

Theme 3: XML Schema

Complex Types

Defining Complex Types

- Simple type elements **may not** be empty, and...
- ...they **may not** contain child elements or attributes.
- For those, you must define a **complex type** element.
- There are no built-in complex types. You must create one.
- A complex type may be **anonymous** or **named**.

Defining Complex Types

- **An element rule with an anonymous complex type:**

```
<xs:element name="tagName">
  <xs:complexType>
    <!-- The content model -->
  </xs:complexType>
</xs:element>
```

Defining Complex Types

- **A named complex type and a rule using it:**

```
<xs:complexType name="typeName">  
    <!-- The content model -->  
</xs:complexType>  
  
...  
<xs:element name="tagName" type="typeName"/>
```

The Content Model

- A complex type's **content model** can consist of:
 1. The **content type**;
 2. A **model group** if there are child elements, and
 3. Any attribute rules.
- The **content type** of a complex type can be either:
 - Simple content, or
 - Complex content.

Content Types

- **Simple content** refers to text.
- **Complex content** refers to child elements.
- By default, the content type is **complex content**...
- ...so you must only explicitly specify the content type...
- ...if the content type is **simple content**.

Text-Only Complex Types

- A **simple type** element can contain **only text**, but...
- ...a **complex type** element with **simple content**...
- ...can contain **text and attributes**.
- For the abovementioned complex type, you must specify:
 - What type of text it must contain (i.t.o. a **simple type**), then
 - Extend it with one or more **attribute rules**.

Text-Only Complex Types

- **E.g. an element that contains a string and one attribute:**

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


Model Group

Child Elements

- **To define element with child elements:**

```
<xs:element name="elName">
    <xs:complexType>
        <!-- Model group -->
        <!-- Attribute rules -->
    </xs:complexType>
</xs:element>
```

Child Elements

- You will use a **model group** to dictate...
- ...the structure and order of the child elements.
- Then, you will define attributes for the element, if any.
 - Attribute rules must **always** appear **after** the model group.

Model Groups

- The **model group** can be one of the following:
 - **xs:sequence** for an ordered sequence of children;
 - **xs:choice** for a choice between children;
 - **xs:all** for an unordered list of children, or
 - A nest combination of **xs:sequence** and **xs:choice**.
- E.g. a sequence of choices, or a choice between sequences.

Model Groups

- E.g. a sequence of children to appear in that order:

```
<xs:complexType name="typeName">
  <xs:sequence>
    <xs:element name="child1" type="xs:string"/>
    <xs:element name="child2" type="xs:decimal"/>
  </xs:sequence>
  <!--Attribute rules -->
</xs:complexType>
```

Model Groups

- **Note:** An element rule within a complex type...
- ...can also define an anonymous type...
(within the **xs:element** tags)
- ...or refer to an existing named type.

Model Groups (Continued)

In a complex type definition that allows child elements...

- ...you can use **xs:choice** as the model group...
- ...to allow the XML author to choose between children.

Model Groups (Continued)

- **E.g. an element with either an A child, or a B child:**

```
<xs:element name="tagName">
  <xs:complexType>
    <xs:choice>
      <xs:element name="A" type="xs:string"/>
      <xs:element name="B" type="xs:string"/>
    </xs:choice>
    <!-- Attribute rules, if any -->
  </xs:complexType>
</xs:element>
```


Model Groups (Continued)

- Use **xs:all** to allow an unordered list of child elements:

```
<xs:complexType name="typeName">
  <xs:all>
    <xs:element name="A" type="xs:string"/>
    <xs:element name="B" type="xs:string"/>
  </xs:all>
  <!-- Attribute rules, if any -->
</xs:complexType>
```

Model Groups (Continued)

- In the example on the previous slide...
- ...the author must add both **A** and **B**...
- ...but they may be in any order.
- **Note:** You may not nest **xs:all** with other model groups.

Nesting Model Groups

- **You can nest** xs:sequence **and** xs:choice **in many combinations:**

```
<xs:complexType name="typeName">
  <xs:sequence>
    <xs:choice>
      <xs:element name="A" type="xs:string"/>
      <xs:element name="a" type="xs:string"/>
    </xs:choice>
    <xs:element name="B" type="xs:string"/>
  </xs:sequence>
</xs:complexType>
```

Empty Elements

- You must define an empty element as a **complex type**...
- ...but you don't define a model group for it.
- **E.g.:**

```
<xs:element name="tagName">
  <xs:complexType>
    <!-- Attribute rules, if any -->
  </xs:complexType>
</xs:element>
```

Mixed Content Elements

- To define a mixed content element...
- ...you must set **mixed="true"** (default: **false**).

```
<xs:element name="tagName">
  <xs:complexType mixed="true">
    <!-- Model group -->
    <!-- Attribute rules, if any -->
  </xs:complexType>
</xs:element>
```

Mixed Content Elements

- **Note:** You cannot use the **xs:all** model group...
- ...in the complex type of a mixed content element.
- **Mixed content** means that the element so defined...
- ...can contain a mixture of text and the defined children.

Deriving New Types

Restrict and Extend

Deriving New Complex Types

- You can derive new complex types from existing ones.
- You will either **extend** or **restrict** the existing type.
- **Extend:** Extends the range of values provided by an existing **type**
 - E.g., adding child elements or attributes.
- **Restrict:** Creating a new **type** from existing **type** based on criteria
 - E.g., setting a default/fixed value for something.

Deriving New Complex Types

- **Creating a new type by** extending **an existing one:**

```
<xs:complexType name="newType">
  <xs:complexContent> <!-- required in this case -->
    <xs:extension base="existingType">
      <xs:sequence>
        <xs:element name="C" type="xs:string"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

Deriving New Complex Types

- The **newType** in the previous example...
- ...will have the same content model as **existingType**...
- ...but it adds the element, **C**, to the end...
- ...of the existing **xs:sequence** model group.

Deriving New Complex Types

In some cases when applying restrictions...

- You need to simply reference the existing type, or...
- ...you need to duplicate the existing type's content model to add the refinement (e.g. **fixed** attribute).
- This is purely dependant on what type of restriction you are adding to an existing type.

Deriving New Complex Types

- **E.g. if this is the existing type:**

```
<xs:complexType name="existingType">
  <xs:sequence>
    <xs:element name="A" type="xs:string"/>
    <xs:element name="B" type="xs:string"/>
  </xs:sequence>
</xs:complexType>
```

Deriving New Complex Types

- **...then it can be restricted like this:**

```
<xs:complexType name="newType">
  <xs:complexContent> <!-- required in this case -->
    <xs:restriction base="existingType">
      <xs:sequence>
        <xs:element name="A" type="xs:string" fixed="blah"/>
        <xs:element name="B" type="xs:string"/>
      </xs:sequence>
    </xs:restriction>
  </xs:complexContent>
</xs:complexType>
```

Attributes

Attributes

- An attribute is always defined in terms of a **simple type**.
- **E.g. an attribute rule:**

```
<xs:attribute name="attName" type="xs:decimal"/>
```

- By default, the XML author need not use the attribute.
- Add the **use** attribute to set it to be **required**.

Attributes

- **E.g. an attribute that the author MUST use:**

```
<xs:attribute name="attName" type="xs:decimal" use="required"/>
```

- Just like with elements, you can add...
- ...a **fixed** or **default** attribute to the attribute rule...
- ...to set a fixed or default value for it.

Occurrences

- You can control how many times an **element...**
- ...or **model group** may appear in the XML doc.
- Use the attributes, **minOccurs** and/or **maxOccurs**.
- If left out, both attributes default to the value, **1**.
- Thus, by default, all items must appear once only.

Occurrences

- Both attributes must contain a non-negative integer...
- ...but **maxOccurs** may contain the keyword, **unbounded**.
- **unbounded** = as many times as you want.
- You may add one or the other, or both.
- Whichever one is left out defaults to **1**.

Occurrences

- **E.g. allowing an element to appear without restrictions:**

```
<xs:element name="A" type="xs:string" maxOccurs="unbounded"/>
```

- **E.g. allowing a model group to occur three times:**

```
<xs:sequence maxOccurs="3">  
    <!-- element rules -->  
</xs:sequence>
```

Occurrences

Note:

- You may not set an **xs:all** nor any elements inside it...
- ...to appear more than **once**.

Occurrences

- Now we can define a mixed content type...
- ...in which the children can occur in any order:

```
<xs:complexType name="typeName" mixed="true">  
    <xs:choice minOccurs="0" maxOccurs="unbounded">  
        <!-- element rules -->  
    </xs:choice>  
</xs:complexType>
```

Named Model Groups

- You can create model groups outside of complex types...
- ...that you can then reference in multiple complex types.
- **E.g. creating a named xs:sequence group:**

```
<xs:group name="groupName">
    <xs:sequence>
        <!-- element rules -->
    </xs:sequence>
</xs:group>
```

Named Model Groups

- **xs:group** must be a child of **xs:schema** (i.e. global).
- Now you can reference it like this:

```
<xs:complexType name="typeName">  
    <xs:group ref="groupName"/>  
    <!-- Attribute rules, if any -->  
</xs:complexType>
```

Named Attribute Groups

- You can group reusable attribute rules together as well.

```
<xs:attributeGroup name="attGroupName"> <!-- global -->  
    <!-- Attribute rules -->  
</xs:attributeGroup>
```

- Then, to reference it anywhere you can put attribute rules:

```
<xs:attributeGroup ref="attGroupName" />
```


Global and Local definition

Local vs. Global Definitions

- These are defined **globally** (as children of **xs:schema**):
 - The element definition of the **root element**;
 - **Named** types (as opposed to anonymous types);
 - Named model and attribute groups.
- All other items are defined **locally** (within something).
- You can, however, define other elements or attributes globally, if you want.

Local vs. Global Definitions

- **To define an element globally:**

```
<xs:schema>
    ...
    <xs:element name="A" type="xs:string"/>
    ...
</xs:schema>
```

Local vs. Global Definitions

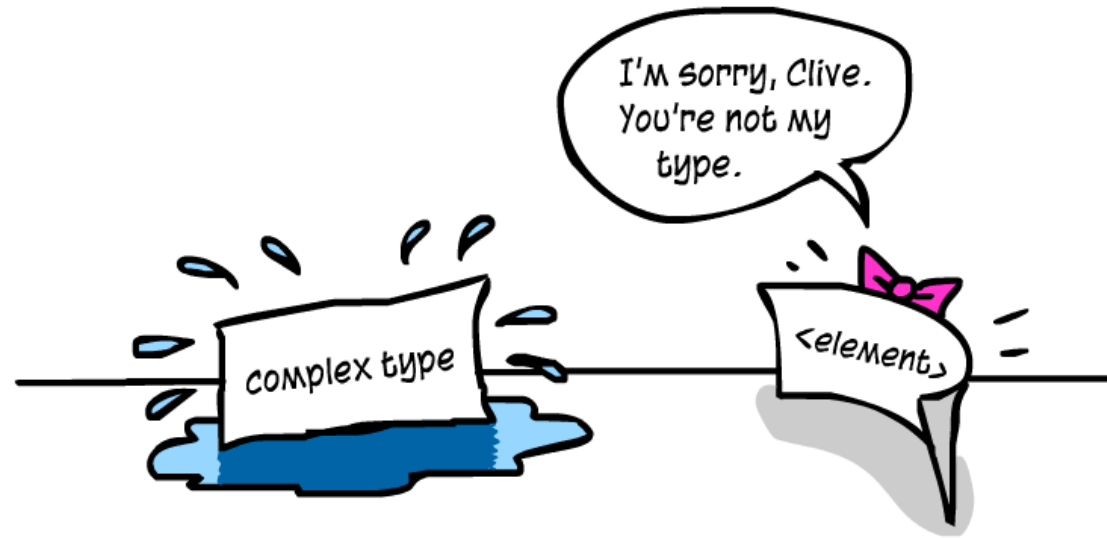
- **You must then reference it locally:**

```
<xs:complexType name="typeName">  
  <xs:sequence>  
    <xs:element ref="A"/>  
  </xs:sequence>  
</xs:complexType>
```

Local vs. Global Definitions

- Global element/attribute definitions...
 - ...except for the root element's definition...
 - ...are **ignored** unless they are referenced locally.
-
- Place **minOccurs** and **maxOccurs** in the reference...
 - ...instead of in the global definition.

Theme 3: XML Schema



END OF THEME 3