



W3C XML Schema



Topics

- Motivation
- Simple types
- Complex types
- Element vs. Attribute
- Occurrences
- List type
- Union type
- Explicit vs. Implicit
- Element content
- Annotation
- Choices and Group
- Namespaces



Motivations for W3C XML Schema

XML Schema Status



- W3C candidate recommendation as of Oct. 2000
 - ◆ <http://www.w3.org/XML/Schema.html>
 - ◆ XML schema aware tools
 - Several free and commercial versions available (Check the above site)
 - NetBeans 5.5 and after (Free)
 - XMLSpy (Commercial, Not Free)
 - Apache Xerces
 - DTD to XML Schema conversion tool



Motivations of XML Schema

- Provide more powerful and flexible schema language than DTD
- Represent XML document syntax in XML language
 - ◆ XML tools can be readily used
- Support **non-textual data types**
 - ◆ Important to B2B, e-Commerce
- Handle complex syntax



Valid vs. Schema-valid

- XML schema is not part of XML 1.0
- XML document that is validated with DTD is “valid”
- XML document that conforms to XML schema is “**schema-valid**”
- XML document that conforms to a particular XML schema is called “**instance document**” of that schema





Definitions vs. Declarations

Definition and Declaration



- Definition
 - ◆ Create **new types** (both simple and complex types)
- Declaration
 - ◆ Enable elements and attributes with specific names and types (both simple and complex) to appear in document instances



Example

<!-- Definition: Creation of a type -->

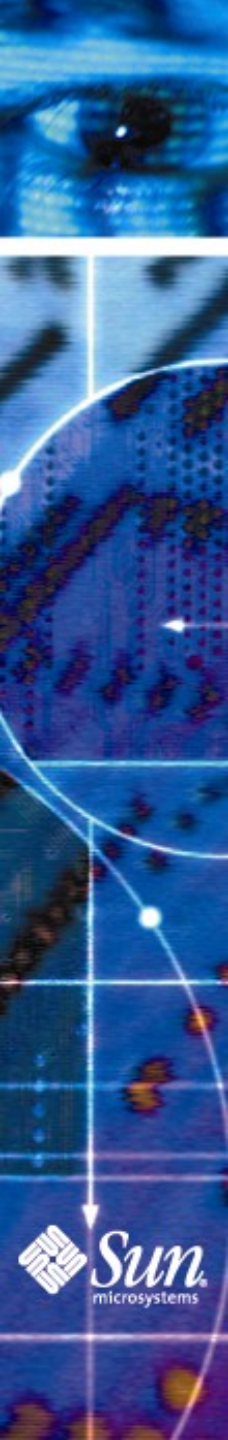
```
<xsd:simpleType name="zipUnion">
```


```
  <xsd:union memberTypes="USState listOfMyIntType"/>
```

```
</xsd:simpleType>
```

<!-- Declaration -->

```
<element name="zips" type="zipUnion">
```





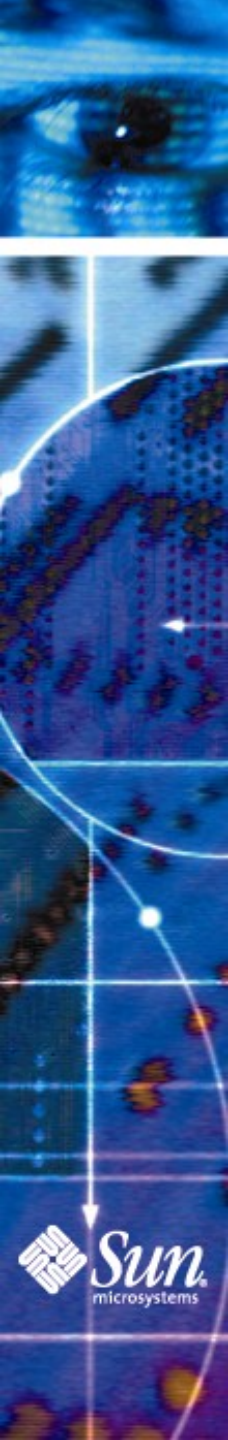
Schema Data Types: Simple Types & Complex Types

Schema Data Types



- Simple type
 - ◆ Do not have sub-elements
 - Do not have “*element*” sub-elements
 - Do not have “*attribute*” sub-elements
 - ◆ Predefined type or derived from predefined type
- Complex type
 - ◆ Have either “*element*” sub-elements or “*attribute*” sub-elements





Simple Types

Predefined Simple Types

- String, CDATA, token, byte, unsignedByte, binary, integer, positiveInteger, negativeInteger, nonNegativeInteger, nonPositiveInteger, int, unsignedInt, long, unsignedLong, short, unsignedShort, decimal, float, double, boolean, time, timeInstant, timePeriod, timeDuration, date, month, year, century, recurringDay, recurringDate, recurringDuration, Name, QName, NCName, uriReference, language, ID, IDREF, IDREFS, ENTITY, ENTITIES, NOTATION, NMTOKEN, NMTOKENS

Examples of Predefined Simple type

<element name="Title" type="string"/>

<element name="Heading" type="string"/>

<element name="Topic" type="string"/>

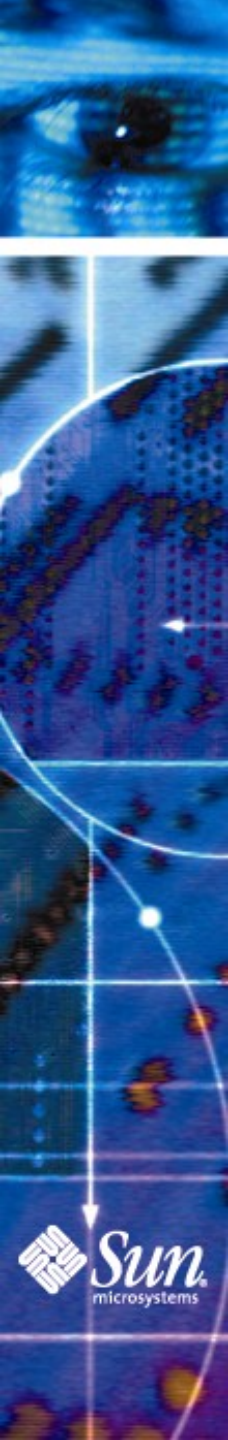
<element name="Price" type="decimal"/>

<attribute name="focus" type="string"/>



Derived Simple Type

- Derived from existing simple types (predefined or derived)
- Typically **restricting** existing simple type
 - ◆ The legal range of values for a new type is subset of the ones of existing type
 - ◆ Existing type is called **base** type
 - ◆ Use **restriction** element along with **facets** to restrict the range of values
 - Facets are rules of restriction



Example of Derived Simple Type 1 (Numeric range)

```
<xsd:simpleType name="myInteger">  
  <xsd:restriction base="xsd:integer">  
    <xsd:minInclusive value="10000"/>  
    <xsd:maxInclusive value="99999"/>  
  </xsd:restriction>  
</xsd:simpleType>
```

- Defining *myInteger* type whose range of value is between 10000 and 99999
- *minInclusive* and *maxInclusive* are facets that can be applied to *integer* type

Example of Derived Simple Type 2 (Regular expression)

```
<xsd:simpleType name="SKU">  
  <xsd:restriction base="xsd:string">  
    <xsd:pattern value="\d{3}-[A-Z]{2}"/>  
  </xsd:restriction>  
</xsd:simpleType>
```

- Defining new type called *SKU*
- *pattern* is a facet that can be applied to *string*
 - ◆ Regular expression
 - ◆ three digits followed by a hyphen followed by two upper-case ASCII letters

Example of Derived Simple Type 3 (Enumeration)

```
<xsd:simpleType name="USState">  
  <xsd:restriction base="xsd:string">  
    <xsd:enumeration value="AK"/>  
    <xsd:enumeration value="AL"/>  
    <xsd:enumeration value="AR"/>  
    <!-- and so on ... -->  
  </xsd:restriction>  
</xsd:simpleType>
```

- *enumeration* facet limits a simple type to a set of distinct values



Complex Type

Complex Type



- Defined using “*complexType*” element
- Typically contain
 - ◆ *element* declarations
 - ◆ element references
 - ◆ *attribute* declarations



complexType Example 1

```
<xsd:complexType name="USAddress" >
  <xsd:sequence>
    <xsd:element name="name" type="xsd:string" />
    <xsd:element name="street" type="xsd:string" />
    <xsd:element name="city" type="xsd:string" />
    <xsd:element name="state" type="xsd:string" />
    <xsd:element name="zip" type="xsd:decimal" />
  </xsd:sequence>
  <xsd:attribute name="country" type="xsd:NMTOKEN"
    use="fixed" value="US"/>
</xsd:complexType>
```

complexType Example 1

- Definition of *USAddress* type
- It contains 5 element declarations and one attribute declaration
- USAddress definition contains only declarations involving simple types: *string*, *decimal*, and *NMTOKEN*

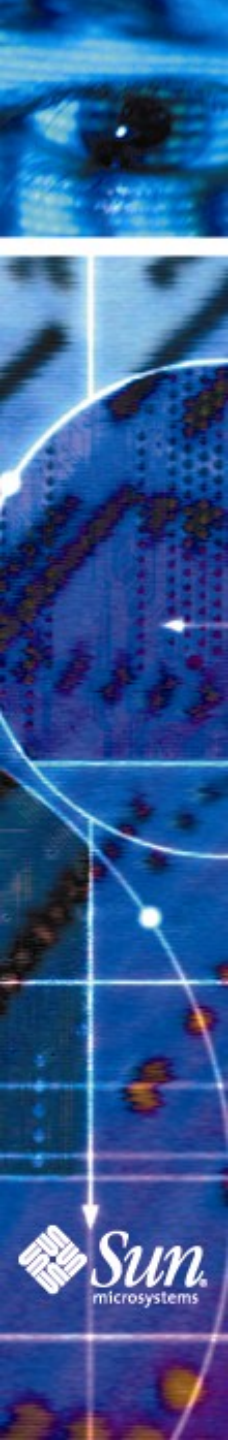


complexType Example 2

```
<xsd:complexType name="PurchaseOrderType">
  <xsd:sequence>
    <xsd:element name="shipTo" type="USAddress"/>
    <xsd:element name="billTo" type="USAddress"/>
    <xsd:element ref="comment" minOccurs="0"/>
    <xsd:element name="items" type="Items"/>
  </xsd:sequence>
  <xsd:attribute name="orderDate" type="xsd:date"/>
</xsd:complexType>
```

complexType Example 2

- Definition of *PurchaseOrder* type
- Contains element declarations referencing complex types, e.g. *USAddress*, *Items*
- Contains element declaration referencing “pre-defined” simple types: *date*



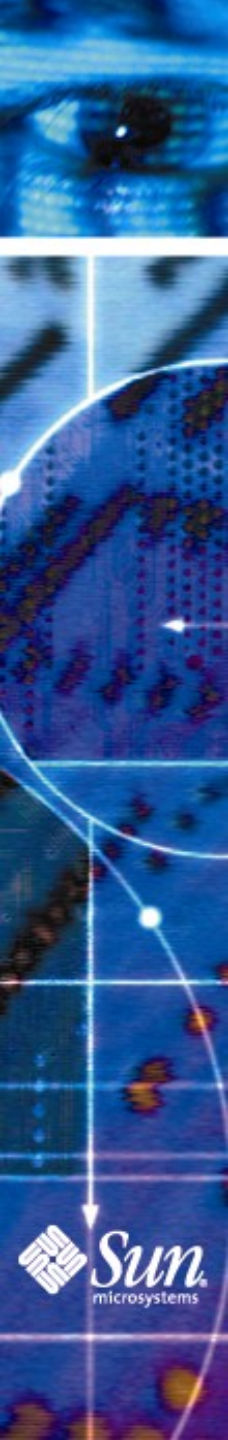


Elements vs. Attributes

Element vs. Attribute



- Element declarations can reference both simple types or complex types
- All attribute declarations can reference only simple types
 - ◆ Because they cannot contain other sub-elements





ref Attribute

- To use an existing element or attribute rather than declaring a new element or attribute
- Existing element must be **global element** - an element that is declared under root element

ref Example

```
<xsd:schema
  xmlns:xsd="http://www.w3.org/2000/08/XMLSchema">

  <xsd:element name="purchaseOrder" type="PurchaseOrderType"/>
  <xsd:element name="comment" type="xsd:string"/>

  <xsd:complexType name="PurchaseOrderType">
    <xsd:sequence>
      <xsd:element name="shipTo" type="USAddress"/>
      <xsd:element name="billTo" type="USAddress"/>
      <xsd:element ref="comment" minOccurs="0"/>
      <xsd:element name="items" type="Items"/>
    </xsd:sequence>
    <xsd:attribute name="orderDate" type="xsd:date"/>
  </xsd:complexType>
```





Occurrences

Occurrences of Elements

- *minOccurs*
- *maxOccurs*
- *fixed* = “Hannah”
 - ◆ If the element appears (optional), the value **must** be “Hannah”, otherwise the value is set to “Hannah” by the parser
- *default* = “Hannah”
 - ◆ If the element appears (optional), the value is set to what is specified, otherwise value is set to “Hannah” by the parser

Example

- `<element name="test" type="string"`
 - ◆ `minOccurs="1" maxOccurs="1"`
 - ◆ `minOccurs="1" maxOccurs="1" fixed="Hannah"`
 - ◆ `minOccurs="2" maxOccurs="unbounded"`
 - ◆ `minOccurs="0" maxOccurs="1" fixed="Hannah"`
 - ◆ `minOccurs="0" maxOccurs="1" default="Hannah"`
 - ◆ `minOccurs="0" maxOccurs="2" default="Hannah"`
 - ◆ `minOccurs="0" maxOccurs="0"`
- `>`





Occurrences of Attributes

- Attributes can occur once or not at all
- “*use*” attribute
 - ◆ required
 - ◆ optional
 - ◆ fixed
 - ◆ default
- “*value*” attribute

Example

- <attribute name="test" type="string"
 - ◆ use="required"
 - ◆ use="required" value="37"
 - ◆ use="optional"
 - ◆ use="fixed", value="37"
 - ◆ use="default" value="37"
 - ◆ use="prohibited"
- >

Example

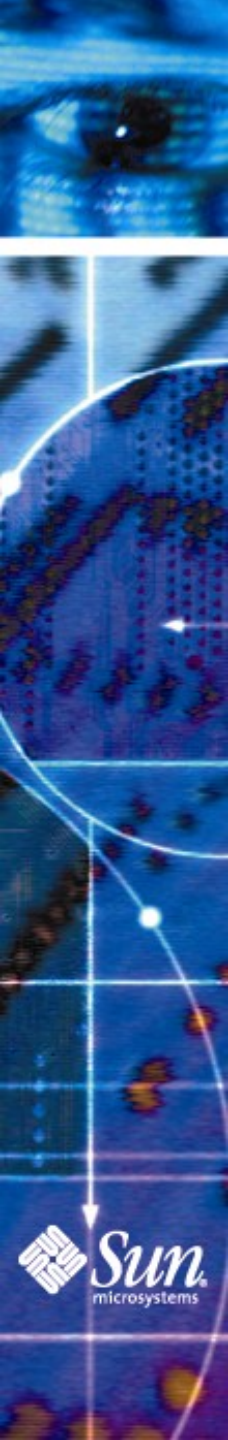
```
<xsd:complexType name="USAddress">
  <xsd:sequence>
    <xsd:element name="name" type="xsd:string"/>
    <xsd:element name="street" type="xsd:string"/>
    ...
  </xsd:sequence>
  <xsd:attribute name="country" type="xsd:NMTOKEN"
    use="fixed" value="US"/>
</xsd:complexType>
```

- Appearance of a *country* attribute is optional
- Its value must be *US* if it does appear
- If it does not appear, parser will create a country attribute with value *US*

Attributes

- Enumeration
 - ◆ *simpleType* element with *base* attribute
 - ◆ *base* attribute specifies the type





Complete Example

Example

```
<complexType name="ContentsType">
  <element name="Chapter" maxOccurs="*">
    <complexType>
      <element name="Heading" type="string" minOccurs="0" />
      <element name="Topic" maxOccurs="*">
        <complexType content="string">
          <attribute name="subSections" type="integer" />
        </complexType>
      </element>
      <attribute name="focus" default="Java">
        <simpleType base="string">
          <enumeration value="XML" />
          <enumeration value="Java" />
        </simpleType>
      </attribute>
    </complexType>
  </element>
</complexType>
```

Complete Example (1st page)

```
<?xml version="1.0"?>
<schema targetNamespace="http://www.oreilly.com/catalog/javaxml/"
        xmlns="http://www.w3.org/1999/XMLSchema"
        xmlns:JavaXML="http://www.oreilly.com/catalog/javaxml/">

  <element name="Book" type="JavaXML:BookType" />

  <complexType name="BookType">
    <element name="Title" type="string" />
    <element name="Contents" type="JavaXML:ContentsType" />
    <element name="Copyright" type="string" />
  </complexType>
```



Continued

```
<complexType name="ContentsType">
  <element name="Chapter" maxOccurs="*">
    <complexType>
      <element name="Heading" type="string" minOccurs="0" />
      <element name="Topic" maxOccurs="*">
        <complexType content="string">
          <attribute name="subSections" type="integer" />
        </complexType>
      </element>
      <attribute name="focus" default="Java">
        <simpleType base="string">
          <enumeration value="XML" />
          <enumeration value="Java" />
        </simpleType>
      </attribute>
    </complexType>
  </element>
  <element name="SectionBreak" minOccurs="0" maxOccurs="*">
    <complexType content="empty" />
  </element>
</complexType>

</schema>
```



List Type

List Type

- Comprised of sequences of **atomic simple types**
- Three built-in list types
 - ◆ *NMTOKENS, IDREFS, ENTITIES*
- User defined List type
 - ◆ Derive from atomic types
- **facets**
 - ◆ *length, minLength, maxLength, enumeration*



Example of List Type

- Schema

```
<xsd:simpleType name="listOfMyIntType">  
  <xsd:list itemType="myInteger"/>  
</xsd:simpleType>
```

- Instance Document

```
<listOfMyInt>20003 15037 95977 95945</listOfMyInt>
```

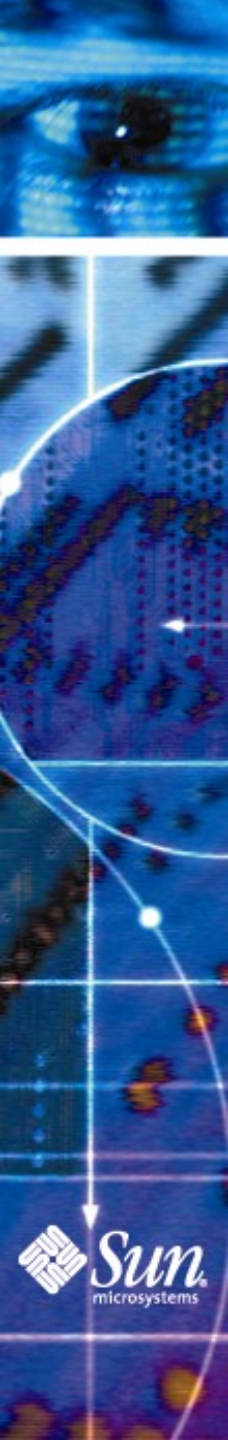
Example: List Type with Facet

```
<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>
```

```
<element name="sixStates" type="SixUSStates">
```

- Define a list of exactly six US states (*SixUSStates*), we first define a new list type called *USStateList* from *USState*, and then we derive *SixUSStates* by restricting *USStateList* to only six items
- `<sixStates>PA NY CA NY LA AK</sixStates>`



Union Type



Union Type

- Enables an element or attribute value to be one or more instances of one type drawn from the union of multiple atomic and list types
- facets: *pattern* and *enumeration*

Union Type for Zipcodes

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

```
<element name="zips" type="zipUnion">
```

```
<zips>CA</zips>
```

```
<zips>95630 95977 95945</zips>
```

```
<zips>AK</zips>
```



Explicit Type vs. Implicit Type

Explicit Type vs. Implicit Type

- Explicit type
 - ◆ One in which **a name** is given to the type
 - ◆ Element that uses the type is generally defined in a different section of the file
 - ◆ Object-oriented in that same explicit type is used as the type for several different elements
- Implicit type (nameless type)
 - ◆ Use when the type is not needed by multiple elements

Example of Explicit Type

`<!-- Type has a name zipUnion -->`

```
<xsd:simpleType name="zipUnion">
```

```
  <xsd:union memberTypes="USState listOfMyIntType"/>
```

```
</xsd:simpleType>
```

`<!-- zipUnion type is used in other parts of Schema document -->`

```
<element name="zips" type="zipUnion">
```

...

```
<element name="theOtherZips" type="zipUnion">
```

...

```
<element name="theThirdZips" type="zipUnion">
```

Example of Implicit Type

```
<xsd:complexType name="Items">                                <!-- Explicit complexType -->
  <xsd:sequence>
    <xsd:element name="item" minOccurs="0" maxOccurs="unbounded">
      <xsd:complexType>                                       <!-- Implicit complexType -->
        <xsd:sequence>
          <xsd:element name="productName" type="xsd:string"/>
          <xsd:element name="quantity">
            <xsd:simpleType>                                   <!-- Implicit simpleType -->
              <xsd:restriction base="xsd:positiveInteger">
                <xsd:maxExclusive value="100"/>
              </xsd:restriction>
            </xsd:simpleType>
          </xsd:element>
          <xsd:element name="USPrice" type="xsd:decimal"/>
          <xsd:element ref="comment" minOccurs="0"/>
          <xsd:element name="shipDate" type="xsd:date" minOccurs="0"/>
        </xsd:sequence>
        <xsd:attribute name="partNum" type="SKU"/>
      </xsd:complexType>
    </xsd:element>
  </xsd:sequence>
</xsd:complexType>
```




Element Content



Element Content

- How content of an element gets constructed
- Three different ways
 - ◆ Complex types from simple types
 - ◆ Mixed content
 - Elements mixed with character content
 - ◆ Empty content

Complex Types from Simple Types

`<USPrice>345.67</USPrice>` (usage in document instance)

`<xsd:element name="USPrice" type="decimal"/>` (in XML schema)

`<internationalPrice currency="EUR">423.46</internationalPrice>`

??? (what would be done in XML schema?)

- Need to create `complexType` based on simple type
 - ◆ Simple type cannot have attributes
 - ◆ Have to have *attribute* declaration
 - ◆ Based on *decimal* simple type

Complex Type from a Simple Type

```
<xsd:element name="internationalPrice">
  <xsd:complexType>
    <xsd:simpleContent>
      <xsd:extension base="xsd:decimal">
        <xsd:attribute name="currency"
          type="xsd:string" />
      </xsd:extension>
    </xsd:simpleContent>
  </xsd:complexType>
</xsd:element>
```

- *simpleContent* indicates that the content model of the new type contains only character data and no *element* declaration

Mixed Content

- Sub-elements mixed with character data

<letterBody>

<salutation>Dear Mr.<name>Robert Smith</name>.</salutation>

Your order of <quantity>1</quantity> <productName>Baby Monitor</productName> shipped from our warehouse on

<shipDate>1999-05-21</shipDate>

</letterBody>



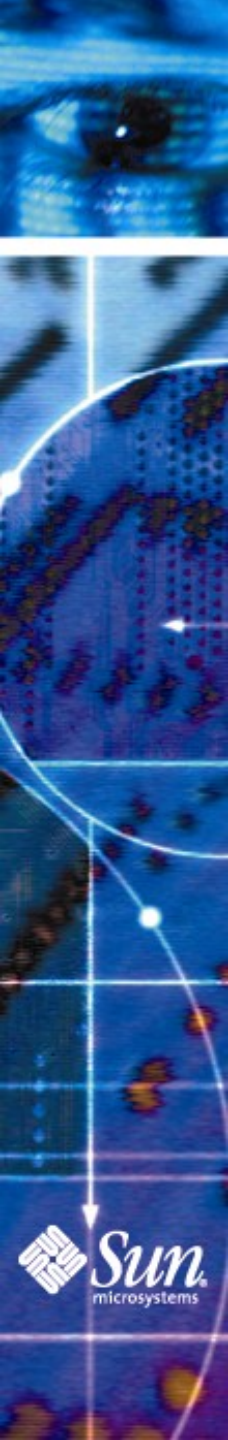
Mixed Content

```
<xsd:element name="letterBody">
  <xsd:complexType mixed="true">
    <xsd:sequence>
      <xsd:element name="salutation">
        <xsd:complexType mixed="true"> <!-- Implicit definition -->
          <xsd:sequence>
            <xsd:element name="name" type="xsd:string"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="quantity" type="xsd:positiveInteger"/i>
      <xsd:element name="productName" type="xsd:string"/>
      <xsd:element name="shipDate" type="xsd:date" minOccurs="0"/>
      <!-- etc -->
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```



Empty Content

- Define a type which do not declare any elements in its content
 - ◆ Type's content model is empty



Empty Content 1

<internationalPrice currency="EUR" value="345.23"/>

<xsd:element name="internationalPrice">

<xsd:complexType>

<xsd:complexContent>

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

**<xsd:attribute name="currency"
type="xsd:string"/>**

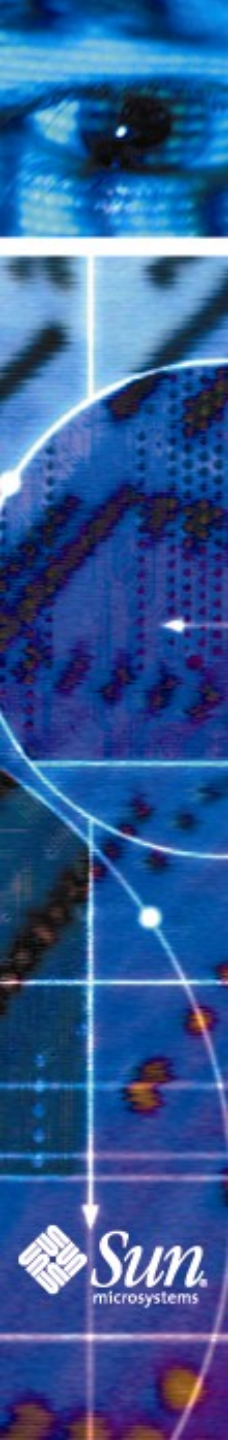
**<xsd:attribute name="value"
type="xsd:decimal"/>**

</xsd:restriction>

</xsd:complexContent>

</xsd:complexType>

</xsd:element>



Empty Content 2

- complexContent
 - ◆ To restrict or extend the content model of a complex type
- `<xsd:restriction base="xsd:anyType">`



Empty Content 3

```
<xsd:element name="internationalPrice">
  <xsd:complexType>
    <xsd:attribute name="currency"
      type="xsd:string"/>
    <xsd:attribute name="value"
      type="xsd:decimal"/>
  </xsd:complexType>
</xsd:element>
```

- A complex type defined without *complexContent* is interpreted as shorthand for complex content that restricts *anyType*



anyType



anyType

- Base type from which all simple and complex types are derived
- Does not constrain its contents in any way
- Default type when no type is specified
 - ◆ `<xsd:element name="anything" type="xsd:anyType" />` is same as
 - ◆ `<xsd:element name="anything"/>`
- Use more constrained types whenever possible



Annotation



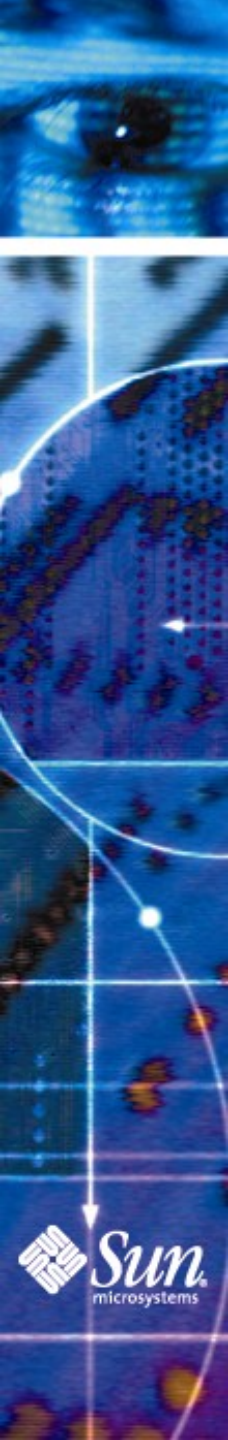
Annotation

- Appears at the beginning of most schema constructions
- Can have two sub-elements
 - ◆ *documentation*
 - ◆ *applInfo*
- *documentation*
 - ◆ *For human readable materials*
- *applInfo*
 - ◆ For tools, stylesheets and other applications

Example of Annotation

```
<xsd:element name="internationalPrice">
  <xsd:annotation>
    <xsd:documentation> element declared with anonymous type
  </xsd:documentation>
</xsd:annotation>
<xsd:complexType>
  <xsd:annotation>
    <xsd:documentation> empty anonymous type with 2 attributes
  </xsd:documentation>
</xsd:annotation>
<xsd:complexContent>
  <xsd:restriction base="xsd:anyType">
    <xsd:attribute name="currency" type="xsd:string" />
    <xsd:attribute name="value" type="xsd:decimal" />
  </xsd:restriction>
</xsd:complexContent>
</xsd:complexType>
</xsd:element>
```





Choices & Group



Choice and Group

- *choice*

- ◆ Only one of its children to appear in an instance

- *group*

- ◆ Grouping a group of elements
- ◆ Further constraints
 - *sequence*

- *all*

- ◆ Appear zero or once
- ◆ In any order



Choice and Sequence Groups

```
<xsd:complexType name="PurchaseOrderType">
  <xsd:sequence>
    <xsd:choice>
      <xsd:group ref="shipAndBill" />
      <xsd:element name="singleUSAddress" type="USAddress" />
    </xsd:choice>
    <xsd:element ref="comment" minOccurs="0"/>
    <xsd:element name="items" type="Items" />
  </xsd:sequence>
  <xsd:attribute name="orderDate" type="xsd:date" />
</xsd:complexType>

<xsd:group name="shipAndBill">
  <xsd:sequence>
    <xsd:element name="shipTo" type="USAddress" />
    <xsd:element name="billTo" type="USAddress" />
  </xsd:sequence>
</xsd:group>
```


Example of *all*

```
<xsd:complexType name="PurchaseOrderType">  
  <xsd:all>  
    <xsd:element name="shipTo" type="USAddress"/>  
    <xsd:element name="billTo" type="USAddress"/>  
    <xsd:element ref="comment" minOccurs="0"/>  
    <xsd:element name="items" type="Items" />  
  </xsd:all>  
  <xsd:attribute name="orderDate" type="xsd:date" />  
</xsd:complexType>
```



Attribute Group

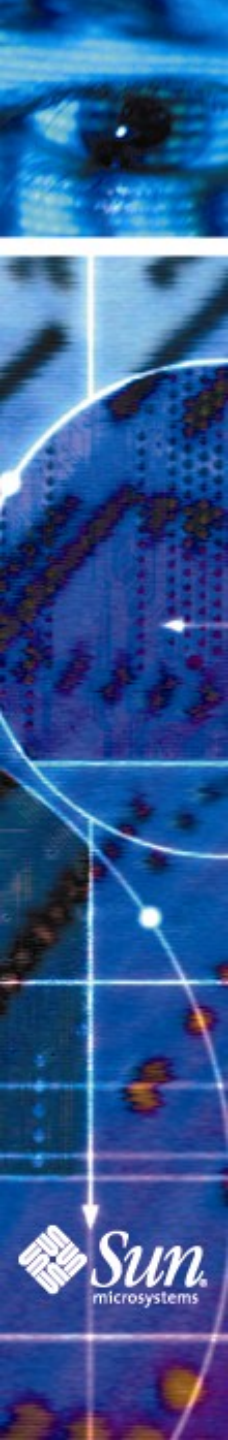
- Define attribute group using *attributeGroup* element
- Referenced in multiple definitions and declarations
- Improve readability and maintenance
- They have to appear at the end of complex type definitions



Example of *attributeGroup*

```
<xsd:attributeGroup name="ItemDelivery">
  <xsd:attribute name="partNum" type="SKU"/>
  <xsd:attribute name="weightKg" type="xsd:decimal"/>
  <xsd:attribute name="shipBy">
    <xsd:simpleType>
      <xsd:restriction base="xsd:string">
        <xsd:enumeration value="air"/>
        <xsd:enumeration value="land"/>
        <xsd:enumeration value="any"/>
      </xsd:restriction>
    </xsd:simpleType>
  </xsd:attribute>
</xsd:attributeGroup>
```

```
<!-- attributeGroup replaces individual declarations -->
<xsd:attributeGroup ref="ItemDelivery"/>
```





Schema Namespaces

Schema Namespaces

- Two namespaces to deal with
 - ◆ Namespace for XML Schema document itself
 - <http://www.w3.org/2000/08/XMLSchema>
 - In XML Schema document, this is set as default namespace
 - Prefix string convention is schema
 - ◆ Namespace for XML document being constrained



Schema Namespaces

- targetNamespace
 - ◆ Is the namespace that is going to be assigned to the schema you are creating.
 - ◆ It is the namespace an instance is going to use to access the types it declares





Summary

- Status
- Motivation
- Namespaces
- **Vocabularies**
 - ◆ element
 - ◆ complexType
 - ◆ attribute
 - ◆ simpleType
 - ◆ enumeration



XML Document and XML Schema

- XML document (Instance document) is not required to make a reference to XML schema
 - ◆ Validator has to have access to XML schema
- Hints of where to get schema document
 - ◆ Validator can ignore these hints

schemaLocation

- In an instance document, the attribute `xsi:schemaLocation`

```
<purchaseReport
  xmlns="http://www.example.com/Report"
  xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
  xsi:schemaLocation="http://www.example.com/Report
    http://www.example.com/Report.xsd"
  period="P3M" periodEnding="1999-12-31">
  <!-- etc -->
</purchaseReport>
```

References

- XML Schema Primer on W3C Candidate Recommendation 24 October 2000, Edited by David Fallside, <http://www.w3.org/TR/2000/CR-xmlschema-0-20001024/>
- “Java and XML” written by Brett McLaughlin, O’Reilly, June 2000 (First edition), Chapter 4 “Constraining XML”, XML Schema section, page 108-123

