TR/xmlschema-2/#built-in-datatypes (https://www.w3.org/TR/xmlschema-2/#built-in-datatypes)

Setting Restrictions

You can specify that a particular element must have certain content, using the **fixed** attribute, or set the default content, using the **default** attribute:

```
<xs:element name="CurrencyMarker" type="xs:string" fixed="$" />
```

This forces the **<CurrencyMarker>** element to only have the dollar sign. If, in the XML file, the element **<CurrencyMarker value="€" />** were used, the XML file would not validate.

```
<xs:element name="CurrencyMarker" type="xs:string" default="$" />
```

This sets the default value for **<CurrencyMarker>** to be the dollar sign, but if there is something else in the XML, it would still validate. In this case, if there were nothing in the content of **<CurrencyMarker />**, it would be set to $.

You can also restrict the content of an XML element to a set of acceptable values, using the tag **<xs:restriction>** and then the appropriate restriction.

Number value restrictions using **xs:minInclusive** and **xs:maxInclusive**

Restricting numbers from 0-10:

```
<xs:restriction base="xs:integer">
      <xs:minInclusive value="0" />
      <xs:maxInclusive value="10" />
</xs:restriction>
```

Restricting numbers from 42-1024:

```
<xs:restriction base="xs:integer">
      <xs:minInclusive value="42" />
      <xs:maxInclusive value="1024" />
</xs:restriction>
```

Content value restrictions using **xs:enumeration**

Restricting the **<university>** element to only Stanford, Harvard, or Yale:

```
<xs:element name="university">
 <xs:simpleType>
 <xs:restriction base="xs:string">
   <xs:enumeration value="Stanford" />
   <xs:enumeration value="Harvard" />
   <xs:enumeration value="Yale" />
 </xs:restriction>
 </xs:simpleType>
</xs:element>
```

or:

```
<xs:element name="university" type="univType" />

<xs:simpleType name="univType">
 <xs:restriction base="xs:string">
   <xs:enumeration value="Stanford" />
   <xs:enumeration value="Harvard" />
   <xs:enumeration value="Yale" />
 </xs:restriction>
</xs:simpleType>
```

Restricting on a pattern, using **xs:pattern**

In this example, the element **<onlyOneLowercaseLetterAllowed>** would only validate if the element has one lowercase letter as its value:

```
<xs:element name="onlyOneLowercaseLetterAllowed">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:pattern value="[a-z]" />
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

In this example, the element **<onlyTwoUpperOrLowercaseLettersAllowed>** would only validate if the element has two letters -- either case -- as its value:

```
<xs:element name="onlyTwoUpperOrLowercaseLettersAllowed">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:pattern value="[a-zA-Z][a-zA-Z]" />
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

In this example, the element **<ChooseLetterXorLetterYorLetterZ>** would only validate if the element has either "X", "Y", or "Z" as its value:

```
<xs:element name="ChooseLetterXorLetterYorLetterZ">
 <xs:simpleType>
 <xs:restriction base="xs:string">
   <xs:pattern value="[XYZ]" />
 </xs:restriction>
 </xs:simpleType>
</xs:element>
```

In this example, the element **<ZipCode>** would only validate if the element has a 5-digit number:

```
<xs:element name="ZipCode">
 <xs:simpleType>
 <xs:restriction base="xs:string">
   <xs:pattern value="[0-9][0-9][0-9][0-9][0-9]" />
 </xs:restriction>
 </xs:simpleType>
</xs:element>
```

You can also specify that the acceptable value would be zero or more combinations, by surrounding the desired pattern in parentheses followed by the * symbol.

In this example, the element **<ZeroOrMoreUpperLetters>** would validate if the element has either nothing, or if there is content, that the content only contains uppercase letters:

```
<xs:element name="ZeroOrMoreUpperletters">
 <xs:simpleType>
 <xs:restriction base="xs:string">
   <xs:pattern value="([A-Z])*" />
 </xs:restriction>
 </xs:simpleType>
</xs:element>
```

To specify that the acceptable value would be one or more combinations, surround the desired pattern in parentheses followed by the + symbol.

In this example, the element **<OneOrMoreNumbers>** would validate if the element has one or more numbers:

```
<xs:element name="OneOrMoreNumbers">
 <xs:simpleType>
 <xs:restriction base="xs:string">
   <xs:pattern value="([0-9])+" />
 </xs:restriction>
 </xs:simpleType>
</xs:element>
```

To specify that the acceptable value would be one from a series of possibilities, use the pipe symbol | in the value.

In this example, the element **<ShirtColor>** would only validate if the element has red, green, or purple:

```
<xs:element name="ShirtColor">
 <xs:simpleType>
 <xs:restriction base="xs:string">
   <xs:pattern value="red|green|purple" />
 </xs:restriction>
 </xs:simpleType>
</xs:element>
```

You can also specify a pattern with a specific number of characters, using the curly brace to identify the desired number of characters

In this example, the element **<password>** would only validate if the element has exactly 8 characters in a row, either lowercase or uppercase or a number from 0-9:

```
<xs:element name="password">
 <xs:simpleType>
 <xs:restriction base="xs:string">
   <xs:pattern value="[a-zA-Z0-9]{8}" />
 </xs:restriction>
 </xs:simpleType>
</xs:element>
```

Restricting the length of a value of an element, using **xs:length**

You can specify the exact number of characters for a value. In this example, the element **<password>** would only validate if the element has exactly 8 characters:

```
<xs:element name="password">
 <xs:simpleType>
 <xs:restriction base="xs:string">
   <xs:length value="8" />
 </xs:restriction>
 </xs:simpleType>
</xs:element>
```

You can also specify the minimum (**xs:minLength**) and maximum (**xs:maxLength**) number of characters for a value. In this example, the element **<password>** would only validate if the element has between 8 and 20 characters:

```
<xs:element name="password">
 <xs:simpleType>
 <xs:restriction base="xs:string">
   <xs:minLength value="8" />
   <xs:maxLength value="20" />
 </xs:restriction>
 </xs:simpleType>
</xs:element>
```

Requiring child elements using a **compositor**

You can also define how child elements inside of a parent element appears in a valid XML file

- **<xs:sequence>**
  all of the child elements in the XML must appear in the order in which they are listed in the XSD.
- **<xs:choice>**
  only one of the child elements listed in the XSD must appear in the XML
- **<xs:all>**
  all of the child elements listed in the XSD must appear in the XML, but in any order

Here are some sample XML code examples, the schema applied to the code, and whether the XML is valid:

Valid/Invalid XML based on XSD

In the XML File	In the XSD	Valid?

`<director>`
`<firstname>George</firstnam`
`e>`
`<lastname>Lucas</lastname>`
`</director>`
	`<xs:sequence>`
	`<xs:element name="director">`
	`<xs:complexType>`
	`<xs:sequence>`
	`<xs:element name="lastname" type="xs:string" />`
	`<xs:element name="firstname" type="xs:string" />`
	`</xs:sequence>`
	`</xs:complexType>`
		Not valid, because in the XML file, the order goes director, then firstname and lastname, but in the XSD, the firstname and lastname elements are in different order, and it's using xs:sequence

`<director>`
`<firstname>George</firstnam`
`e>`
`<lastname>Lucas</lastname>`
`</director>`
	`<xs:sequence>`
	`<xs:element name="director">`
	`<xs:complexType>`
	`<xs:choice>`
	`<xs:element name="lastname" type="xs:string" />`
	`<xs:element name="middlename" type="xs:string" />`
	`<xs:element name="firstname" type="xs:string" />`
		Valid, because despite that firstname and

	<xs:choice> </xs:complexType>	lastname elements are in different order, and that there's no middlename, it's using xs:choice
<director> <firstname>George</firstname> <lastname>Lucas</lastname> </director>	<xs:sequence> <xs:element name="director"> <xs:complexType> <xs:all> <xs:element name="firstname" type="xs:string" /> <xs:element name="middlename" type="xs:string" /> <xs:element name="lastname" type="xs:string" /> </xs:all> </xs:complexType>	**Not valid**, because despite that firstname and lastname elements are in correct order, there's no middlename, and the XSD is using xs:all, which requires all elements to be present.

Datatype restrictions

Datatype Constraints and Descriptions

Constraint	Description
xs:enumeration	Defines a list of acceptable values
xs:fractionDigits	Specifies the maximum number of decimal places allowed. Must be equal to or greater than zero

xs:length	Specifies the exact number of characters or list items allowed. Must be equal to or greater than zero
xs:maxExclusive	Specifies the upper bounds for numeric values (the value must be less than this value)
xs:maxInclusive	Specifies the upper bounds for numeric values (the value must be less than or equal to this value)
xs:maxLength	Specifies the maximum number of characters or list items allowed. Must be equal to or greater than zero
xs:minExclusive	Specifies the lower bounds for numeric values (the value must be greater than this value)
xs:minInclusive	Specifies the lower bounds for numeric values (the value must be greater than or equal to this value)
xs:minLength	Specifies the minimum number of characters or list items allowed. Must be equal to or greater than zero
xs:pattern	Defines the exact sequence of characters that are acceptable
xs:totalDigits	Specifies the exact number of digits allowed. Must be greater than zero
whiteSpace	Specifies how white space (line feeds, tabs, spaces, and carriage returns) is handled

Occurrence Indicators

These are used to define how often an element is allowed to occur.

- **maxOccurs** - indicates the maximum number of times the element may occur (use *unbounded* for an unlimited number of times)
- **minOccurs** - indicates the minimum number of times the element may occur (default is 1 -- if there is a possibility that the element won't exist in the XML file, be sure to set minOccurs to 0)

Suppose we have the following XML file which listed all of the *Star Wars* movies that existed in March of 2018:

```
<starwars>
 <movie>
   <title>Star Wars: A New Hope</title>
```

```
  </movie>
  <movie>
    <title>Empire Strikes Back</title>
  </movie>
  <movie>
    <title>Return of the Jedi</title>
  </movie>
  <movie>
    <title>The Phantom Menace</title>
  </movie>
  <movie>
    <title>Attack of the Clones</title>
  </movie>
  <movie>
    <title>Revenge of the Sith</title>
  </movie>
  <movie>
    <title>The Force Awakens</title>
  </movie>
  <movie>
    <title>Rogue One</title>
  </movie>
  <movie>
    <title>The Last Jedi</title>
  </movie>
</starwars>
```

In the XSD, we might want to set the movie element to have a minOccurs set at 0 and a maxOccurs set to 9, since there were 9 films at that point in the series:

```
<xs:element name="movie" type="xs:string" minOccurs="0" maxOccurs="9" />
```

The only problem with setting the maxOccurs to 9 is that a new *Star Wars* movie opened in May 2018.

Which means that if the XML file is updated to include the new Han Solo movie *Solo*, we'd have 10.

And the 11th film opened in December of 2019 *(The Rise of Skywalker)*.

What to do?

One fix would be to update maxOccurs to 10 or 11 or some other higher number. But a better fix would be to set the maxOccurs to **unbounded**, which indicates that there could be an unlimited number of that element.

```
<xs:element name="movie" type="xs:string" minOccurs="0" maxOccurs="unbounded" />
```

Complex Elements

Let's talk more in-depth about complex elements. A complex element is an XML element that contains other elements and/or attributes.

There are four kinds of complex elements:

1. empty elements with one or more attribute
2. elements that contain only other elements
3. elements that contain only text and attributes (simpleContent)
4. elements that contain both other elements and text

Each of these elements may contain attributes as well.

Complex Elements Examples

Here is a complex XML element -- **img --** which is *empty*, but *contains attributes*:

```
<img src="/images/picture.jpg" alt="a beautiful picture of the ocean" />
```

Here is another complex XML element -- **director** -- which contains *only other elements*:

```
<director>
  <firstname>George</firstname>
  <lastname>Lucas</lastname>
</director>
```

Here is yet another complex XML element -- **title** -- which contains *only text and an attribute*:

```
<title ranking="2">A New Hope</title>
```

And here is a complex XML element -- **publication** -- which contains *both elements and text*:

```
<publication>
  <distributor>20th Century Fox</distributor> released on <date>1977-05-27</date>
```

```
</publication>
```

Defining Complex Elements

Look at this complex XML element -- director -- which contains the elements **firstname** & **lastname**:

```
<director>
  <firstname>George</firstname>
  <lastname>Lucas</lastname>
</director>
```

We can define a complex element in an XML Schema two different ways:

1. The **director** element can be declared directly by naming the element:

```
<xs:element name="director">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="firstname" type="xs:string" />
      <xs:element name="lastname" type="xs:string" />
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

If you use the method described above, only the **director** element can use the specified complex type. Note that the child elements, "firstname" and "lastname", are surrounded by the <xs:sequence> indicator. **Again, this means that the child elements must appear in the same order as they are declared.**

2. The **director** element can have a type attribute that refers to the name of the complex type to use:

```
<xs:element name="director" type="personInfo" />

<xs:complexType name="personInfo">
  <xs:sequence>
    <xs:element name="firstname" type="xs:string" />
    <xs:element name="lastname" type="xs:string" />
  </xs:sequence>
</xs:complexType>
```

Using method #2, several elements can refer to the same complex type:

```
<xs:element name="director" type="personInfo" />
<xs:element name="producer" type="personInfo" />
<xs:element name="actor" type="personInfo" />

<xs:complexType name="personInfo">
  <xs:sequence>
    <xs:element name="firstname" type="xs:string" />
    <xs:element name="lastname" type="xs:string" />
  </xs:sequence>
</xs:complexType>
```

You can also base a complex element on an *existing* complex element and add some elements:

```
<xs:element name="director" type="fullpersoninfo" />

<xs:complexType name="personinfo">
  <xs:sequence>
    <xs:element name="firstname" type="xs:string" />
    <xs:element name="lastname" type="xs:string" />
  </xs:sequence>
</xs:complexType>

<xs:complexType name="fullpersoninfo">
  <xs:complexContent>
    <xs:extension base="personinfo">
      <xs:sequence>
        <xs:element name="phone" type="xs:string" />
        <xs:element name="email" type="xs:string" />
        <xs:element name="website" type="xs:string" />
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

Complex Elements #1: Empty Elements With Attributes

Empty elements are those that have no content inside them. In XML, we define an empty element using the forward slash to terminate the tag.

Here's an example, using the XHTML tag **img** (with the attributes **src** and **alt**)

```
<img src="/images/picture.jpg" alt="a beautiful picture of the ocean" />
```

To describe this in a schema:

```
<xs:element name="img">
  <xs:complexType>
     <xs:attribute name="src" type="xs:string" />
     <xs:attribute name="alt" type="xs:string" />
  </xs:complexType>
</xs:element>
```

If you want to force an attribute to be used in order for the XML file to be valid, use the **use="required"** attribute in the **xs:attribute**.

use="optional" is the default.

```
<img src="/images/picture.jpg" />
```

This schema would say that the above code is invalid (the alt attribute is required, but missing):

```
<xs:element name="img">
  <xs:complexType>
     <xs:attribute name="src" type="xs:string" />
     <xs:attribute name="alt" type="xs:string" use="required" />
  </xs:complexType>
</xs:element>
```

Complex Elements 2: Elements That Contain Only Other Elements

I mentioned this above, when I first started talking about complex elements (the **director** element contains only other elements - **firstname** and **lastname)**:

```
<director>
 <firstname>George</firstname>
 <lastname>Lucas</lastname>
</director>
```

A valid XSD for this might be:

```
<xs:element name="director">
```

```
  <xs:complexType>
    <xs:sequence>
      <xs:element name="firstname" type="xs:string" />
      <xs:element name="lastname" type="xs:string" />
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Complex Elements 3: Elements That Contain Only Text and Attributes (simpleContent)

If the XML is supposed to contain only simple content (text and attributes), we can surround the **xs:simpleContent** element around the content. When using simple content, you must define an extension OR a restriction:

```
<xs:element name="somename">
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="basetype">
           ...
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>
```

OR

```
<xs:element name="somename">
  <xs:complexType>
    <xs:simpleContent>
      <xs:restriction base="basetype">
           ...
      </xs:restriction>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>
```

Let's say you are creating this XML file for the people in a classroom:

```
<personname role="teacher">Bob Smith</personname>
<personname role="student">Sally Student</personname>
```

The section of the XSD dealing with the **personname** element might have:

```
<xs:element name="personname">
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="xs:string">
        <xs:attribute name="role" type="xs:string" />
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>
```

or, if we want to re-use the type **nametype** for other elements:

```
<xs:element name="personname" type="nametype" />
<xs:complexType name="nametype">
 <xs:simpleContent>
 <xs:extension base="xs:string">
 <xs:attribute name="role" type="xs:string" />
 </xs:extension>
 </xs:simpleContent>
</xs:complexType>
```

Complex Elements 4: Mixed Elements That Contain Both Other Elements and Text

Mixed complex type elements contain attributes, elements, and/or text.

Consider the following XML:

```
<studentletter>
  Dear <name>Sally Student</name>,
  Thank you for taking <classname>XML Fundamentals</classname>, which ended on <enddate>2022-12-15</en
ddate>.
  Your overall score in the class was <score>92</score>, which means you earned this grade: <finalgrad
e>A</finalgrade>.
</studentletter>
```

The following schema declares the **studentletter** element:

```
<xs:element name="studentletter">
  <xs:complexType mixed="true">
    <xs:sequence>
      <xs:element name="name" type="xs:string" />
      <xs:element name="classname" type="xs:string" />
      <xs:element name="enddate" type="xs:date" />
      <xs:element name="score" type="xs:positiveInteger" />
      <xs:element name="finalgrade" type="xs:string" />
    </xs:sequence>
```

```
  </xs:complexType>
</xs:element>
```

I needed to use **mixed="true"** because between **<studentletter>** and **</studentletter>**, there are content other than just elements ("Dear", "Thank you for taking", etc.). To enable character data to appear between the child-elements, you must include **mixed="true"**.

If we want to re-use the complexType for other elements, we could give it a name and then apply it to other elements (I'll name it **letterType**):

```
<xs:element name="studentletter" type="letterType" />

<xs:complexType name="letterType" mixed="true">
  <xs:sequence>
      <xs:element name="name" type="xs:string" />
      <xs:element name="classname" type="xs:string" />
      <xs:element name="enddate" type="xs:date" />
      <xs:element name="score" type="xs:positiveInteger" />
      <xs:element name="finalgrade" type="xs:string" />
  </xs:sequence>
</xs:complexType>
```

(I used **xs:sequence** to indicate that the *name*, *classname*, *enddate*, *score*, and *finalgrade* needed to be present and in that order; as discussed earlier in this lecture, I could have chosen to use **xs:choice** or **xs:all**.)

The **xs:any** Element

The **xs:any** element allows an XML document to include elements not specified by the schema. If you include an *xs:any* element to a schema, you should add a *minOccurs="0"* indicator.

```
<xs:any minOccurs="0" />
```

The **xs:anyAttribute** Element

The **xs:anyAttribute** element allows an XML document to include attributes not specified by the

schema.

```
<xs:anyAttribute />
```

OK, I think that's enough about Schemas.