Ch. 11: Defining Complex Types

Seoul National University, Internet Database Laboratory

July, 2015

Contents

- Complex Type Basics
- Deriving Anonymous Complex Types
- Deriving Named Complex Types
- Defining Complex Types That Contain Child Elements
- Requiring Child Elements to Appear in Sequence
- Allowing Child Elements to Appear in Any Order
- Creating a Set of Choices
- Defining Elements to Contain Only Text
- Defining Empty Elements
- Defining Elements with Mixed Content

Contents

- Deriving Complex Types from Existing Complex Types
- Referencing Globally Defined Elements
- Controlling How Many
- Defining Named Model Groups
- Referencing a Named Model Group
- Defining Attributes
- Requiring an Attribute
- Predefining an Attribute's Content
- Defining Attribute Groups
- Referencing Attribute Groups
- Local and Global Definitions

Complex Type Basics [1/2]

- Complex type element
  - Can have **child elements** and/or **attributes**
  - Further subdivided into
    - Simple content: Only allows string content
    - Complex content: Allows child elements
    - Both can have attributes

- Four complex types
  - Text only
  - Element only
  - Empty
  - Mixed content

Complex Type Basics [2/2]

- Four complex types

[xml : text only]

```
<year_built ear="BC">
        282</year_built>
```

Figure11.1

[xml : element only]

```
<ancient_wonders>
   <wonder>
...
   </wonder>
</ancient_wonders>
```

Figure11.2

[xml :empty element]

```
<source sectionid="101"
   newspaper="21" />
```

Figure11.3

[xml : mixed content]

```
<story>In 294 BC, the people of the
island of Rhodes began building a
colossal statue of the sun god
Helios. They believed that it was
because of his blessings that they
were able to withstand a long siege
on the island and emerge victorious.
<para/>
The Colossus was built with bronze,
reinforced with iron, ...
</story>
```

Figure11.4

Deriving Anonymous Complex Types [1/4]

- Anonymous complex types: Complex type with no name

[xsd : anonymous complex type definition]

```
<xs:element name = "year_built“ >
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension
       base="xs:positiveInteger">
        <xs:attribute name="era"
         type="xs:string"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>
```

Figure11.5

[xml : valid XML element]

```
<year_built era=“BC”>
   282
</year_built>
```

Figure11.1

Deriving Anonymous Complex Types [2/4]

- Anonymous complex types: Complex type with no name

[xsd : anonymous complex type definition]

```
<xs:element name="year_built">
 <xs:complexType>
  <xs:complexContent>
   <xs:restriction
     base="xs:anyType">
    <xs:sequence>
     <xs:element name="wonder"
       type="wonderType"/>
    </xs:sequence>
   </xs:restriction>
  </xs:complexContent>
 </xs:complexType>
</xs:element>
```

Figure11.6

[xml : valid XML element]

```
<ancient_wonders>
   <wonder>
...
   </wonder>
</ancient_wonders>
```

Figure11.2

Deriving Anonymous Complex Types [3/4]

- Anonymous complex types: Complex type with no name

[xsd : anonymous complex type definition]

```
<xs:element
name="ancient_wonders">
 <xs:complexType>
  <xs:sequence>
   <xs:element name="wonder"
     type="wonderType"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
```

Figure11.7

[xml : valid XML element]

```
<ancient_wonders>
   <wonder>
...
   </wonder>
</ancient_wonders>
```

Figure11.2

Deriving Anonymous Complex Types [4/4]

- Anonymous complex types: Complex type with no name

[xsd : anonymous complex type definition]

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

  <xs:complexType>

    <xs:simpleContent>

      <xs:extension
        base="xs:positiveInteger">

        <xs:attribute name="era"
          type="xs:string"/>

      </xs:extension>

    </xs:simpleContent>

  </xs:complexType>

</xs:element>
```

Figure11.8

[xml : valid XML element]

```
<year_built era="BC">
   282</year_built>
```

Figure11.9

Deriving Named Complex Types

- Named complex type: It can be reused

[xsd : definition of named complex type]

```
<xs:complexType name="yearType">

  <xs:simpleContent>

    <xs:extension
     base="xs:positiveInteger">

      <xs:attribute name="era"
       type="xs:string"/>

    </xs:extension>

  </xs:simpleContent>

</xs:complexType>
```

Figure11.10

[xsd : use the new type by name]

```
<xs:complexType name="historyTpe">

  <xs:sequence>

    <xs:element name="year_built"
      type="yearType"/>

    <xs:element name="year_destroyed"
      type="yearType"/>
```

Figure11.11

[xml : valid XML instance]

```
<year_built era="BC">
  282</year_built>
<year_destroyed era="BC">
  226</year_destroyed>
```

Figure11.12

Defining Complex Types That Contain Child Elements [1/2]

- Element only type

[xsd]

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

<xs:complexType>

 <xs:complexContent>

  <xs:restriction base="xs:anyType">

    <xs:sequence>

      <xs:element name="wonder“
       type="wonderType"/>

    </xs:sequence>

   </xs:restriction>

 </xs:complexContent>

</xs:complexType>

</xs:element>
```

=

[xsd]

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

<xs:complexType>

  <xs:sequence>

    <xs:element name="wonder“
      type="wonderType"/>

  </xs:sequence>

</xs:complexType>

</xs:element>
```

Figure11.14

Figure11.13

Defining Complex Types That Contain Child Elements [2/2]

- The default condition
  - *"complex content that restricts anyType"*
  - You can and should always omit the ***<xs:complexContent>*** and ***<xs:restriction base="anyType">***
- Content model
  - The child elements of complex type
- Model groups
  - The structure and order of child elements within their parent

Requiring Child Elements to Appear in Sequence

- Order of child elements
  - Child elements must appear in an XML document
  - <xs:sequence> is basically equivalent to the comma (,) in DTD

Order of child elements

[xsd]

```
<xs:complexType name="wonderType">

  <xs:sequence>

   <xs:element name="name"
    type="nameType"/>

   <xs:element name="location"
    type="xs:string"/>

   <xs:element name="height"
    type="heightType"/>

   <xs:element name="history"
    type="historyType"/>

   <xs:element name="main_image"
    type="imageType"/>

   <xs:element name="source"
    type="sourceType"/>

  </xs:sequence>

</xs:complexType>
```

Figure11.15

Allowing Child Elements to Appear in Any Order

- In any order

[xsd]

```
<xs:complexType name="historyType">

  <xs:all>

    <xs:element name="year_built"
     type="yearType"/>

    <xs:element name="year_destroyed"
     type="yearType"/>

    <xs:element name="how_destroyed"
     type="destrType"/>

    <xs:element name="story"
     type="storyType"/>

  </xs:all>

</xs:complexType>
```

Figure11.16

[xml]

```
<history>
  <year_built era="BC">
    282</year_built>
  <story>In 294 BC, the people of the
    island of Rhodes began building a
    colossal statue ...</story>
  <year_destroyed era="BC">
    226</year_destroyed>
  <how_destroyed>
    earthquake</how_destroyed>
</history>
```

Figure11.17-a

```
<history>
  <story>In 294 BC, the people of the
    island of Rhodes began building a
    colossal statue ...</story>
  <year_built era="BC">
    282</year_built>
  <how_destroyed>
    earthquake</how_destroyed>
  <year_destroyed era="BC">
    226</year_destroyed>
</history>
```

Figure11.17-b

Allowing Child Elements to Appear in Any Order

- The members of an <xs:all> element
  - May appear once or not at all
    - Depending on their individual ***minOccurs*** and ***maxOccurs*** attributes
  - The minOccurs attribute
    - May only set to 0 or 1
  - The maxOccurs attribute
    - May only be set to 1

- <xs:all> element
  - Can only contain individual element declarations or references
    - Not other groups
  - Can only be contained in, and must be **the sole child of** , an element only complete type definition

Creating a Set of Choices [1/2]

- To offer a choice of child elements

[xsd]

```
<xs:complexType name="wonderType">
  <xs:sequence>
    <xs:element name="name"
     type="nameType"/>
    <xs:choice>
      <xs:element name="location"
       type="xs:string"/>

      <xs:sequence>
        <xs:element name="city"
         type="xs:string"/>
        <xs:element name="country"
         type="xs:string"/>
      </xs:sequence>

    </xs:choice>
...
  </xs:sequence>
</xs:complexType>
```

Figure11.18

[xml]

```
<wonder>
  <name language="English">
    Colossus of Rhodes</name>
  <location>Rhodes, Greece</location>
...
</wonder>
```

Figure11.19-a

[xml]

```
<wonder>
  <name language="English">
    Colossus of Rhodes</name>
  <city>Rhodes</city>
  <country>Greece</country>
...
</wonder>
```

Figure11.19-b

Creating a Set of Choices [2/2]

- Default condition
  - ***minOccurs*** and ***maxOccurs*** attribute values are both ***1***
    - Only one of the elements in a set of choices can appear in a valid XML document

- maxOccurs="unbounded"
  - Equivalent to adding an asterisk (*) to a set of choices in a DTD

- The <xs:choice> element is basically equivalent to **the vertical bars in DTD**

Defining Elements to Contain Only Text

- Text only type
  - It contains a **text value** and **no child elements**
  - It can have one or more attributes

[xsd]

```
<xs:complexType name="yearType">

  <xs:simpleContent>

    <xs:extension base=
      "xs:positiveInteger">

      <xs:attribute name="era"
        type="xs:string"/>

    </xs:extension>

  </xs:simpleContent>

</xs:complexType>
```

Figure11.20

[xml]

```
<year_built era="BC">
  282</year_built>
```

Figure11.21

[xml]

```
<year_built era="BC">
  long ago</year_built>
```

Figure11.22

Defining Empty Elements

- To define an "empty element" complex type

[xsd]

```
<xs:complexType name="sourceType">

  <xs:attribute name="sectionid"
    type="xs:positiveInteger"/>

  <xs:attribute name="newspaperid"
    type="xs:positiveInteger"/>

</xs:complexType>
```

Figure11.23

[xml]

```
<source sectionid="101"
  newspaperid="21"/>
```

Figure11.25

=

[xsd]

```
<xs:complexType name="sourceType">

  <xs:complexContent>

    <xs:restriction base="xs:anyType">

      <xs:attribute name="sectionid"
        type="xs:positiveInteger"/>

      <xs:attribute name="newspaperid"
        type="xs:positiveInteger"/>

    </xs:restriction>

  </xs:complexContent>

</xs:complexType>
```

Figure11.24

Defining Elements with Mixed Content

- Mixed content type
  - Contain both child elements and text

[xsd]

```
<xs:complexType name="story"
 mixed="true">

  <xs:sequence>

    <xs:element name="para"
      maxOccurs="unbounded">

      <xs:complexType/>

    </xs:element>

  </xs:sequence>

</xs:complexType>
```

Figure11.26

[xml]

```
<story>
  In 294 BC, the people of the island
  of Rhodes began building a colossal
  statue of the sun god Helios. They
  believed that it was because of his
  blessings that they were able to
  withstand a long siege on the
  island and emerge victorious.
  <para/>
  The Colossus was built with bronze,
  reinforced with iron, and weighted
  with stones. While it is often
  depicted straddling Mandrákion
  harbor, this is now considered
  technically impossible; and
  therefore, it likely stood beside
  the harbor.
  <para/>
  The statue was toppled by an
  earthquake in 226 BC. ...
</story>
```

Figure11.27

Deriving Complex Types from Existing Complex Types [1/2]

- Derive a new complex type from an existing type
  - **Extension**
    - Add new element to the existing type
  - **Restriction**
    - You duplicate the base type and then refine it

[xsd]

```
<xs:complexType name="historyType">
  <xs:sequence>
    <xs:element name="year_built"
     type="yearType"/>
    <xs:element name="year_destroyed"
     type="yearType" minOccurs="0"/>
    <xs:element name="how_destroyed"
     type="destrType" minOccurs="0"/>
    <xs:element name="story"
     type="storyType"/>
  </xs:sequence>
</xs:complexType>
```

Figure11.28

Deriving Complex Types from Existing Complex Types [2/2]

- Deriving Complex Types from Existing Complex Types
  - New complex types derived using restrictions must be valid subsets of the existing complex type

[xsd]

```
<xs:complexType name="newHistoryType">

  <xs:complexContent>

    <xs:extension base="historyType">

      <xs:sequence>

        <xs:element name="who_built"
         type="xs:string"/>

      </xs:sequence>

    </xs:extension>

  </xs:complexContent>

</xs:complexType>
```

Figure11.29

[xsd]

```
<xs:complexType name="newHistoryType">

  <xs:complexContent>

    <xs:restriction base="historyType">

      <xs:sequence>

        <xs:element name="year_built“
          type="yearType"/>

        <xs:element name="year_destroyed“
          type="yearType"/>

        <xs:element name="how_destroyed"
          type="destrType" fixed="fire"/>

        <xs:element name="story"
          type="storyType"/>

      </xs:sequence>

    </xs:restriction>

  </xs:complexContent>

</xs:complexType
```

Figure11.30

Referencing Globally Defined Elements

- Globally defined elements
  - A child of the xs:schema element
  - To be used in the XML Schema document
    - It must be called or *referenced*
- Locally declared elements
  - Automatically referenced by the parent definition in which they appear

[xsd]

```
<xs:schema xmlns:xs="http://
  www.w3.org/2001/XMLSchema">

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

Figure11.31

[xsd]

```
<xs:complexType name="wonderType">
  <xs:sequence>
    <xs:element ref="name"/>
    <xs:element name="location"
      type="xs:string"/>
    <xs:element name="height"
      type="heightType"/>
...
```

[xml]

```
<wonder>
  <name language="English">
    Colossus of Rhodes</name>
  <location>Rhodes, Greece</location>
  <height units="feet">107</height>
...
```

Figure11.32

Figure11.33

Controlling How Many [1/2]

- To specify the minimum number of occurrences
  - **minOccurs="n"**
    - n indicates the fewest number of times the element, sequence, unordered list, or set of choices may occur for the XML document to be considered valid
  - **maxOccurs="n"**
    - n indicates the maximum number of times the element, sequence, unordered list, or set of choices may occur for the XML document to be considered valid

[xsd]

```
<xs:complexType name="wonderType">
  <xs:sequence>
...
    <xs:element name="contributor">
      <xs:complexType>
        <xs:sequence>
          <xs:element ref="name"
            minOccurs="0"
            maxOccurs="unbounded"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
```

Figure11.34

[xsd]

```
<xs:element name="contributor">
  <xs:complexType>
    <xs:choice minOccurs="1"
       maxOccurs="4">
      <xs:element ref="name"/>
      <xs:element name="organization"
         type="xs:string"/>
    </xs:choice>
  </xs:complexType>
</xs:element>
```

Figure11.35

Controlling How Many [2/2]

- An element must appear exactly one time in a valid XML document
  - Unless specified by either of these occurrence attributes

- ***minOccurs*** attribute must be a non-negative integer
- ***maxOccurs*** attribute can be any non-negative integer
  - The word ***unbounded***
    - the element can appear any number of times

- The ***minOccurs*** and ***maxOccurs*** attributes cannot be used when defining an element globally
  - Local references to global elements
  - Locally defined elements

Defining Named Model Groups

- Group the elements together
  - To make it easier to refer to them all at once

- Named Model Groups
  - May only be defined at the top-level of a schema
  - A child element of <xs:schema>
  - It may be referenced as many times as you would like

- Analogous to a parameter entity in DTDs

[xsd]

```
<xs:group name="image_element">
  <xs:sequence>

    <xs:element name="image">
      <xs:complexType>
        <xs:attribute name="file"
          type="xs:anyURI"/>
        <xs:attribute name="w"
          type="xs:positiveInteger"/>
        <xs:attribute name="h"
          type="xs:positiveInteger"/>
      </xs:complexType>
    </xs:element>

    <xs:element name="source"
      type="xs:string"/>

  </xs:sequence>
</xs:group>
```

Figure11.36

Referencing a Named Model Group

- Reference it in other groups
  - **ref="model_group_name"**

[xsd]

```
<xs:element name="main_image">
  <xs:complexType>
    <xs:sequence>
      <xs:group ref="image_element" />
      <xs:element name="caption"
         type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="thumbnail_image">
  <xs:complexType>
    <xs:sequence>
      <xs:group ref="image_element" />
      <xs:element name="frame_border"
         type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Figure11.37

[xml]

```
<main_image>
  <image file="colossus.jpg"
     w="528" h="349" />
  <source>
    Greek Historical Archives</source>
  <caption>Part of a series of the
    Seven Wonders of the World,
    engraved by Marten Heemskerk.
  </caption>
</main_image>

<thumbnail_image>
  <image file="colossus_tn.jpg"
     w="80" h="120" />
  <source>
    Greek Historical Archives</source>
  <frame_border>Blue</frame_border>
</thumbnail_image>
```

Figure11.38

Defining Attributes

- Defining Attributes
  - To use a base or named simple type
    - type="simple_type"
  - To use an anonymous simple type
    - <xs:simpleType>
    - <xs:restriction>
  - To use a globally defined attribute
    - ref="label"
- Attributes must be defined at the very end of the complex type
  - After all the elements in the complex type have been defined

[xsd]

```
<xs:complexType name="sourceType">

  <xs:attribute name="sectionid"
     type="xs:positiveInteger"/>

  <xs:attribute name="newspaperid">
    <xs:simpleType>
      <xs:restriction
         base="xs:positiveInteger">
        <xs:pattern value="\d{4}"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:attribute>

</xs:complexType>
```

Figure11.39

[xml]

```
<source sectionid="101"
   newspaperid="21"/>
```

Figure11.40

Requiring an Attribute

- Unless you specify otherwise, attribute is always optional
  - It may appear or be absent from a valid XML document

- You can insist that an attribute be present (or not)
  - use ="required"
    - The attribute must appear
  - use ="prohibited"
    - The attribute is not present
  - use="optional"
    - Default condition, it's unnecessary

[xsd]

```
<xs:complexType name="sourceType">

  <xs:attribute name="sectionid"
     type="xs:positiveInteger"
     use="required"/>

  <xs:attribute name="newspaperid">
    <xs:simpleType>
      <xs:restriction
         base="xs:positiveInteger">
        <xs:pattern value="\d{4}"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:attribute>

</xs:complexType>
```

Figure11.41

[xml]

```
<source sectionid="141"
   newspaperid="9999"/>
```

Figure11.42-a

[xml]

```
<source sectionid="2"/>
```

Figure11.42-b

Predefining an Attribute's Content [1/2]

- Predefine an attribute's content
  - Fixed value
  - Default value

[xsd]

```
<xs:attribute name="sectionid"
   type="xs:positiveInteger"/>
<xs:attribute name="newspaperid"
   type="xs:positiveInteger"
   fixed="21"/>
```

Figure11.43

[xml]

```
<source sectionid="101"
   newspaperid="21"/>
```

[xml]

```
<source sectionid="101"></source>
```

[xml]

```
<source newspaperid="64"/>
```

Figure11.44

[xsd]

```
<xs:attribute name="sectionid"
   type="xs:positiveInteger"/>
<xs:attribute name="newspaperid"
   type="xs:positiveInteger"
   default="21"/>
```

Figure11.45

[xml]

```
<source sectionid="101"
   newspaperid="21"/>
```

[xml]

```
<source sectionid="101"></source>
```

[xml]

```
<source newspaperid="64"/>
```

Figure11.46

Predefining an Attribute's Content [2/2]

- 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

- The *default* attribute's value
  - is set to the default value, if the attribute is omitted from the XML document
  - 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

Defining Attribute Groups

- Attribute groups
  - It's more efficient to define an attribute group
    - And then refer to the attributes all at once
  - May only be defined at top-level of a schema
    - A child element of <xs:schema>
    - It may be referenced as many times as you like
  - Can contain references to other attribute groups

[xsd]

```
<xs:attributeGroup name="imageAttrs">

  <xs:attribute name="file"
     type="xs:anyURI" use="required"/>

  <xs:attribute name="w"
     type="xs:positiveInteger"
     use="required"/>

  <xs:attribute name="h"
     type="xs:positiveInteger"
     use="required"/>

</xs:attributeGroup>
```

Figure11.47

Referencing Attribute Groups

- Reference an attribute group
  - Attributes and attribute groups must be defined after all other elements have been defined
  - Attribute groups are analogous to parameter entities in DTDs
    - They are limited to representing only collections of attributes

[xsd]

```
<xs:complexType name="imageType">
  <xs:attributeGroup ref="imageAttrs"/>
</xs:complexType>

<xs:complexType name="videoType">
  <xs:attributeGroup ref="imageAttrs"/>
  <xs:attribute name="format"
     type="xs:string"/>
</xs:complexType>
```

Figure11.48

[xml]

```
<main_image file="colossus.jpg"
   w="528" h="349"/>

<main_video file="colossus.mov"
   w="320" h="240"
   format="quicktime"/>
```

Figure11.49

Local and Global Definitions [1/2]

- Globally defined element
  - Its scope is anywhere in the entire schema
  - is defined as a child of the <xs:schema> element
  - Must be explicitly ***referenced***
  - Can be reused in XML document

- Locally defined element
  - Its scope is within its parent element only
  - is defined as the child of some other element
  - Automatically become part of an XML document
- One of the benefits of using locally defined elements
  - Element's scope is isolated
    - The element's name and definition cannot conflict with other elements
      - in the same XML Schema using the same name
  - In a DTD, every element is declared globally

Local and Global Definitions [2/2]

```
<xs:element name="name">
 <xs:complexType>
  <xs:simpleContent>
   <xs:extension
     base="xs:string">
    <xs:attribute name="language"
      type="xs:string"
      use="required">
   </xs:extension>
  </xs:simpleContent>
 </xs:complexType>
</xs:element>
```

Figure11.50

name elements, children of contributor would be invalid

```
<xs:complexType name="wonderType">
 <xs:sequence>
  <xs:element name="name">
   <xs:complexType>
    <xs:simpleContent>
     <xs:extension base="xs:string">
      <xs:attribute name="language"
        type="xs:string"
        use="required"/>
     </xs:extension>
    </xs:simpleContent>
   </xs:complexType>
  </xs:element>
...
 <xs:element name="contributor">
  <xs:complexType>
   <xs:sequence>
    <xs:element name="name" type="xs:string"
      minOccurs="0" maxOccurs="unbounded"/>
   </xs:sequence>
  </xs:complexType>
 </xs:element>
...
</complexType>
```

each name **elements** defined locally

Figure11.51