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

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

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

```
<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>
```
Anonymous complex types: Complex type with no name

[xsd: anonymous complex type definition]

```xml
<x:s:element name="year_built">
  <x:s:complexType>
    <x:s:simpleContent>
      <x:s:extension base="x:s:positiveInteger">
        <x:s:attribute name="era" type="x:s:string"/>
      </x:s:extension>
    </x:s:simpleContent>
  </x:s:complexType>
</x:s:element>
```

[xml: valid XML element]

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

Figure 11.1
Anonymous complex types: Complex type with no name

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

[xml: valid XML element]

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

Figure 11.2

Figure 11.6
Anonymous complex types: Complex type with no name

[xsd: anonymous complex type definition]

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

[xml: valid XML element]

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

Figure 11.2

Figure 11.7
Anonymous complex types: Complex type with no name

[xsd : anonymous complex type definition]

```xml
<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>
```

[xml : valid XML element]

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

Figure 11.8

Figure 11.9
Deriving Named Complex Types

- Named complex type: It can be reused

**[xsd : definition of named complex type]**

```xml
<xsd:complexType name="yearType">
    <xsd:simpleContent>
        <xsd:extension base="xsd:positiveInteger">
            <xsd:attribute name="era" type="xsd:string"/>
        </xsd:extension>
    </xsd:simpleContent>
</xsd:complexType>
```

**[xsd : use the new type by name]**

```xml
<xsd:complexType name="historyTpe">
    <xsd:sequence>
        <xsd:element name="year_built" type="xsd:yearType"/>
        <xsd:element name="year_destroyed" type="xsd:yearType"/>
    </xsd:sequence>
</xsd:complexType>
```

**[xml : valid XML instance]**

```xml
<year_built era="BC">282</year_built>
<year_destroyed era="BC">226</year_destroyed>
```
Defining Complex Types That Contain Child Elements [1/2]

- **Element only type**

```xml
<xsd:element name="ancient_wonders">
  <xsd:complexType>
    <xsd:complexContent>
      <xsd:restriction base="xsd:anyType">
        <xsd:sequence>
          <xsd:element name="wonder" type="wonderType"/>
        </xsd:sequence>
      </xsd:restriction>
    </xsd:complexContent>
  </xsd:complexType>
</xsd:element>
```

```xml
<xsd:element name="ancient_wonders">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="wonder" type="wonderType"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```
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
Order of child elements

- Child elements must appear in an XML document

- `<xs:sequence>` is basically equivalent to the comma (,) in DTD

Figure 11.15
Allowing Child Elements to Appear in Any Order

- In any order

[xsd]

```xml
<xsd:complexType name="historyType">
  <xsd:all>
    <xsd:element name="year_built" type="yearType"/>
    <xsd:element name="year_destroyed" type="yearType"/>
    <xsd:element name="how_destroyed" type="destrType"/>
    <xsd:element name="story" type="storyType"/>
  </xsd:all>
</xsd:complexType>
```

[xml]

```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>
```

Figure 11.16

```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>
  <how_destroyed>earthquake</how_destroyed>
</history>
```

Figure 11.17-a

```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 ...
  <year_destroyed era="BC">226</year_destroyed>
  <how_destroyed>earthquake</how_destroyed>
</history>
```

Figure 11.17-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

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

Figure 11.19-a

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

Figure 11.19-b

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

Figure 11.18
Creating a Set of Choices [2/2]

- Default condition
  - \textit{minOccurs} and \textit{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 \texttt{<xs:choice>} element is basically equivalent to the \texttt{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

```xml
<year_built era="BC">282</year_built>
```
Figure 11.21

```xml
<year_built era="BC">long ago</year_built>
```
Figure 11.22

```xml
<xs:complexType name="yearType">
  <xs:simpleContent>
    <xs:extension base="xs:positiveInteger">
      <xs:attribute name="era" type="xs:string"/>
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>
```
Figure 11.20
Defining Empty Elements

- To define an “empty element” complex type

```xml
<xs:complexType name="sourceType">
  <xs:attribute name="sectionid" type="xs:positiveInteger"/>
  <xs:attribute name="newspaperid" type="xs:positiveInteger"/>
</xs:complexType>
```

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

---

```xml
<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>
```

---

```xml
<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>
```

---

Figure 11.23

Figure 11.24

Figure 11.25
Defining Elements with Mixed Content

- Mixed content type
  - Contain both child elements and text

[xsd]

```xml
<xs:complexType name="story" mixed="true">
  <xs:sequence>
    <xs:element name="para" maxOccurs="unbounded">
      <xs:complexType/>
    </xs:element>
  </xs:sequence>
</xs:complexType>
```

[xml]

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

Figure 11.28
Deriving Complex Types from Existing Complex Types

- New complex types derived using restrictions must be valid subsets of the existing complex type

```xml
<xsd:complexType name="newHistoryType">
  <xsd:complexContent>
    <xsd:extension base="historyType">
      <xsd:sequence>
        <xsd:element name="who_built" type="xsd:string"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```

```xml
<xsd:complexType name="newHistoryType">
  <xsd:complexContent>
    <xsd:restriction base="historyType">
      <xsd:sequence>
        <xsd:element name="year_built" type="yearType"/>
        <xsd:element name="year_destroyed" type="yearType"/>
        <xsd:element name="how_destroyed" type="destrType" fixed="fire"/>
        <xsd:element name="story" type="storyType"/>
      </xsd:sequence>
    </xsd:restriction>
  </xsd:complexContent>
</xsd:complexType>
```
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

```xml
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="name">
    <xsd:complexType>
      <xsd:simpleContent>
        <xsd:extension base="xsd:string">
          <xsd:attribute name="language" type="xsd:string"/>
        </xsd:extension>
      </xsd:simpleContent>
    </xsd:complexType>
  </xsd:element>

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

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

Figure 11.31

Figure 11.32

Figure 11.33
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

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

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

Figure 11.34

Figure 11.35
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]

```xml
<xsd:group name="image_element">
  <xsd:sequence>
    <xsd:element name="image">
      <xsd:complexType>
        <xsd:attribute name="file" type="xsd:anyURI"/>
        <xsd:attribute name="w" type="xsd:positiveInteger"/>
        <xsd:attribute name="h" type="xsd:positiveInteger"/>
      </xsd:complexType>
    </xsd:element>
    <xsd:element name="source" type="xsd:string"/>
  </xsd:sequence>
</xsd:group>
```

Figure11.36
Referencing a Named Model Group

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

**[xsd]**

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

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

**[xml]**

```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>
```

Figure11.37

Figure11.38
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
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

```xml
<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>
```

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

Figure 11.42-a

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

Figure 11.42-b

```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>
```

Figure 11.41
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"/>
```

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

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

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

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

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

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

Figure 11.43

Figure 11.44

Figure 11.45

Figure 11.46
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

```xml
<xsd:attributeGroup name="imageAttrs">
  <xsd:attribute name="file" type="xs:anyURI" use="required"/>
  <xsd:attribute name="w" type="xs:positiveInteger" use="required"/>
  <xsd:attribute name="h" type="xs:positiveInteger" use="required"/>
</xsd:attributeGroup>
```

Figure 11.47
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

```
<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>
```

```
<main_image file="colossus.jpg" w="528" h="349"/>
<main_video file="colossus.mov" w="320" h="240" format="quicktime"/>
```

Figure 11.48

Figure 11.49
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]

Figure11.50

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

name elements, children of contributor would be invalid

Figure11.51

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

each name elements defined locally