A Brief Tutorial on XML Schema

Chin-Lung Chang
Institute of Information Science
Academia Sinica, Taipei, Taiwan

outline

1. Introduction
2. Namespace
3. Declaration
4. Simple Type
5. Complex Type
6. Substitution group
7. Identity constraint
8. Schema Composition

Introduction

Why XML Schema?

The purposes of XML DTD:

- Define a document vocabulary and structure
- Validate document instances

Some shortcomings of XML DTD

- Too few data types (and only for attribute values only)
- Not in XML syntax
- Do not support XML namespace

XML Schema Requirements

- Support XML Namespace
- More primitive data types
- Extensibility and modularity
- Conformance
  - Use XML syntax in the definition of XML schema documents

Terminology

- XML Schema
- XML Schema Definition Language (XSDL)
  - The W3C XML Recommendation, Part 0-2.
- XML Schema document
  - An XML schema defined by using XSDL.
- XML Schema instance
  - An XML document conforming to some XML Schema document.

XSDL

- Major Features:
  - Rich collection of primitive date types
  - Namespace support
  - The ability to define and refine data types
  - Local content model for XML elements
  - Reusability

A Complete Example

- An XML schema document

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="product" type="productType"/>
  <xsd:complexType name="productType">
      <xsd:sequence>
          <xsd:element name="number" type="xsd:integer"/>
          <xsd:element name="date" type="xsd:date"/>
      </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

- An XML schema instance

```
<product>
   <number>557</number>
   <date>2001-04-02</date>
</product>
```

Namespace

Why namespace?

- A book record

```
<book>
    <name>XML tutorial</name>
</book>
```

- An author record

```
<author>
    <name>Chin-Lung Chang</name>
<author>
```

- Combining book and author records? A conflict of element name!

```
<book>
    <book_name>XML tutorial</book_name>
    <author_name>Chin-Lung Chang</author_name>
</book>
```

XML Namespace

- Namespace: A collection of element and attribute names
  – The namespace is identified by a URI.
  – Two-part naming convention:
    - The prefix name
    - The URI of the XML Namespace

- xmlns:foo="http://www.foo.org/"

Declaring Namespace

- Namespace are declared using a special attribute that starts with the xmlns attribute name.
- It is not possible to associate a prefix to an empty URI string. Ex. xmlns:prod=“”
- An example:

```
<prod:product xmlns:prod=“http://example.org/prod”>
   <prod:number>557</prod:number>
   <prod:size system=“US-DRESS”>10</prod:size>
</prod:product>
```

The Default Namespace

- It is used to associate unprefixed element names to a namespace.
- <order xmlns="http://example.org/ord">
- A default namespace declaration does not apply to attributes.
- A default namespace declaration may have an empty URI string.
  - xmlns=""
  - This means that unprefixed element names are not in any namespace

Default Namespace: An Example

- Attribute “system” is not in any namespace.

```
<order xmlns=“http://example.org/ord”
           xmlns:prod=“http://example.org/prod”>
  <number>12345</number>
  <items>
    <prod:number>557</prod:number>
    <prod:size system=“US-DRESS”>10</prod:size>
  </items>
</order>
```

Terminology about Namespace

- Qualified name
  - Name qualified with a namespace
- Unqualified name
  - Name that is not in any namespace
- Prefixed name
  - Name that contain a namespace prefix
- Unprefixed name
  - Name that does not contain a prefix

Scope of A Namespace

- Namespace declarations, including default namespace declarations, can appear in any start-tag in the document.

```
<order xmlns="http://example.org/ord">
  <number>123ABBCC123</number>
  <items>
      <prod:product xmlns:prod="http://example.org/prod">
            <prod:number>557</prod:number>
            <prod:size system="US-DRESS">10</prod:size>
       </prod:product>
       <prod:product> …..</prod:product>  ←  Invalided
    </items>
</order>
```

Overriding A Namespace Declaration

- If a namespace declaration appears within the scope of another namespace declaration with the same prefix, it overrides it. (including default namespace)

```
<order xmlns=http://example.org/ord
          xmlns:prod="http://example.org/prod">
 <number>123ABBCC123</number>
 <items>
     <prod:product xmlns:prod="http://example.org/prod2">
          <prod:number>557</prod:number>
          <prod:size system="US-DRESS">10</prod:size>
      </prod:product>
   </items>
</order>
```

Target Namespace

- XSDL allows a schema document to define ONE namespace, known as its target namespace
  - A schema document cannot have more than one target namespace.
  - Elements defined in the schema document will be referred to by the target namespace.
- Every component declared or defined by a global declaration is associated with that target namespace.
- Local declarations may or may not use the target namespace.

Target Namespace: An Example

```
<xsd:schema xmlns:xsd=“http://www.w3.org/2001/XMLSchema”
   xmlns=“http://example.org/prod”  targetNamespace=“http://
   example.org/prod”>

  <xsd:element name=“number” type=“xsd:integer”/>
  <xsd:element name=“size” type=“SizeType”/>

  <xsd:simpleType name=“SizeType”>
       <--……-->
  </xsd:simpleType>

</xsd:schema>
```

Declaration

Declaration *v.s.* Definition

- *Declarations* – enable element and attributes with specific names and types to appear in document instance
  - Element declaration
  - Attribute declaration

- *Definitions* - create a new type
  - simple type definitions
  - complex type definitions
  - attribute group, model group definitions

Global Element Declaration

```
<xsd:schema xmlns:xsd=“http://www.w3.org/2001/XMLSchema”
                 xmlns=“http://example.org/prod”
                 targetNamespace="http://example.org/prod">
   <xsd:element name="name" type="xsd:string"/>
   <xsd:element name="size" type="xsd:integer"/>
   <xsd:complexType name="ProductType">
   <xsd:sequence>
      <xsd:element ref="name"/>
      <xsd:element ref="size" minOccurs="0"/>
   </xsd:sequence>
   </xsd:complexType>
</xsd:schema>
```

Local Element Declaration

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
                    xmlns="http://example.org/prod"
                    targetNamespace="http://example.org/prod">
    <xsd:complexType name="ProductType">
        <xsd:sequence>
            <xsd:element name="name" type="xsd:string"/>
            <xsd:element name="size" type="xsd:integer" minOccurs="0"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:schema>
```

- The appearance of the *size* element is optional as the *minOccurs* attribute has value 0
  - minOccurs and maxOccurs attributes indicate the minimum and maximum number of times an element may appear.
  - The default value of minOccurs and maxOccurs is 1

Default in Element Declaration

- The default value will be filled in if the element is empty.

```
<xsd:element name="product">
   <xsd:complexType>
      <xsd:choice minOccurs="0“ maxOccurs="unbounded">
          <xsd:element name="name" type="xsd:string" default="N/A"/>
       <xsd:element name="size“ type="xsd:integer“ default="12"/>
      </xsd:choice>
   </xsd:complexType>
</xsd:element>
```

```
<product>             <product>
   <name/>               <name/>    ← no default value filled in
   <size/>               <size>12</size>
</product>            </product>
```

Global Attribute Declaration

```
<xsd:schema xmlns:xsd=“http://www.w3.org/2001/XMLSchema”
                     xmlns="http://example.org/prod"
                     targetNamespace="http://example.org/prod">

    <xsd:attribute name="system" type="xsd:string"/>
    <xsd:attribute name="dim" type="xsd:integer"/>

    <xsd:complexType name="SizeType">
       <xsd:attribute ref="system" use="required"/>
       <xsd:attribute ref="dim"/>
    </xsd:complexType>
</xsd:schema>

<size prod:system=“US” prod:dim=“1”/>
```

Local Attribute Declarations

```
<xsd:schema xmlns:xsd=“http://www.w3.org/2001/XMLSchema”
                  xmlns=“http://example.org/prod”
                  targetNamespace="http://example.org/prod">
   <xsd:complexType name="SizeType">
      <xsd:attributename="system“ type="xsd:string“ use="required“/>
      <xsd:attribute name="dim" type="xsd:integer"/>
   </xsd:complexType>
</xsd:schema>
```

- Note that the *system* attribute is *required*.
  - The use attribute indicates whether the attribute is required, optional or even prohibited
  - The default value of use is optional

Default in Attribute Declaration

- A default value is filled in if the attribute is absent from the element.

```
<xsd:element name="size">
    <xsd:complexType>
        <xsd:attribute name="dim“ type="xsd:integer" default="1"/>
    </xsd:complexType>
</xsd:element>
```

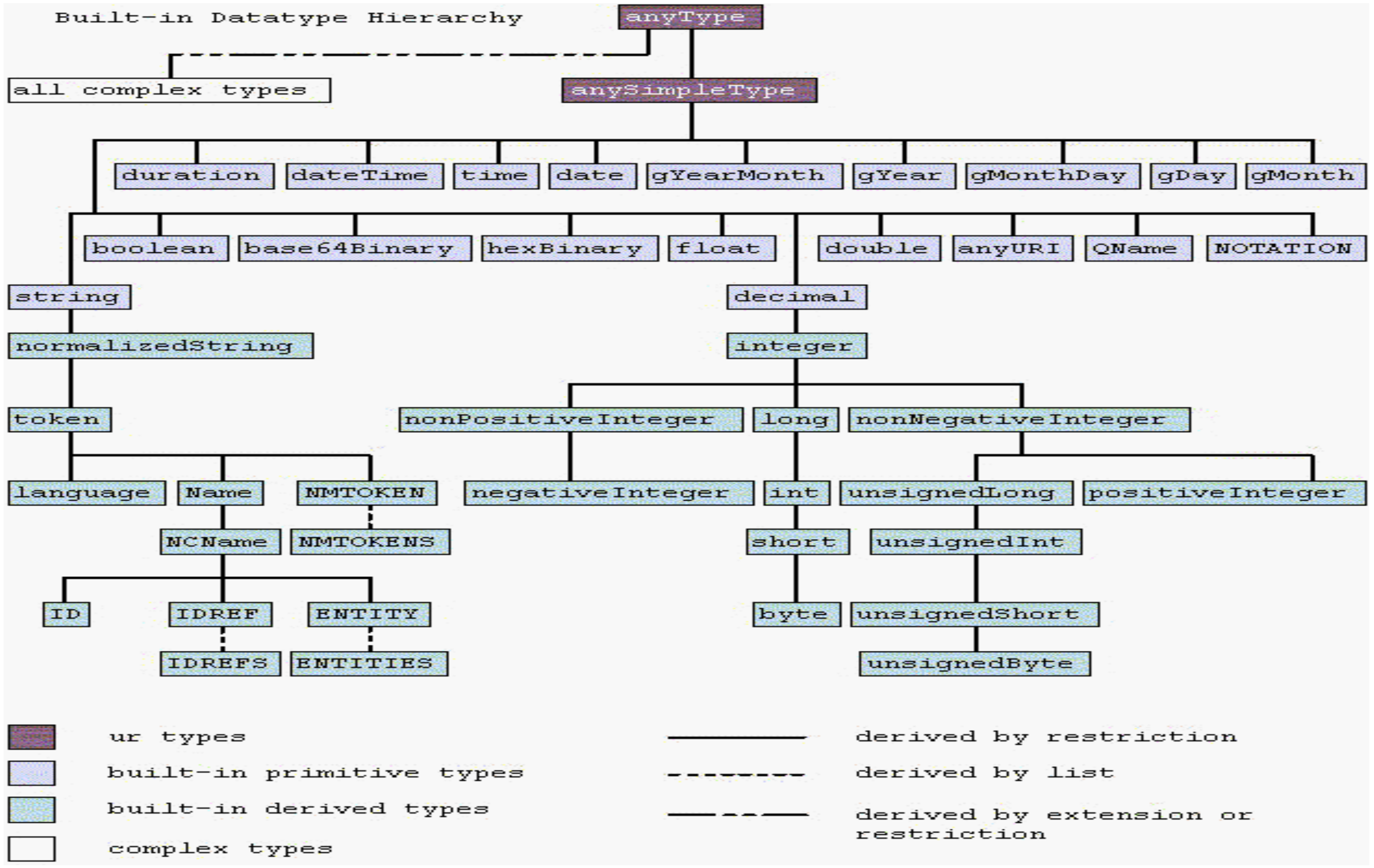
```
<size/>          → <size dim=“1”/>
```

Simple Type

3 Varieties

- Atomic Type
  - The value of atomic type is indivisible.
- List Type
  - Comprise of sequences of atomic type values.
- Union Type
  - The value of Union type contains one or more instance of one type drawn from the union of atomic and list types.

Primitive Type Hierarchy



Simple Type Definition: An Example

```
<xsd:simpleType name="DressSizeType">
   <xsd:restriction base="xsd:integer">
      <xsd:minInclusive value="2"/>
      <xsd:maxInclusive value="18"/>
   </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="MediumDressSizeType">
   <xsd:restriction base="DressSizeType">
      <xsd:minInclusive value="8"/>
      <xsd:maxInclusive value="12"/>
   </xsd:restriction>
</xsd:simpleType>
```

Available facets

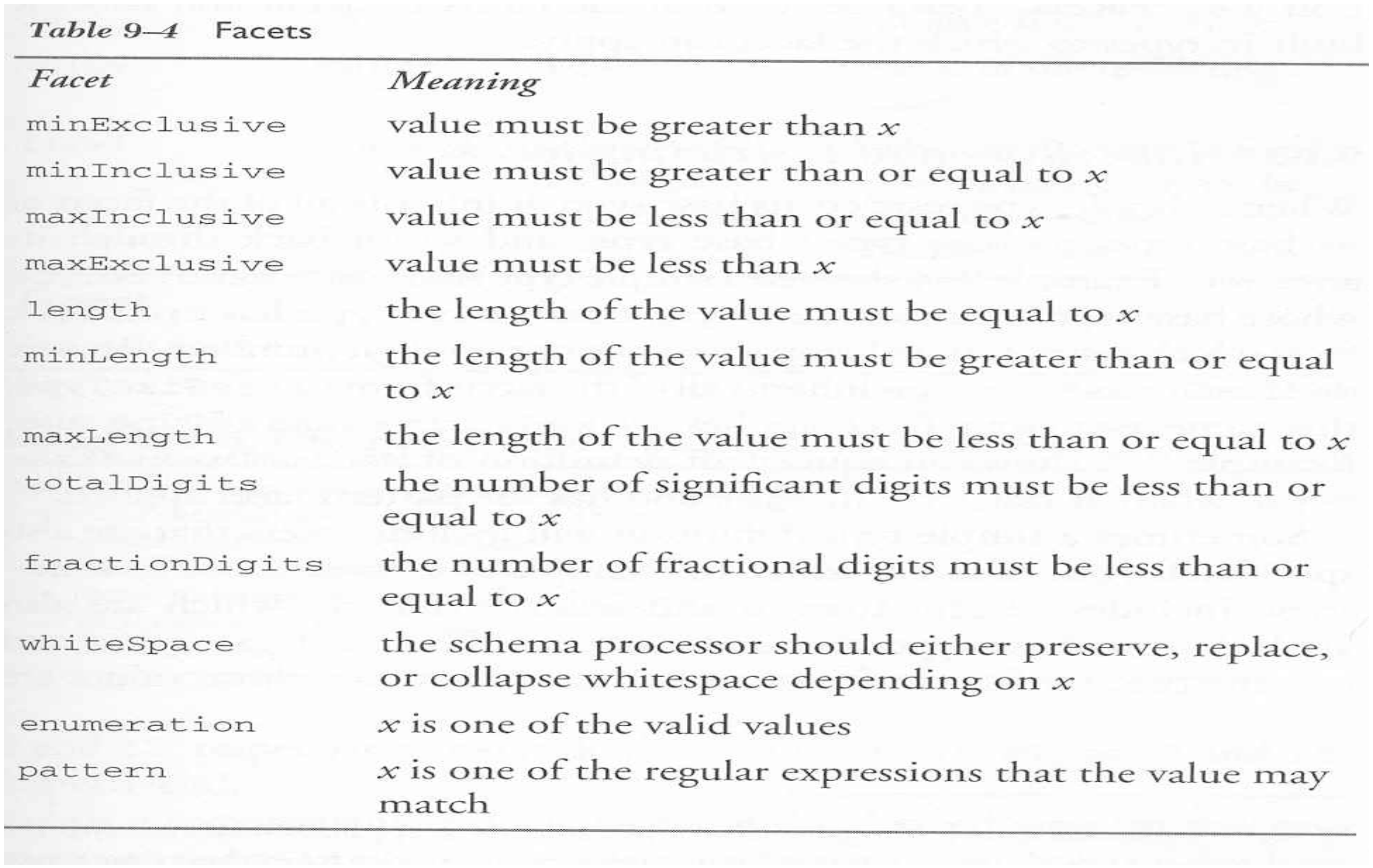


Table 9–4 Facets

Facet	*Meaning*
`minExclusive`	value must be greater than x
`minInclusive`	value must be greater than or equal to x
`maxInclusive`	value must be less than or equal to x
`maxExclusive`	value must be less than x
`length`	the length of the value must be equal to x
`minLength`	the length of the value must be greater than or equal to x
`maxLength`	the length of the value must be less than or equal to x
`totalDigits`	the number of significant digits must be less than or equal to x
`fractionDigits`	the number of fractional digits must be less than or equal to x
`whiteSpace`	the schema processor should either preserve, replace, or collapse whitespace depending on x
`enumeration`	x is one of the valid values
`pattern`	x is one of the regular expressions that the value may match

Restricting Element Content

- The facets of the derived type must be more restrictive than those of the base type.

- Within a *restriction* element in a type definition
  - You can specify any of the facets, in any order.
  - The only facets that may appear more than once in the same restriction are pattern and enumeration.

Illegal Restriction

```
<xsd:simpleType name="DressSizeType">
    <xsd:restriction base="xsd:integer">
        <xsd:minInclusive value="2"/>
        <xsd:maxInclusive value="18"/>
    </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="SmallDressSizeType">
    <xsd:restriction base="DressSizeType">
        <xsd:minInclusive value="0"/>
        <xsd:maxInclusive value="6"/>
    </xsd:restriction>
</xsd:simpleType>
```

Enumeration

- Allows you to specify a distinct set of valid values for its base type.
  - If type of size element is xsd:integer, the valid value could be

    <size>2</size>

    <size>02</size>

- Basic Rules
  - Can appear multiple times in a single restriction.
  - Each enumeration value must be unique, and must be valid for base type.
  - Enumeration facet can be applied to any type except boolean

Enumeration: Example

```
<xsd:simpleType name="SMLXSizeType">
    <xsd:restriction base="xsd:token">
        <xsd:enumeration value="small"/>
        <xsd:enumeration value="medium"/>
        <xsd:enumeration value="large"/>
    </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="SMLSizeType">
    <xsd:restriction base="SMLXSizeType">
        <xsd:enumeration value="small"/>
        <xsd:enumeration value="medium"/>
    </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="XSMLXSizeType">
    <xsd:restriction base="SMLXSizeType">
        <xsd:enumeration value="extra small"/>
        <xsd:enumeration value="small"/>
        <xsd:enumeration value="medium"/>
    </xsd:restriction>
</xsd:simpleType>
```

Fixed Facets

- value of the fixed facets will not be changed in further restriction.

```
<xsd:simpleType name="DressSizeType">
   <xsd:restriction base="xsd:integer">
      <xsd:minInclusive value="2" fixed="true"/>
      <xsd:maxInclusive value="18" fixed="true"/>
   </xsd:restriction>
</xsd:simpleType>
```

Preventing Simple Type Derivation

- XSDL allows you to prevent derivation of other types from your type definition.
  – Just specify the final attribute in your type definition.

```
<xsd:simpleType name="DressSizeType" final="restriction">
    <xsd:restriction base="xsd:integer">
        <xsd:minInclusive value="2"/>
        <xsd:maxInclusive value="18"/>
    </xsd:restriction>
</xsd:simpleType>
```

final

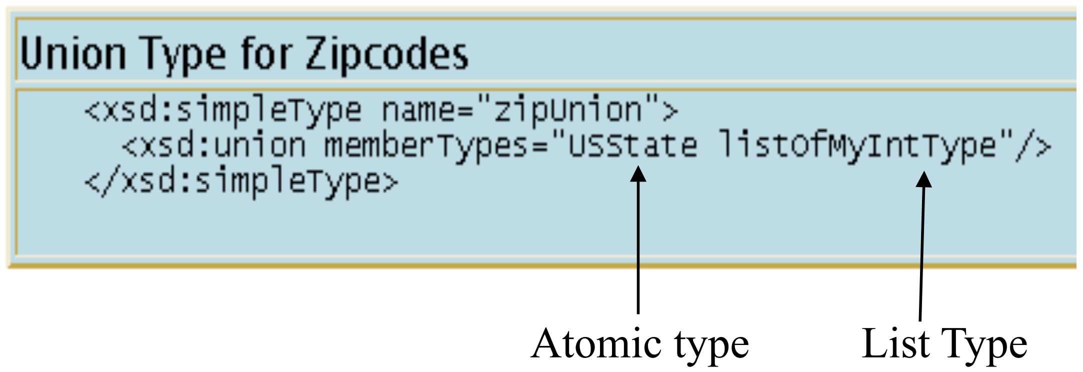
- Values
  - “#all”, prevent any other type from extending or restricting your type
  - “extension”, prevent extending your type
  - “restriction”,prevent restricting your type
  - “extension restriction” or “restriction extension”, equal to “#all”
  - “”, there are no restriction

- finalDefault of schema element

```
<xsd:complexType name="ProductType" final="#all">
    <xsd:sequence>
        <xsd:element name="number" type="ProdNumType"/>
        <xsd:element name="name" type="xsd:string"/>
    </xsd:sequence>
</xsd:complexType>
```

Union Types

- Union types allow a value to conform to any one of several different simple types.
- You must union at least one simple type, and there is no limit of how many member types may exist.



Union Type for Zipcodes

```
<xsd:simpleType name="zipUnion">
  <xsd:union memberTypes="USState listofMyIntType"/>
</xsd:simpleType>
```

Examples:
<zips>CA</zips>
<zips>32308 32306</zips>

Union Type: An Example

```
<xsd:simpleType name="SizeType">
    <xsd:union>
        <xsd:simpleType>
            <xsd:restriction base="xsd:integer">
                <xsd:minInclusive value="2"/>
                <xsd:maxInclusive value="18"/>
            </xsd:restriction>
        </xsd:simpleType>
        <xsd:simpleType>
            <xsd:restriction base="xsd:token">
                <xsd:enumeration value="small"/>
                <xsd:enumeration value="medium"/>
                <xsd:enumeration value="large"/>
            </xsd:restriction>
        </xsd:simpleType>
    </xsd:union>
</xsd:simpleType>
```

Restricting Union Types

- It is possible to restrict a union type.
- Restriction of a union type always result in a union type.
- Only two facets may be applied to union types: pattern and enumeration.
- These restrictions are considered to be in addition to the restrictions of the individual member types.

```
<xsd:simpleType name="SmallSizeType">
    <xsd:restriction base="SizeType"> ← union type
        <xsd:enumeration value="2"/>
        <xsd:enumeration value="4"/>
        <xsd:enumeration value="6"/>
        <xsd:enumeration value="small"/>
    </xsd:restriction>
</xsd:simpleType>
```

Unions of Unions

```
<xsd:simpleType name="InternationalSizeType">
    <xsd:union memberTypes="SizeType">
        <xsd:simpleType>
            <xsd:restriction base="xsd:integer">
                <xsd:minInclusive value="24"/>
                <xsd:maxInclusive value="54"/>
            </xsd:restriction>
        </xsd:simpleType>
    </xsd:union>
</xsd:simpleType>
```

- Expansion of SizeType

List Types

- List types are whitespace-spearated lists of atomic values.
- A list type is defined by designating another simple type (an atomic or union type) as its item type.
- You can create new list type by derivation from existing atomic types.
- You cannot create list types from existing list types nor from complex type. That is,
  - No list of list, and
  - No list of complexType.

List Type: An Example

```
<xsd:simpleType name="AvailableSizesType">
    <xsd:list itemType="DressSizeType"/>
</xsd:simpleType>

Instance: <availableSizes>10 12 14</availableSizes>

<xsd:simpleType name="AvailableSizesType">
    <xsd:list>
        <xsd:simpleType>
            <xsd:restriction base="xsd:integer">
                    <xsd:minInclusive value="2"/>
                    <xsd:maxInclusive value="18"/>
            </xsd:restriction>
        </xsd:simpleType>
    </xsd:list>
</xsd:simpleType>
```

- Either the itemType attribute or the simpleType child must appear, not both.

Restricting List Types

- A limited number of facets may be applied to list types.
  - Length, minLength, maxLength and enumeration are facets of list types
- These facets have slightly different behavior when applied to a list type, because they apply to the list as a whole, not to the individual items in the list.

The Length Facets

- Length facets length, minLength, and maxLength may be used to restrict list types.
- The length is measured in number of items in the list, not the length of each item.

```
<xsd:simpleType name=“USStateList”>
  <xsd:list itemType=“USState”/>
</xsd:simpleType>

<xsd:simpleType name=“SixUSStates”>
  <xsd:restriction base=“USStateList”>
     <xsd:length value=“6”/>
  </xsd:restriction>
</xsd:simpleType>

<sixStates>PA NY CA FL LA AK</sixStates>
```

The Enumeration Facet

- What if we like to have following instance defined with list type?

```
<size>small</size>
<size>small medium</size>
<size>small medium large</size>
```

- To restrict the values in a list to a specific set

```
<xsd:simpleType name="AvailableSizesType">
    <xsd:restriction>
        <xsd:simpleType>
            <xsd:list itemType="xsd:token"/>
        </xsd:simpleType>
        <xsd:enumeration value="small"/>
        <xsd:enumeration value="medium"/>
        <xsd:enumeration value="large"/>
    </xsd:restriction>
</xsd:simpleType>
```

- Non-valid list definition → <size>small</size> <size>large</size>

The Enumeration Facet

Restriction of base type:

```
<xsd:simpleType name="AvailableSizesType">
        <xsd:list>
                <xsd:simpleType>
                        <xsd:restriction base="xsd:token">
                                <xsd:enumeration value="small"/>
                                <xsd:enumeration value="medium"/>
                                <xsd:enumeration value="large"/>
                        </xsd:restriction>
                </xsd:simpleType>
        </xsd:list>
</xsd:simpleType>
```

Lists and Strings

- Be careful when deriving list types from the string-based types, whose values may contain whitespace.

```
<xsd:simpleType name="AvailableSizesType">
    <xsd:list itemType="SMLXSizeType"/>
</xsd:simpleType>

<xsd:simpleType name="SMLXSizeType">
    <xsd:restriction base="xsd:token">
        <xsd:enumeration value="small"/>
        <xsd:enumeration value="medium"/>
        <xsd:enumeration value="large"/>
        <xsd:enumeration value="extra large"/>
    </xsd:restriction>
</xsd:simpleType>
```

Invalid Instance:

```
<availableSizes>
small
extra large
</availableSizes>
```

More on List Types

- List of list
  - Lists of lists are not legal.
  - The item type of a list type cannot be a list type, nor can it be derived at any level from another list type (for example, a union of a list).
- Lists of unions
  - Each item in the list must simply be valid value of one of the member types of the union type.
  - The only restriction on lists of unions is that the union type cannot have any list types among its member types.

List of Union: An Example

```
<xsd:simpleType name="SizeType">
    <xsd:union memberTypes="DressSizeType SMLXSizeType"/>
</xsd:simpleType>

<xsd:simpleType name="AvailableSizesType">
    <xsd:list itemType="SizeType"/>
</xsd:simpleType>
```

Instance:

```
<availableSizes>10 large 2</availableSizes>
```

Complex Type

Complex Type Definition: Examples

```
<xsd:complexType name="ProductType">
    <xsd:sequence>
        <xsd:elementname="number“ type="ProdNumType"/>
        <xsd:element name="name" type="xsd:string"/>
        <xsd:element name="size" type="SizeType"/>
    </xsd:sequence>
</xsd:complexType>

<xsd:element name="product" type="ProductType"/>
< ------------------------------------------------------- >
<xsd:element name="product">
    <xsd:complexType>
        <xsd:sequence>
                <xsd:element name="number" type="ProdNumType"/>
                <xsd:element name="name" type="xsd:string"/>
                <xsd:element name="size" type="SizeType"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
```

Content Types

- What is content ?
  - The content of an element are the character data and child elements that are between its tags.
- There are four types of content for complex types
  - Simple
  - Element-only
  - Mixed
  - Empty

Simple Content

```
<xsd:complexType name="SizeType">
  <xsd:simpleContent>
    <xsd:extension base="xsd:integer">
        <xsd:attribute name="system" type="xsd:token"/>
    </xsd:extension>
  </xsd:simpleContent>
</xsd:complexType>

<xsd:element name=“size” type=“SizeType”/>
```

Instance:

```
<size system="US-DRESS">10</size>
```

Element-only Content

```
<xsd:complexType name="ProductType">
    <xsd:sequence>
        <xsd:element name="number" type="ProdNumType"/>
        <xsd:element name="name" type="xsd:string"/>
        <xsd:element name="size" type="SizeType"/>
        <xsd:element name="color" type="ColorType"/>
    </xsd:sequence>
</xsd:complexType>

<product>
    <number>557</number>
    <name>Short-Sleeved Linen Blouse</name>
    <size system="US-DRESS">10</size>
    <color value="blue"/>
</product>
```

Mixed Content

```
<xsd:complexType name="LetterType" mixed="true">
    <xsd:sequence>
        <xsd:element name="custName" type="CustNameType"/>
        <xsd:element name="prodName" type="xsd:string"/>
        <xsd:element name="prodSize" type="SizeType"/>
        <!--...-->
    </xsd:sequence>
</xsd:complexType>
```

```
<letter>Dear <custName>Priscilla Walmsley</custName>,
Unfortunately, we are out of stock of the <prodName>Short-Sleeved
Linen Blouse</prodName> in size <prodSize>10</prodSize> that you
ordered...</letter>
```

- The character data that is directly contained in the letter element is not assigned a data type, it is completely unrestricted.

Empty content

- No content model → empty content

```
<xsd:complexType name="ColorType">
    <xsd:attribute name="value" type="ColorValueType"/>
</xsd:complexType>

Instance:
<color value="blue"/>
```

Model Group

- The order and structure of the children of a complex type are known as its "content model".
- Model group allow you to group child element declarations or references together to construct more meaningful content models.
- There are 3 kinds of model groups:
  - Sequence
  - Choice
  - All
- Every complex type has exactly one model group child.

Sequence Groups

```
<xsd:complexType name="ProductType">
   <xsd:sequence>
      <xsd:element name="number" type="ProdNumType"/>
      <xsd:element name="name" type="xsd:string"/>
      <xsd:element name="size" type="SizeType" minOccurs="0"/>
      <xsd:element name="color" type="ColorType" minOccurs="0"/>
   </xsd:sequence>
</xsd:complexType>

<product>
   <number>557</number>
   <name>Short-Sleeved Linen Blouse</name>
   <size system="US-DRESS">10</size>
   <color value="blue"/>
</product>
```

Choice Groups

```
<xsd:complexType name="ItemsType">
    <xsd:choice>
        <xsd:element name="shirt" type="ShirtType"/>
        <xsd:element name="hat" type="HatType"/>
        <xsd:element name="umbrella" type="UmbrellaType"/>
    </xsd:choice>
</xsd:complexType>

<items>
    <shirt>...</shirt>
</items>
<items>
    <hat>...</hat>
</items>
```

More Choice Groups

```
<xsd:complexType name="ItemsType">
    <xsd:choice minOccurs="0" maxOccurs="unbounded">
        <xsd:element name="shirt" type="ShirtType"/>
        <xsd:element name="umbrella" type="UmbrellaType"/>
        <xsd:element name="hat" type="HatType"/>
    </xsd:choice>
</xsd:complexType>

<items>
    <shirt>...</shirt>
    <hat>...</hat>
    <umbrella>...</umbrella>
    <shirt>...</shirt>
    <shirt>...</shirt>
</items>
```

Nesting of Sequence & Choice

```
<xsd:complexType name="ProductType">
    <xsd:sequence>
        <xsd:element name="number" type="ProdNumType"/>
        <xsd:element name="name" type="xsd:string"/>
        <xsd:choice minOccurs="0" maxOccurs="unbounded">
                <xsd:element name="size" type="SizeType"/>
                <xsd:element name="color" type="ColorType"/>
        </xsd:choice>
    </xsd:sequence>
</xsd:complexType>
```

All Groups

```
<xsd:complexType name="ProductType">
    <xsd:all>
        <xsd:element name="number" type="ProdNumType"/>
        <xsd:element name="name" type="xsd:string"/>
        <xsd:element name="size" type="SizeType" minOccurs="0"/>
        <xsd:element name="color" type="ColorType" minOccurs="0"/>
    </xsd:all>
</xsd:complexType>

<product>
    <color value="blue"/>
    <size system="US-DRESS">10</size>
    <number>557</number>
    <name>Short-Sleeved Linen Blouse</name>
</product>
```

Attribute Declarations

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/
   XMLSchema">
   <xsd:attribute  name="effDate“ type="xsd:date"/>
   <xsd:complexType name="ProductType">
      <xsd:sequence>
            <!--...-->
      </xsd:sequence>
      <xsd:attribute ref="effDate"/>
         <xsd:attribute name=“local” type=“xsd:string”/>
   </xsd:complexType>
</xsd:schema>
```

How to Derive Complex Types?

- Restriction
  - Restricting the valid contents of a type.
  - Values of new type is a subset of those of the base type.
  - All values of the restricted type are valid with respect to the base type.
- Extension
  - Adding additional children and/or attributes to a type.

Complex Type Content

- Simple content
  - Has only character data content; no children elements.
- complex content
  - One of three cases: mixed, element-only, or empty.
- A complex type can be derived from another complex type by using a
  - simpleContent element
  - complexContent element

simpleContent

- simpleContent element is used when deriving a complex type from
  - A simple type;
  - Another complex type with simple content.
- How?
  - Add or remove attributes, or
  - Restrict the simple type of the character content (facets).

complexContent

- complexContent element is used when deriving a complex type from
  - Another complex type which itself has complex content model (mixed, element-only, or empty).

- How?
  - Add or remove parts of the content model as well as the attributes.

Extension of Complex Types

Simple Content Extension

```
<xsd:complexType name="SizeType">
    <xsd:simpleContent>
        <xsd:extension base="xsd:integer">
                <xsd:attribute name="system" type="xsd:token"/>
        </xsd:extension>
    </xsd:simpleContent>
</xsd:complexType>

<size system="US-DRESS">10</size>
```

Complex Content Extension

- Add content model of the base type and attributes.

- Example.

  Basic Type:

```
<xsd:complexType name="ProductType">
    <xsd:sequence>
       <xsd:element name="number" type="ProdNumType"/>
       <xsd:element name="name" type="xsd:string"/>
    </xsd:sequence>
</xsd:complexType>
```

Complex Type Extension (Example)

Extended:

```
<xsd:complexType name="ShirtType">
    <xsd:complexContent>
        <xsd:extension base="ProductType">
            <xsd:choice maxOccurs="unbounded">
                <xsd:element name="size" type="SizeType"/>
                <xsd:element name="color" type="ColorType"/>
            </xsd:choice>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
```

The Effective Content Model (Example)

```
<xsd:complexType name="ShirtType">
    <xsd:sequence>
        <xsd:sequence>
            <xsd:element name="number" type="ProdNumType"/>
            <xsd:element name="name" type="xsd:string"/>
        </xsd:sequence>
        <xsd:choice maxOccurs="unbounded">
            <xsd:element name="size" type="SizeType"/>
            <xsd:element name="color" type="ColorType"/>
        </xsd:choice>
    </xsd:sequence>
</xsd:complexType>

<shirt>
    <number>10</number>
    <name>red shirt</name>
    <size>9</size>
<shirt>
```

Attribute Extension

```
<xsd:complexType name="ItemType">
    <xsd:attribute name="id" type="xsd:ID" use="required"/>
    <xsd:attribute ref="xml:lang"/>
</xsd:complexType>

<item id=“item22” xml:lang=“en”/>

<xsd:complexType name="ProductType">
    <xsd:complexContent>
        <xsd:extension base="ItemType">
            <xsd:attribute name="effDate" type="xsd:date"/>
            <xsd:attribute name="lang" type="xsd:language"/>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>

<product id="prod557“ xml:lang="en“ lang="en“ effDate="2001-04-02"/>
```

Restriction of Complex Types

Complex Type Restriction

- Complex type may be restricted by
  - Eliminating or restricting attributes;
  - Subsetting content models.

Simple Content Restriction

```
<xsd:complexType name="SmallSizeType">
    <xsd:simpleContent>
        <xsd:restriction base="SizeType">
                <xsd:minInclusive value="2"/>
                <xsd:maxInclusive value="6"/>
                <xsd:attribute  name="system" type="xsd:token"
                                                    use="required“/>
        </xsd:restriction>
    </xsd:simpleContent>
</xsd:complexType>
```

- The “base” attribute must be a complex type with simple content, not a simple type.

Complex Content Restriction

- Complex content restriction allow you to restrict the content model and/or attributes of a complex type.
- Repeating all of the content model that is desired.
- Rule of validation
  - All instances of the new restricted type must also be valid with respect to the base type.

```
<xsd:complexType name="ProductType">
    <xsd:sequence>
        <xsd:element name="number" type="ProdNumType"/>
        <xsd:element name="name" type="xsd:string"/>
        <xsd:element name="size" type="SizeType" minOccurs="0"/>
        <xsd:element name="color" type="ColorType" minOccurs="0"/>
    </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="RestrictedProductType">
    <xsd:complexContent>
        <xsd:restriction base="ProductType">
                <xsd:sequence>
                        <xsd:element name="number" type="ProdNumType"/>
                        <xsd:element name="name" type="xsd:string"/>
                </xsd:sequence>
        </xsd:restriction>
    </xsd:complexContent>
</xsd:complexType>
```

Restricting Element Declaration

```
<xsd:sequence>
    <xsd:element name="a" maxOccurs="3"/>
    <xsd:element name="b" fixed="bValue"/>
    <xsd:element name="c" type="xsd:string"/>
</xsd:sequence>
<!--Legal restriction:-->
<xsd:sequence>
    <xsd:element name="a" maxOccurs="2"/>
    <xsd:element name="b" fixed="bValue"/>
    <xsd:element name="c" type="xsd:token"/>
</xsd:sequence>
<!--Illegal restriction:-->
<xsd:sequence>
    <xsd:element name="a" maxOccurs="4"/>
    <xsd:element name="b" fixed="newValue"/>
    <xsd:element name="c" type="xsd:integer"/>
</xsd:sequence>
```

Attribute Restriction

- When defining a restriction, you may either restrict or remove attributes of the base type.
- The only attributes that need to appear in the derived type are the ones you want to restrict or remove.
- The legal ways to restrict an attribute
  - Change the attribute's type, as long as the new type is a restriction of the original type.
  - Add, change or remove a default value.
  - Make optional attributes required.
  - Make optional attribute prohibited (remove it).

Attribute Restriction: Examples

```
<xsd:complexType name="BaseType">
    <xsd:attribute name="a" type="xsd:integer"/>
    <xsd:attribute name="b" type="xsd:string"/>
    <xsd:attribute name="c" type="xsd:string" default="c"/>
    <xsd:attribute name=“d" type="xsd:string"/>
    <xsd:attribute name=“e" type="xsd:string"/>
    <xsd:attribute name="x" type="xsd:string"/>
</xsd:complexType>

<xsd:complexType name="DerivedType">
    <xsd:complexContent>
        <xsd:restriction base="BaseType">
                <xsd:attribute name="a" type="xsd:positiveInteger"/>
                <xsd:attribute name="b" type="xsd:string" default="b"/>
                <xsd:attribute name="c" type="xsd:string" default="c2"/>
                <xsd:attribute name=“d" type="xsd:string" use="required"/>
                <xsd:attribute name=“e" type="xsd:string" use="prohibited"/>
        </xsd:restriction>
    </xsd:complexContent>
</xsd:complexType>
```

Prevent Derivation: The *final* Attrribute

- Values
  - “#all”, prevent any other type from extending or restricting your type
  - “extension”, prevent extending your type
  - “restriction”, prevent restricting your type
  - “extension restriction” or “restriction extension”, equal to “#all”
  - “”, there are no restriction

```
<xsd:complexType name="ProductType" final="#all">
    <xsd:sequence>
        <xsd:element name="number" type="ProdNumType"/>
        <xsd:element name="name" type="xsd:string"/>
    </xsd:sequence>
</xsd:complexType>
```

Type Substitution

Type Substitution: Example

```
<xsd:complexType name="ProductType">
    <xsd:sequence>
        <xsd:element name="number" type="ProdNumType"/>
        <xsd:element name="name" type="xsd:string"/>
    </xsd:sequence>
</xsd:complexType>
<xsd:element name="product" type="ProductType"/>

<xsd:complexType name="ShirtType">
    <xsd:complexContent>
        <xsd:extension base="ProductType">
                <xsd:choice maxOccurs="unbounded">
                            <xsd:element name="size" type="SizeType"/>
                            <xsd:element name="color" type="ColorType"/>
                </xsd:choice>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
```

Substitution in Document Instance

```
<items xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <product xsi:type="ShirtType">
        <number>557</number>
        <name>Short-Sleeved Linen Blouse</name>
        <color value="blue"/>
    </product>
    <!--...-->
</items>
```

Controlling Substitution

- You may want to control the substitution of derived types.
  - The attribute block is used to limit the substitution of derived types in instance.
  - The attribute abstract is used to forces the definition of derived types.

block: Example

```
<xsd:complexType name="ProductType" block="extension">
      <xsd:sequence>
           <xsd:element name="number" type="ProdNumType"/>
           <xsd:element name="name" type="xsd:string"/>
      </xsd:sequence>
</xsd:complexType>

<xsd:element name="product" type="ProductType"/>

<xsd:complexType name="ShirtType">
      <xsd:complexContent>
           <xsd:extension base="ProductType">
                     <xsd:choice maxOccurs="unbounded">
                                    <xsd:element name="size" type="SizeType"/>
                                    <xsd:element name="color" type="ColorType"/>
                     </xsd:choice>
           </xsd:extension>
      </xsd:complexContent>
</xsd:complexType>
<xsd:element name="shirt" type="ShirtType"/>
```

```
<!-- Illegal substitution of ShirtType-->
<product xsi:type="ShirtType">
               <number>557</number>
               <name>Short-Sleeved Linen Blouse</name>
               <color value="blue"/>
</product>
```

Blocking Type Substitution in Element Declaration

- An element declaration can also have the block attribute, with the same valid values as for complexType.

```
<xsd:element name="product" type="ProductType“ block=“extension”/>
```

abstract: example

```
<xsd:complexType name="ProductType" abstract="true">
    <xsd:sequence>
        <xsd:element name="number" type="ProdNumType"/>
        <xsd:element name="name" type="xsd:string"/>
    </xsd:sequence>
</xsd:complexType>

<xsd:element name="product" type="ProductType"/>

<xsd:complexType name="ShirtType">
    <xsd:complexContent>
        <xsd:extension base="ProductType">
                <xsd:choice maxOccurs="unbounded">
                        <xsd:element name="size" type="SizeType"/>
                        <xsd:element name="color" type="ColorType"/>
                </xsd:choice>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>

<xsd:element name="shirt" type="ShirtType"/>
```

abstract: instance

```
<product xsi:type="ShirtType">
        <number>557</number>
        <name>Short-Sleeved Linen Blouse</name>
        <color value="blue"/>
</product>

<shirt>
        <number>557</number>
        <name>Short-Sleeved Linen Blouse</name>
        <color value="blue"/>
</shirt>
```

Substitution group

substitution group(1)

- Each substitution group consists of
  - a head
  - One or more members
- Wherever the head element declaration is referenced in a content model, one of the member element declarations may be substituted in place of the head
  - Element substitution

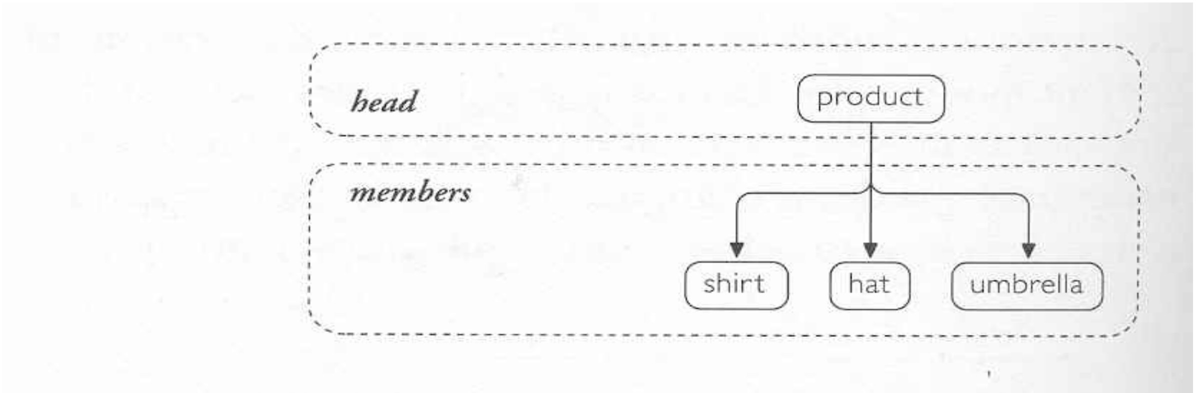
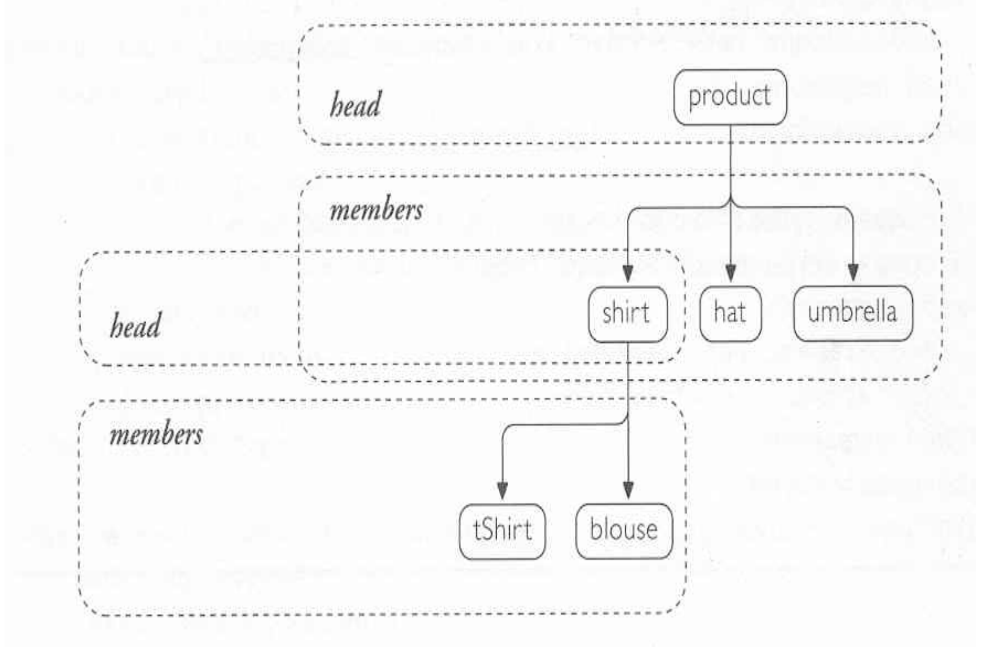


Figure 16–1 Substitution group hierarchy

substitution groups(2)

- Each element declaration can only be a member of one substitution group
- A member of one group may be the head of another group
- tShirt and blouse may substitute for product (or shirt)



gure 16–2 Multi-level substitution group hierarchy

Declaring a substitution group

- Only a global element declaration can be the head of a substitution group
- Members
  - Each of the declaration use the substitutionGroup attribute to indicate that it is substitutable for head
  - Members of a substitution group must be globally declared

Example: head

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:element name="items" type="ItemsType"/>
    <xsd:complexType name="ItemsType">
        <xsd:sequence>
                <xsd:element ref="product" maxOccurs="unbounded"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:element name="product" type="ProductType"/>
    <xsd:complexType name="ProductType">
        <xsd:sequence>
                <xsd:element ref="number"/>
                <xsd:element ref="name"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:schema>
```

Example: member

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

    <xsd:element name="shirt" type="ShirtType“ substitutionGroup="product"/>

    <xsd:complexType name="ShirtType">
       <xsd:complexContent>
           <xsd:extension base="ProductType">
           <xsd:sequence>
              <xsd:element name="size" type="ShirtSizeType"/>
              <xsd:element name="color" type="ColorType"/>
           </xsd:sequence>
           </xsd:extension>
       </xsd:complexContent>
    </xsd:complexType>
    <!--...-->
    <xsd:element name="umbrella" substitutionGroup="product"/>
    <!--...-->
</xsd:schema>
```

Example: instance

```
<items>
    <product>
        <number>999</number>
        <name>Special Seasonal</name>
    </product>
    <shirt>
        <number>557</number>
        <name>Short-Sleeved Linen Blouse</name>
        <size>10</size>
        <color value="blue"/>
    </shirt>
    <hat>
        <number>563</number>
        <name>Ten-Gallon Hat</name>
        <size>L</size>
    </hat>
    <umbrella>
        <number>443</number>
        <name>Deluxe Golf Umbrella</name>
    </umbrella>
</items>
```

Type constraints

- Members of a substitution group must have types that are
  - the same as the type of the head, or
  - derived from type of head by either extension or restriction
- If a substitution group member is declared without a type, it automatically takes on the type of the head of its substitution group.
  - umbrella

If head element’s type is anyType

- All types are derived from anyType
- Therefore the members of the substitution group in this case can have any data type, including simple types

Controlling substitution group

- Three attributes of element declarations control the creation and use of substitutions
  - The final attribute: prevent from defining schemas that use your element declaration as the head of a substitution group. (only for global element declaration)
  - The block attribute: blocking substitution (type substitution or substitution group) in instances
  - The abstract attribute: forcing substitution, declared to serve as the head of a substitution group. (only for global element declaration)

Identity Constraint

An Instance

```
<catalog>
    <department number="021">
        <product effDate="2000-02-27”>
                <number>557</number>
                <name>Short-Sleeved Linen Blouse</name>
                <price currency="USD">29.99</price>
        </product>
        <product effDate="2001-04-02”>
                <number>563</number>
                <name>Ten-Gallon Hat</name>
                <price currency="USD">69.99</price>
        </product>
        <product>
                <number>443</number>
                <name>Deluxe Golf Umbrella</name>
                <price currency="USD">49.99</price>
        </product>
    </department>
</catalog>
```

The *unique* Constraint

```
<xsd:element name="catalog" type="CatalogType">
    <xsd:unique name="prodNumKey">
        <xsd:selector xpath="*/product"/>
        <xsd:field xpath="number"/>
    </xsd:unique>
</xsd:element>

<xsd:element name="catalog" type="CatalogType">
    <xsd:unique name="dateAndProdNumKey">
        <xsd:selector xpath="department/product"/>
        <xsd:field xpath="number"/>
        <xsd:field xpath="@effDate"/>
    </xsd:unique>
</xsd:element>
```

Another Instance

```
<order>
        <number>123ABBCC123</number>
        <items>
               <shirt number="557">
                             <quantity>1</quantity>
                             <color value="blue"/>
               </shirt>
               <shirt number="557">
                             <quantity>1</quantity>
                             <color value="sage"/>
               </shirt>
               <hat number="563">
                             <quantity>1</quantity>
               </hat>
        </items>
        <products>
               <product>
                             <number>557</number>
                             <name>Short-Sleeved Linen Blouse</name>
                             <price currency="USD">29.99</price>
               </product>
               <product>
                             <number>563</number>
                             <name>Ten-Gallon Hat</name>
                             <price currency="USD">69.99</price>
               </product>
        </products>
</order>
```

The *key* & *keyref* constraints

```
<xsd:element name="order" type="OrderType">
    <xsd:keyref name="prodNumKeyRef" refer="prodNumKey">
        <xsd:selector xpath="items/*"/>
        <xsd:field xpath="@number"/>
    </xsd:keyref>

    <xsd:key name="prodNumKey">
        <xsd:selector xpath=".//product"/>
        <xsd:field xpath="number"/>
    </xsd:key>
</xsd:element>
```

Schema Composition

Assembling A Schema from Multiple Schema Documents

- Why breaking down?
  - Easier to reuse;
  - Easier to maintain;
  - Less chances of name collisions.
- How to assemble?
  - Include
  - Import

Include: An Example

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            xmlns="http://example.org/ord"
            targetNamespace="http://example.org/ord">
   <xsd:include schemaLocation="ord.xsd"/>
   <xsd:element name="order" type="OrderType"/>
   <xsd:complexType name="OrderType">
      <xsd:sequence>
           <xsd:element name="number" type="OrderNumType"/>
           <!--...-->
      </xsd:sequence>
   </xsd:complexType>
</xsd:schema>

< -- ord.xsd -- >
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            xmlns="http://example.org/ord"
            targetNamespace="http://example.org/ord">
   <xsd:simpleType name="OrderNumType">
      <xsd:restriction base="xsd:string"/>
   </xsd:simpleType>
</xsd:schema>
```

The Rules of Including

- The *include* elements
  - May only appear at the top level of schema document, and
  - Must appear at the beginning.

- When you use includes, one of the following must be true
  - Both schema documents have the same target namespace;
  - Neither schema document has a target namespace; or
  - The including schema document has target namespace, and the included schema document does not have a target namespace

Import: An Example

```
<xsd:schema xmlns:xsd=“http://www.w3.org/2001/XMLSchema”
                        xmlns=“http://example.org/prod”
                        targetNamespace="http://example.org/prod">

     <xsd:complexType name="ItemsType">
         <xsd:sequence>
                 <xsd:element name="product" type="ProductType"/>
         </xsd:sequence>
     </xsd:complexType>
</xsd:schema>
```

Import: An Example (Continued)

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
                xmlns="http://example.org/ord"
                xmlns:prod="http://example.org/prod"
      targetNamespace="http://example.org/ord">
      <xsd:import namespace="http://example.org/prod"
               schemaLocation="prod.xsd"/>

      <xsd:element name="order" type="OrderType"/>

      <xsd:complexType name="OrderType">
          <xsd:sequence>
              <xsd:element name="number" type="OrderNumType"/>
              <xsd:element name="items" type="prod:ItemsType"/>
              <!--...-->
          </xsd:sequence>
      </xsd:complexType>
</xsd:schema>
```

The Rules of Importing

- The *import* elements
  - May only appear at the top level of a schema document, and
  - Must appear at the beginning.
- Regarding namespace
  - The imported namespace cannot be the same as the target namespace of the importing schema document.
  - If importing schema document has no target namespace, the import element must have a namespace attribute.

Questions?