

XML Schema Types

```

<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            targetNamespace="http://www.books.org"
            xmlns="http://www.books.org"
            elementFormDefault="qualified">
  <xsd:element name="BookStore">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="Book" minOccurs="1" maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="Book">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="Title" minOccurs="1" maxOccurs="1"/>
        <xsd:element ref="Author" minOccurs="1" maxOccurs="1"/>
        <xsd:element ref="Date" minOccurs="1" maxOccurs="1"/>
        <xsd:element ref="ISBN" minOccurs="1" maxOccurs="1"/>
        <xsd:element ref="Publisher" minOccurs="1" maxOccurs="1"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="Title" type="xsd:string"/>
  <xsd:element name="Author" type="xsd:string"/>
  <xsd:element name="Date" type="xsd:string"/>
  <xsd:element name="ISBN" type="xsd:string"/>
  <xsd:element name="Publisher" type="xsd:string"/>
</xsd:schema>

```

BookStore.xsd

Inlining Element Declarations

- In the previous slide we declared an element and then we referred to that element declaration. Alternatively, we can *inline* the element declarations.
- On the following slide is an alternate (equivalent) way of representing the schema shown previously, using *inlined element declarations*.

```

<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            targetNamespace="http://www.books.org"
            xmlns="http://www.books.org"
            elementFormDefault="qualified">
  <xsd:element name="BookStore">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="Book" maxOccurs="unbounded">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name="Title" type="xsd:string"/>
              <xsd:element name="Author" type="xsd:string"/>
              <xsd:element name="Date" type="xsd:string"/>
              <xsd:element name="ISBN" type="xsd:string"/>
              <xsd:element name="Publisher" type="xsd:string"/>
            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>

```

Note that we have moved all the element declarations inline, and we are no longer ref'ing to the element declarations. This results in a much more compact schema.

This way of designing the schema - by inlining everything - is called the *Russian Doll design*.

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.books.org"
  xmlns="http://www.books.org"
  elementFormDefault="qualified">
  <xsd:element name="BookStore">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="Book" maxOccurs="unbounded">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name="Title" type="xsd:string"/>
              <xsd:element name="Author" type="xsd:string"/>
              <xsd:element name="Date" type="xsd:string"/>
              <xsd:element name="ISBN" type="xsd:string"/>
              <xsd:element name="Publisher" type="xsd:string"/>
            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

Anonymous types (no name)

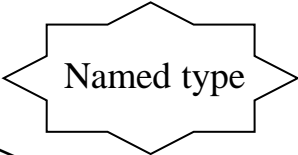
Named Types

- The following slide shows an alternate (equivalent) schema which uses a named *complexType*.

```

<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.books.org"
  xmlns="http://www.books.org"
  elementFormDefault="qualified">
  <xsd:element name="BookStore">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="Book" type="BookPublication" maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <xsd:complexType name="BookPublication">
    <xsd:sequence>
      <xsd:element name="Title" type="xsd:string"/>
      <xsd:element name="Author" type="xsd:string"/>
      <xsd:element name="Date" type="xsd:string"/>
      <xsd:element name="ISBN" type="xsd:string"/>
      <xsd:element name="Publisher" type="xsd:string"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>

```



The advantage of splitting out Book's element declarations and wrapping them in a named type is that now this type can be *reused* by other elements.

Please note that:

```
<xsd:element name="A" type="foo"/>
<xsd:complexType name="foo">
  <xsd:sequence>
    <xsd:element name="B" .../>
    <xsd:element name="C" .../>
  </xsd:sequence>
</xsd:complexType>
```

Element A *references* the complexType *foo*.

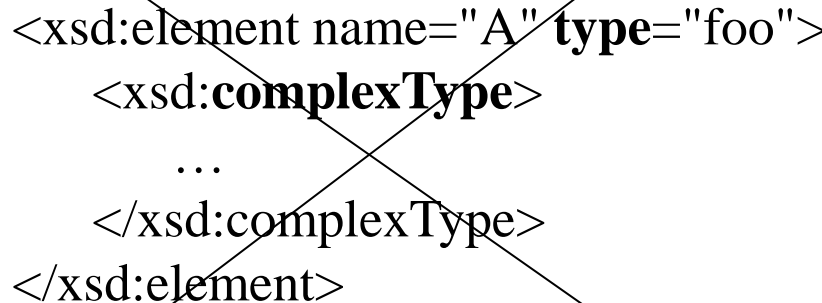
is equivalent to:

```
<xsd:element name="A">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="B" .../>
      <xsd:element name="C" .../>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Element A has the complexType definition *inlined* in the element declaration.

type Attribute or *complexType* Child Element, but not Both!

- An element declaration can have a type attribute, or a complexType child element, but it cannot have **both** a type attribute and a complexType child element.



```
<xsd:element name="A" type="foo">  
  <xsd:complexType>  
    ...  
  </xsd:complexType>  
</xsd:element>
```

Summary of Declaring Elements (two ways to do it)

1

```
<xsd:element name="name" type="type" minOccurs="int" maxOccurs="int" />
```

↑
A simple type
(e.g., xsd:string)
or the name of
a complexType
(e.g., BookPublication)

↑
A nonnegative
integer

↑
A nonnegative
integer or "unbounded"

*Note: minOccurs and maxOccurs can only
be used in nested (local) element declarations.*

2

```
<xsd:element name="name" minOccurs="int" maxOccurs="int">
```

```
  <xsd:complexType>
```

```
    ...
```

```
  </xsd:complexType>
```

```
</xsd:element>
```

Inlined declaration

Lab 2.1

- Practice converting to Inline (Russian Doll) format

Problem

- Defining the Date element to be of type string is unsatisfactory (it allows any string value to be input as the content of the Date element, including non-date strings).
 - We would like to constrain the allowable content that Date can have. Modify the BookStore schema to restrict the content of the Date element to just date values (actually, year values. See next two slides).
- Similarly, constrain the content of the ISBN element to content of this form: d-ddddd-ddd-d or d-ddd-ddddd-d or d-dd-dddddd-d, where 'd' stands for 'digit'

The *date* Datatype

- A *built-in datatype* (i.e., **schema validators know about this datatype**)
- This datatype is used to represent a specific day (year-month-day)
- Elements declared to be of type *date* must follow this form: CCYY-MM-DD
 - range for CC is: 00-99
 - range for YY is: 00-99
 - range for MM is: 01-12
 - range for DD is:
 - 01-28 if month is 2
 - 01-29 if month is 2 and the gYear is a leap year
 - 01-30 if month is 4, 6, 9, or 11
 - 01-31 if month is 1, 3, 5, 7, 8, 10, or 12
 - Example: 1999-05-31 represents May 31, 1999

The *gYear* Datatype

- A built-in datatype (Gregorian calendar year)
- Elements declared to be of type *gYear* must follow this form: CCYY
 - range for CC is: 00-99
 - range for YY is: 00-99
 - Example: 1999 indicates the *gYear* 1999

```

<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http://www.books.org"
    xmlns="http://www.books.org"
    elementFormDefault="qualified">
    <xsd:simpleType name="ISBNType">
        <xsd:restriction base="xsd:string">
            <xsd:pattern value="\d{1}-\d{5}-\d{3}-\d{1}"/>
            <xsd:pattern value="\d{1}-\d{3}-\d{5}-\d{1}"/>
            <xsd:pattern value="\d{1}-\d{2}-\d{6}-\d{1}"/>
        </xsd:restriction>
    </xsd:simpleType>
    <xsd:element name="BookStore">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="Book" maxOccurs="unbounded">
                    <xsd:complexType>
                        <xsd:sequence>
                            <xsd:element name="Title" type="xsd:string"/>
                            <xsd:element name="Author" type="xsd:string"/>
                            <xsd:element name="Date" type="xsd:gYear"/>
                            <xsd:element name="ISBN" type="ISBNType"/>
                            <xsd:element name="Publisher" type="xsd:string"/>
                        </xsd:sequence>
                    </xsd:complexType>
                </xsd:element>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
</xsd:schema>

```

Here we are defining a new (user-defined) data-type, called *ISBNType*.

Declaring Date to be of type *gYear*, and ISBN to be of type *ISBNType* (defined above)

```

<xsd:simpleType name="ISBNType">
  <xsd:restriction base="xsd:string">
    <xsd:pattern value="\d{1}-\d{5}-\d{3}-\d{1}"/>
    <xsd:pattern value="\d{1}-\d{3}-\d{5}-\d{1}"/>
    <xsd:pattern value="\d{1}-\d{2}-\d{6}-\d{1}"/>
  </xsd:restriction>
</xsd:simpleType>

```

"I hereby declare a new type called ISBNType. It is a restricted form of the string type. Elements declared of this type must conform to one of the following patterns:

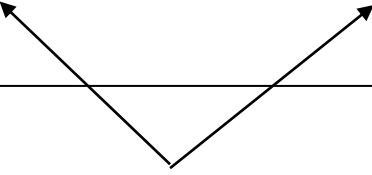
- First Pattern: 1 digit followed by a dash followed by 5 digits followed by another dash followed by 3 digits followed by another dash followed by 1 more digit, or
- Second Pattern: 1 digit followed by a dash followed by 3 digits followed by another dash followed by 5 digits followed by another dash followed by 1 more digit, or
- Third Pattern: 1 digit followed by a dash followed by 2 digits followed by another dash followed by 6 digits followed by another dash followed by 1 more digit."

These patterns are specified using *Regular Expressions*. In a few slides we will see more of the Regular Expression syntax.

Equivalent Expressions

```
<xsd:simpleType name="ISBNType">
  <xsd:restriction base="xsd:string">
    <xsd:pattern value="\d{1}-\d{5}-\d{3}-\d{1}"/>
    <xsd:pattern value="\d{1}-\d{3}-\d{5}-\d{1}"/>
    <xsd:pattern value="\d{1}-\d{2}-\d{6}-\d{1}"/>
  </xsd:restriction>
</xsd:simpleType>
```

```
<xsd:simpleType name="ISBNType">
  <xsd:restriction base="xsd:string">
    <xsd:pattern value="\d{1}-\d{5}-\d{3}-\d{1}|\d{1}-\d{3}-\d{5}-\d{1}|\d{1}-\d{2}-\d{6}-\d{1}"/>
  </xsd:restriction>
</xsd:simpleType>
```



The vertical bar means "or"

<xsd:complexType> or <xsd:simpleType>?

- When do you use the *complexType* element and when do you use the *simpleType* element?
 - Use the *complexType* element when you want to define child elements and/or attributes of an element
 - Use the *simpleType* element when you want to create a new type that is a refinement of a built-in type (string, date, gYear, etc)

Built-in Datatypes

- Primitive Datatypes
 - string → "Hello World"
 - boolean → { true, false, 1, 0 }
 - decimal → 7.08
 - float → 12.56E3, 12, 12560, 0, -0, INF, -INF
 - double → 12.56E3, 12, 12560, 0, -0, INF, -INF
 - duration → P1Y2M3DT10H30M12.3S
 - dateTime → format: CCYY-MM-DDThh:mm:ss
 - time → format: hh:mm:ss.sss
 - date → format: CCYY-MM-DD
 - gYearMonth → format: CCYY-MM
 - gYear → format: CCYY
 - gMonthDay → format: --MM-DD
- Atomic, built-in

Note: 'P' stands for Period and is required
 'T' is the date/time separator
 INF = infinity

Built-in Datatypes (cont.)

- Primitive Datatypes
 - gDay —————→
 - gMonth —————→
 - hexBinary —————→
 - base64Binary —————→
 - anyURI —————→
 - QName —————→
- Atomic, built-in
 - format: ---DD (note the 3 dashes)
 - format: --MM
 - a hex string
 - a base64 string
 - **http://www.xfront.com**
 - a namespace qualified name

The ‘g’ indicates ‘recurring’. e.g. gMonthDay could be used to denote the 25th December. The dashes have to be included as shown.

There are also many derived types i.e. types which are available to you which are based on these primitive types. For example, the derived type *integer* is derived from the primitive type *decimal*.

Creating your own Datatypes

- A new datatype can be defined from an existing datatype (called the "base" type) by specifying values for one or more of the optional *facets* for the base type.
- Example. The *string* primitive datatype has six optional facets:
 - length
 - minLength
 - maxLength
 - pattern
 - enumeration
 - whitespace (legal values: preserve, replace, collapse)

Example of Creating a New Datatype by Specifying Facet Values

```
<xsd:simpleType name="TelephoneNumber"> ①  
  <xsd:restriction base="xsd:string"> ②  
    <xsd:length value="8"/> ③  
    <xsd:pattern value="\d{3}-\d{4}"/> ④  
  </xsd:restriction>  
</xsd:simpleType>
```

1. This creates a new datatype called 'TelephoneNumber'.
2. Elements of this type can hold string values,
3. But the string length must be exactly 8 characters long *and*
4. The string must follow the pattern: ddd-dddd, where 'd' represents a 'digit'.
(Obviously, in this example the regular expression makes the length facet redundant.)

Another Example

```
<xsd:simpleType name="shape">  
  <xsd:restriction base="xsd:string">  
    <xsd:enumeration value="circle"/>  
    <xsd:enumeration value="triangle"/>  
    <xsd:enumeration value="square"/>  
  </xsd:restriction>  
</xsd:simpleType>
```

- This creates a new type called shape.
- An element declared to be of this type must have either the value circle, or triangle, or square.

Facets of the integer Datatype

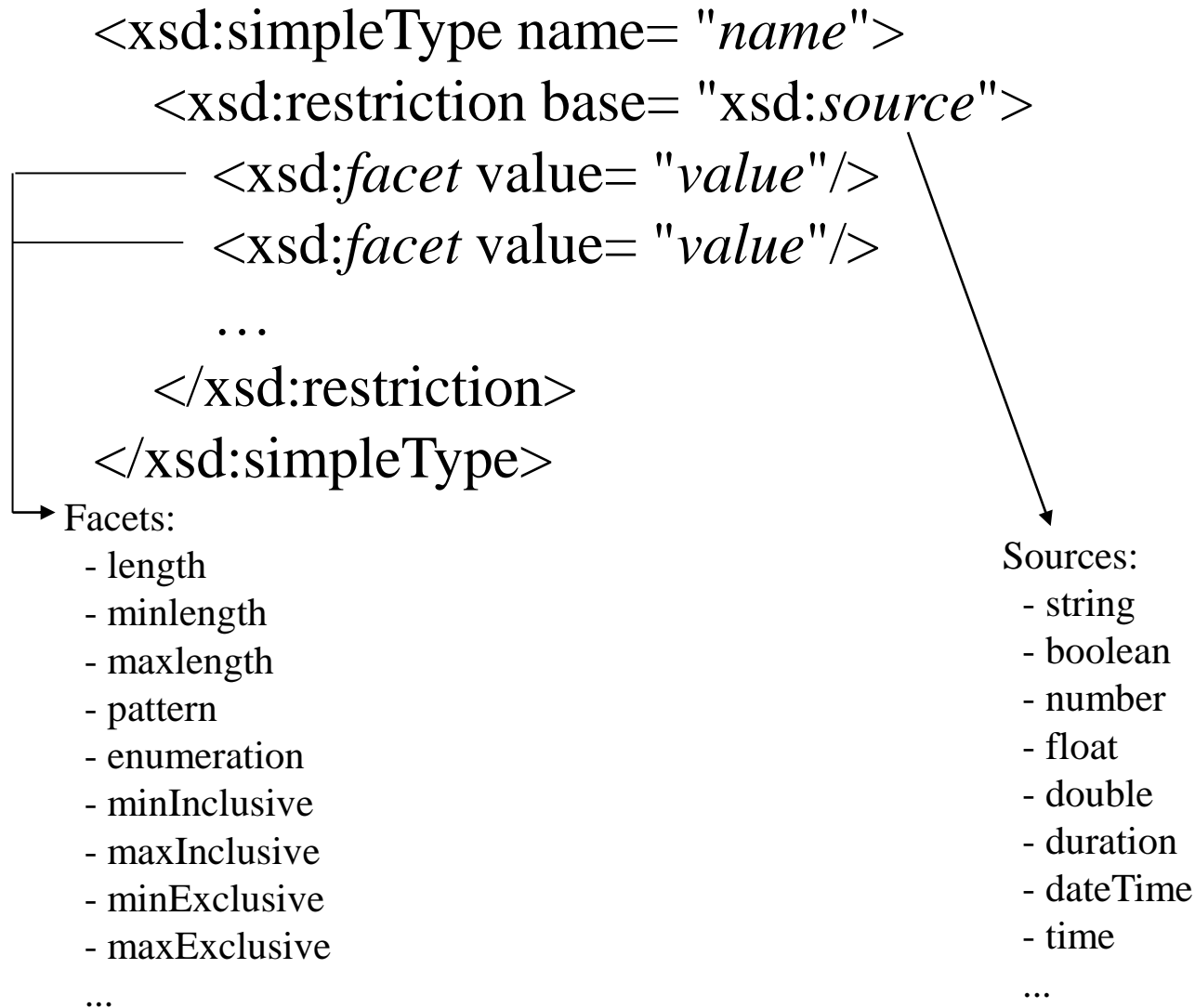
- The *integer* datatype has 8 optional facets:
 - totalDigits
 - pattern
 - whitespace
 - enumeration
 - maxInclusive
 - maxExclusive
 - minInclusive
 - minExclusive

Example

```
<xsd:simpleType name= "EarthSurfaceElevation">  
  <xsd:restriction base="xsd:integer">  
    <xsd:minInclusive value="-1290"/>  
    <xsd:maxInclusive value="29035"/>  
  </xsd:restriction>  
</xsd:simpleType>
```

This creates a new datatype called 'EarthSurfaceElevation'. Elements declared to be of this type can hold an integer. However, the integer is restricted to have a value between -1290 and 29035, inclusive.

General Form of Creating a New Datatype by Specifying Facet Values



Creating a simpleType from another simpleType

- Thus far we have created a simpleType using one of the built-in datatypes as our base type.
- However, we can create a simpleType that uses another simpleType as the base. See next slide.

```
<xsd:simpleType name= "EarthSurfaceElevation">  
  <xsd:restriction base="xsd:integer">  
    <xsd:minInclusive value="-1290"/>  
    <xsd:maxInclusive value="29035"/>  
  </xsd:restriction>  
</xsd:simpleType>
```

```
<xsd:simpleType name= "BostonAreaSurfaceElevation">  
  <xsd:restriction base="EarthSurfaceElevation">  
    <xsd:minInclusive value="0"/>  
    <xsd:maxInclusive value="120"/>  
  </xsd:restriction>  
</xsd:simpleType>
```

This simpleType uses *EarthSurfaceElevation* as its base type.

Fixing a Facet Value

- Sometimes when we define a simpleType we want to require that one (or more) facets have an unchanging value. That is, we want to make the facet a constant.

```
<xsd:simpleType name= "ClassSize">  
  <xsd:restriction base="xsd:nonNegativeInteger">  
    <xsd:minInclusive value="10" fixed="true"/>  
    <xsd:maxInclusive value="60"/>  
  </xsd:restriction>  
</xsd:simpleType>
```

simpleTypes which derive from this simpleType may not change this facet.

```

<xsd:simpleType name= "ClassSizefixed="true"/>
    <xsd:maxInclusive value="60"/>
  </xsd:restriction>
</xsd:simpleType>

```

```

<xsd:simpleType name= "BostonIEEEClassSize">
  <xsd:restriction base="ClassSize">
    <xsd:minInclusive value="15"/>
    <xsd:maxInclusive value="60"/>
  </xsd:restriction>
</xsd:simpleType>

```

Error! Cannot
change the value
of a fixed facet!

Element Containing a User-Defined Simple Type

Example. Create a schema element declaration for an elevation element.

Declare the elevation element to be an integer with a range -1290 to 29035

```
<elevation>5240</elevation>
```

Here's one way of declaring the elevation element:

```
<xsd:simpleType name="EarthSurfaceElevation">  
  <xsd:restriction base="xsd:integer">  
    <xsd:minInclusive value="-1290"/>  
    <xsd:maxInclusive value="29035"/>  
  </xsd:restriction>  
</xsd:simpleType>  
<xsd:element name="elevation" type="EarthSurfaceElevation"/>
```

Element Containing a User-Defined Simple Type (cont.)

Here's an alternative method for declaring elevation:

```
<xsd:element name="elevation">  
  <xsd:simpleType>  
    <xsd:restriction base="xsd:integer">  
      <xsd:minInclusive value="-1290"/>  
      <xsd:maxInclusive value="29035"/>  
    </xsd:restriction>  
  </xsd:simpleType>  
</xsd:element>
```

The simpleType definition is defined inline, it is an *anonymous* simpleType definition.

The disadvantage of this approach is that this simpleType may not be reused by other elements.

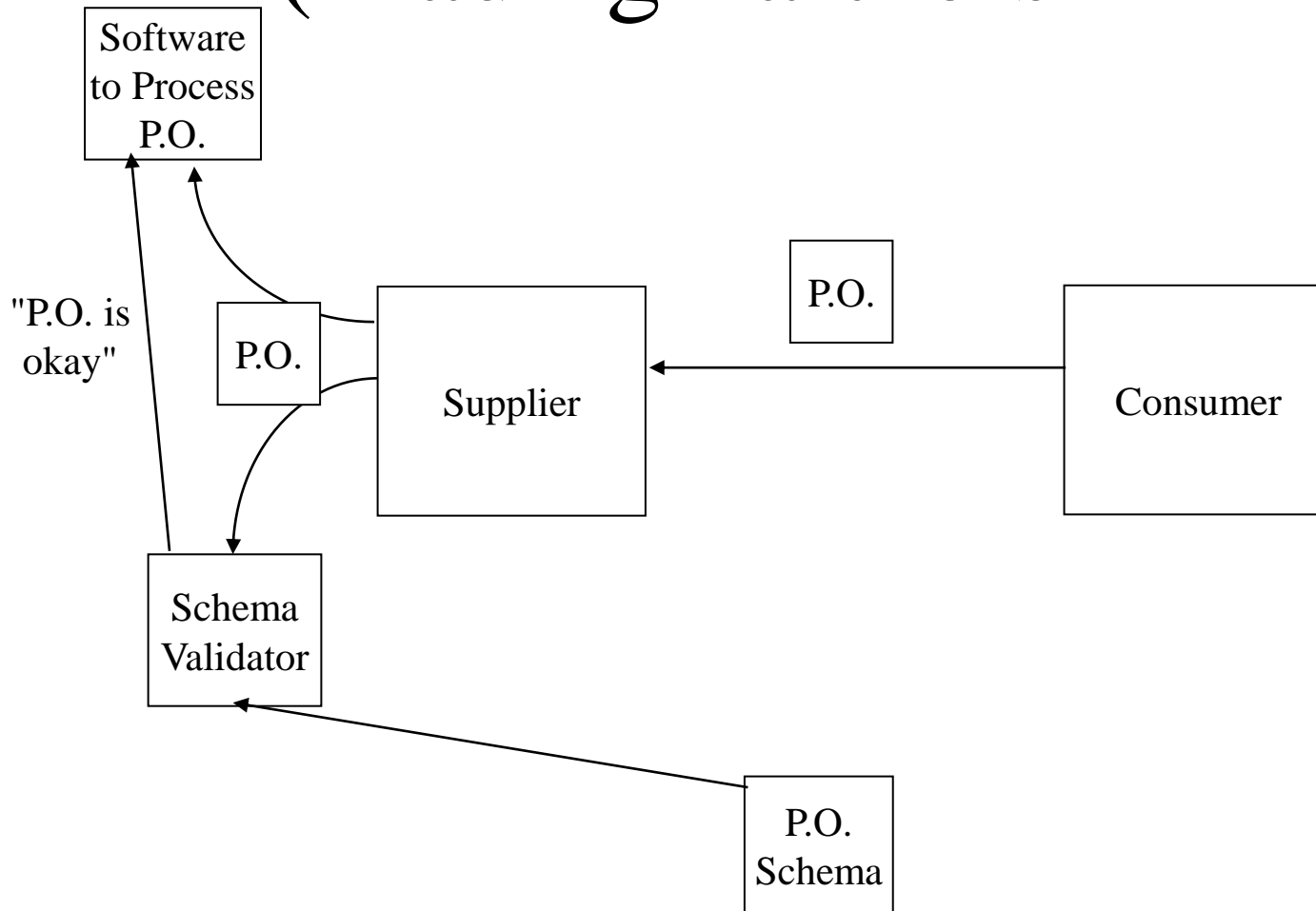
Summary of Declaring Elements (three ways to do it)

1 `<xsd:element name="name" type="type" minOccurs="int" maxOccurs="int"/>`

2 `<xsd:element name="name" minOccurs="int" maxOccurs="int">
 <xsd:complexType>
 ...
 </xsd:complexType>
</xsd:element>`

3 `<xsd:element name="name" minOccurs="int" maxOccurs="int">
 <xsd:simpleType>
 <xsd:restriction base="type">
 ...
 </xsd:restriction>
 </xsd:simpleType>
</xsd:element>`

Classic use of XML Schemas (Trading Partners - B2B)



(Schema at third-party, neutral web site)

What are XML Schemas?

- **Data Model**
 - With XML Schemas you specify how your XML data will be organized, and the datatypes of your data. That is, with XML Schemas you model how your data is to be represented in an instance document.
- **A Contract**
 - Organizations agree to structure their XML documents in conformance with an XML Schema. Thus, the XML Schema acts as a contract between the organizations.
- **A rich source of metadata**
 - An XML Schema document contains lots of data about the data in the XML instance documents, such as the datatypes of the data, the data's range of values, how the data is related to another piece of data (parent/child, sibling relationship), i.e., XML Schemas contain metadata

Regular Expressions

- Recall that the string datatype has a pattern facet. The value of a pattern facet is a regular expression. Below are some examples of regular expressions:

Regular Expression

- Chapter \d
- a*b
- [xyz]b
- a?b
- a+b
- [a-c]x

Example

- Chapter 1
- b, ab, aab, aaab, ...
- xb, yb, zb
- b, ab
- ab, aab, aaab, ...
- ax, bx, cx

Regular Expressions (cont.)

- Regular Expression

- `[a-c]x`
- `[-ac]x`
- `[ac-]x`
- `[^0-9]x`
- `(ho){2}` there
- `(ho\s){2}` there
- `(a|b)+x`

- Example

- `ax, bx, cx`
- `-x, ax, cx`
- `ax, cx, -x`
- any non-digit char followed by `x`
- `hoho` there
- `ho ho` there
- `ax, bx, aax, bbx, abx, bax,...`

Regular Expressions (concluded)

- `\p{L}` • A letter, from any language
- `\p{Lu}` • An uppercase letter, from any language
- `\p{Ll}` • A lowercase letter, from any language
- `\p{N}` • A number - Roman, fractions, etc
- `\p{Nd}` • A digit from any language
- `\p{P}` • A punctuation symbol (the *p* is often left out e.g. `\.` for `.`)
- `\p{Sc}` • A currency sign, from any language

```
<xsd:simpleType name="money">
  <xsd:restriction base="xsd:string">
    <xsd:pattern value="\p{Sc}\p{Nd}+(\.\p{Nd}\p{Nd})?" />
  </xsd:restriction>
</xsd:simpleType>
<xsd:element name="cost" type="money" />
```

"currency sign from any language, followed by one or more digits from any language, optionally followed by a period and two digits from any language"

```
<cost>$45.99</cost>
<cost>¥300</cost>
```

Example R.E.

$[1-9]?[0-9]|1[0-9][0-9]|2[0-4][0-9]|25[0-5]$

0 to 99	100 to 199	200 to 249	250 to 255
---------	------------	------------	------------

This regular expression restricts a *string* to have values between 0 and 255.

... Such a R.E. might be useful in describing an IP address ...

IP Datatype Definition

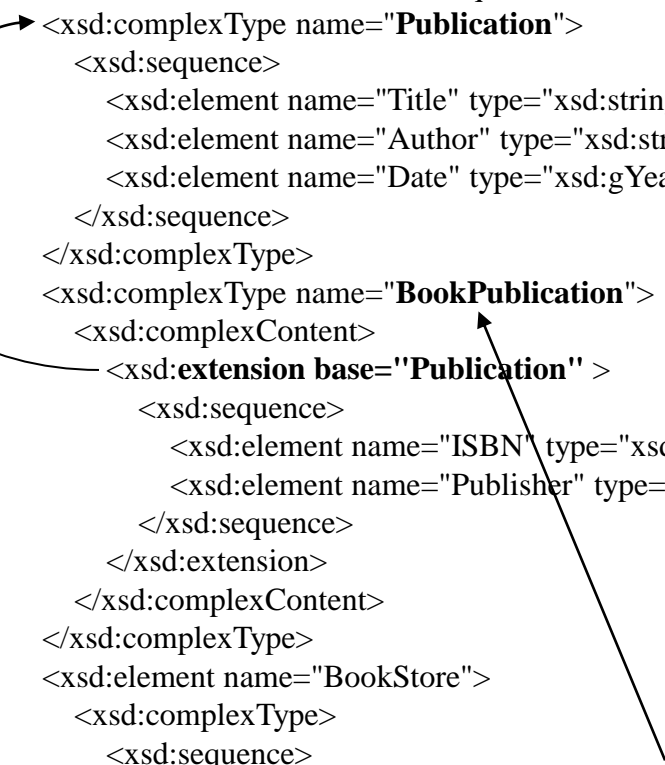
```
<xsd:simpleType name="IP">
  <xsd:restriction base="xsd:string">
    <xsd:pattern value="(([1-9]?[0-9]|1[0-9]|2[0-4][0-9]|25[0-5])\.){3}
      ([1-9]?[0-9]|1[0-9]|2[0-4][0-9]|25[0-5])">
    </xsd:pattern>
  </xsd:restriction>
</xsd:simpleType>
```

Datatype for representing IP addresses. Examples 129.83.64.255 and 64.128.2.71

This datatype restricts each field of the IP address to have a value between zero and 255, i.e. [0-255].[0-255].[0-255].[0-255]

Derived Types

- We can do a form of subclassing with complexType definitions. We call this "derived types"
 - derive by extension: extend the parent complexType with more elements
 - derive by restriction: create a type which is a subset of the base type. There are two ways to subset the elements:
 - redefine a base type element to have a **restricted range of values**, or
 - redefine a base type element to have a more **restricted number of occurrences**.



```

<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.books.org"
  xmlns="http://www.books.org"
  elementFormDefault="qualified">
  <xsd:complexType name="Publication">
    <xsd:sequence>
      <xsd:element name="Title" type="xsd:string" maxOccurs="unbounded"/>
      <xsd:element name="Author" type="xsd:string" maxOccurs="unbounded"/>
      <xsd:element name="Date" type="xsd:gYear"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="BookPublication">
    <xsd:complexContent>
      <xsd:extension base="Publication" >
        <xsd:sequence>
          <xsd:element name="ISBN" type="xsd:string"/>
          <xsd:element name="Publisher" type="xsd:string"/>
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
  <xsd:element name="BookStore">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="Book" type="BookPublication" maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>


```

Note that
BookPublication extends
the *Publication*
type, i.e., we are doing
Derive by Extension

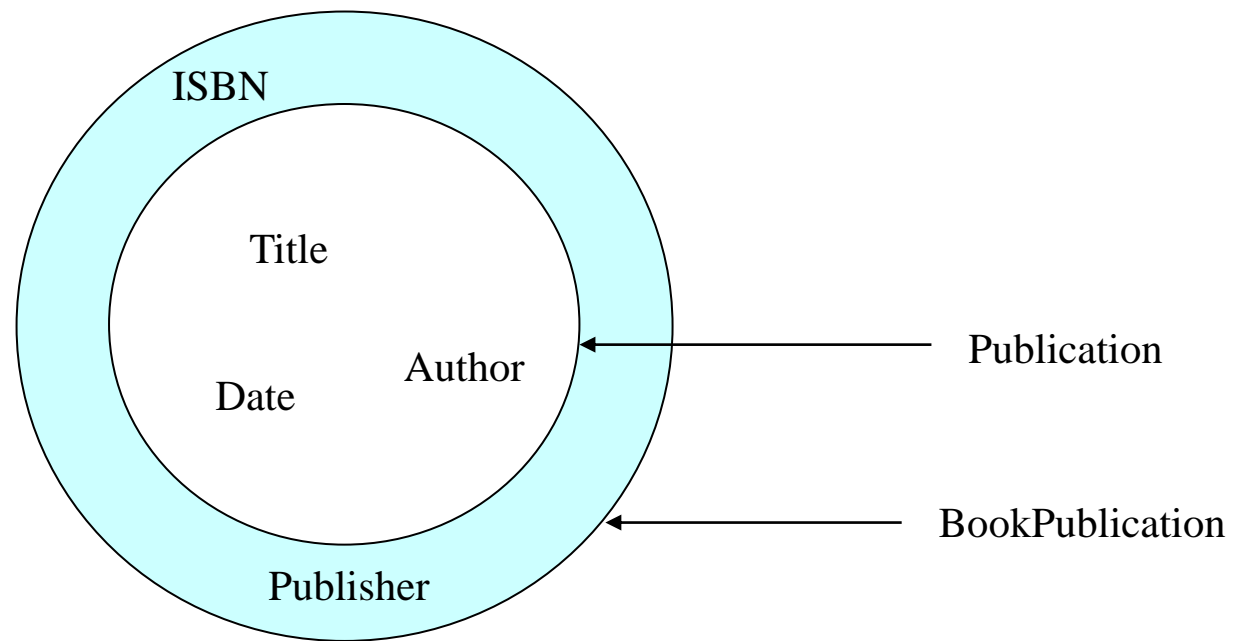
```

<xsd:complexType name="Publication">
  <xsd:sequence>
    <xsd:element name="Title" type="xsd:string" maxOccurs="unbounded"/>
    <xsd:element name="Author" type="xsd:string" maxOccurs="unbounded"/>
    <xsd:element name="Date" type="xsd:gYear"/>
  </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="BookPublication">
  <xsd:complexContent>
    <xsd:extension base="Publication">
      <xsd:sequence>
        <xsd:element name="ISBN" type="xsd:string"/>
        <xsd:element name="Publisher" type="xsd:string"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

```



Elements declared to be of type *BookPublication* will have 5 child elements - *Title*, *Author*, *Date*, *ISBN*, and *Publisher*. Note that the elements in the derived type are **appended** to the elements in the base type.



Derive by Restriction

```

<xsd:complexType name="Publication">
  <xsd:sequence>
    <xsd:element name="Title" type="xsd:string" maxOccurs="unbounded"/>
    <xsd:element name="Author" type="xsd:string" maxOccurs="unbounded"/>
    <xsd:element name="Date" type="xsd:gYear"/>
  </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="SingleAuthorPublication">
  <xsd:complexContent>
    <xsd:restriction base="Publication">
      <xsd:sequence>
        <xsd:element name="Title" type="xsd:string" maxOccurs="unbounded"/>
        <xsd:element name="Author" type="xsd:string"/>
        <xsd:element name="Date" type="xsd:gYear"/>
      </xsd:sequence>
    </xsd:restriction>
  </xsd:complexContent>
</xsd:complexType>

```

Elements of type *SingleAuthorPublication* will have 3 child elements - *Title*, *Author*, and *Date*. However, there must be exactly one *Author* element. Note that in the restriction type you must repeat all the declarations from the base type (except when the base type has an element with `minOccurs="0"` and the subtype wishes to delete it. See next slide).

Deleting an element in the base type

```

<xsd:complexType name="Publication">
  <xsd:sequence>
    <xsd:element name="Title" type="xsd:string" maxOccurs="unbounded"/>
    <xsd:element name="Author" type="xsd:string" minOccurs="0"/>
    <xsd:element name="Date" type="xsd:gYear"/>
  </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="ZeroAuthorPublication">
  <xsd:complexContent>
    <xsd:restriction base="Publication">
      <xsd:sequence>
        <xsd:element name="Title" type="xsd:string" maxOccurs="unbounded"/>
        <xsd:element name="Date" type="xsd:gYear"/>
      </xsd:sequence>
    </xsd:restriction>
  </xsd:complexContent>
</xsd:complexType>

```

Note that in this subtype we have eliminated the *Author* element, i.e., the subtype is just comprised of an unbounded number of *Title* elements followed by a single *Date* element.

If the base type has an element with minOccurs="0", and the subtype wishes to not have that element, then it can simply leave it out.

Prohibiting Derivations

- Sometimes we may want to create a type and disallow all derivations of it, or just disallow extension derivations, or disallow restriction derivations.
 - Rationale: "For example, I may create a complexType and make it publicly available for others to use. However, I don't want them to extend it with their proprietary extensions or subset it to remove, say, copyright information." (Jon Cleaver)

`<xsd:complexType name="Publication" final="#all" ...>` Publication cannot be extended nor restricted

`<xsd:complexType name="Publication" final="restriction" ...>` Publication cannot be restricted

`<xsd:complexType name="Publication" final="extension" ...>` Publication cannot be extended

Terminology: Global versus Local

- Global element declarations, global type definitions:
 - These are element declarations/type definitions that are immediate children of <schema>
- Local element declarations, local type definitions:
 - These are element declarations/type definitions that are nested within other elements/types.


```

<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.books.org"
  xmlns="http://www.books.org"
  elementFormDefault="qualified">
  <xsd:complexType name="Publication">
    <xsd:sequence>
      <xsd:element name="Title" type="xsd:string" maxOccurs="unbounded"/>
      <xsd:element name="Author" type="xsd:string" maxOccurs="unbounded"/>
      <xsd:element name="Date" type="xsd:gYear"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="BookPublication">
    <xsd:complexContent>
      <xsd:extension base="Publication" >
        <xsd:sequence>
          <xsd:element name="ISBN" type="xsd:string"/>
          <xsd:element name="Publisher" type="xsd:string"/>
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
  <xsd:element name="BookStore">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="Book" type="BookPublication" maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>

```

Global type definition

Global type definition

Global element declaration

Local type definition

Local element declarations

Global vs Local ... What's the Big Deal?

- So what if an element or type is global or local. What practical impact does it have?
 - Answer: **only global elements/types can be referenced (i.e., reused)**. Thus, if an element/type is local then it is effectively invisible to the rest of the schema (and to other schemas).

Attributes

- The next slide shows a version of the BookStore DTD that uses attributes. Then, the following slide shows how this is implemented using XML Schemas.

```
<!ELEMENT BookStore (Book+)>
<!ELEMENT Book (Title, Author, Date, ISBN, Publisher)>
<!ATTLIST Book
    Category (autobiography | non-fiction | fiction) #REQUIRED
    InStock (true | false) "false"
    Reviewer CDATA " ">
<!ELEMENT Title (#PCDATA)>
<!ELEMENT Author (#PCDATA)>
<!ELEMENT Date (#PCDATA)>
<!ELEMENT ISBN (#PCDATA)>
<!ELEMENT Publisher (#PCDATA)>
```

BookStore.dtd

```

<xsd:element name="BookStore">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="Book" maxOccurs="unbounded">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="Title" type="xsd:string"/>
            <xsd:element name="Author" type="xsd:string"/>
            <xsd:element name="Date" type="xsd:string"/>
            <xsd:element name="ISBN" type="xsd:string"/>
            <xsd:element name="Publisher" type="xsd:string"/>
          </xsd:sequence>
          <xsd:attributeGroup ref="BookAttributes"/>
        </xsd:complexType>
      </xsd:element>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
<xsd:attributeGroup name="BookAttributes">
  <xsd:attribute name="Category" use="required">
    <xsd:simpleType>
      <xsd:restriction base="xsd:string">
        <xsd:enumeration value="autobiography"/>
        <xsd:enumeration value="non-fiction"/>
        <xsd:enumeration value="fiction"/>
      </xsd:restriction>
    </xsd:simpleType>
  </xsd:attribute>
  <xsd:attribute name="InStock" type="xsd:boolean" default="false"/>
  <xsd:attribute name="Reviewer" type="xsd:string" default=" " />
</xsd:attributeGroup>

```

→ Category (autobiography | non-fiction | fiction) #REQUIRED

→ InStock (true | false) "false"

→ Reviewer CDATA " "

```
<xsd:attribute name="Category" use="required">  
  <xsd:simpleType>  
    <xsd:restriction base="xsd:string">  
      <xsd:enumeration value="autobiography"/>  
      <xsd:enumeration value="non-fiction"/>  
      <xsd:enumeration value="fiction"/>  
    </xsd:restriction>  
  </xsd:simpleType>  
</xsd:attribute>
```

"Instance documents are required to have the Category attribute (as indicated by use="required"). The value of Category must be either autobiography, non-fiction, or fiction (as specified by the enumeration facets)."

Note: attributes can only have simpleTypes (i.e., attributes cannot have child elements).

Summary of Declaring Attributes (two ways to do it)

1

```
<xsd:attribute name="name" type="simple-type" use="how-its-used" default/fixed="value"/>
```

↑
xsd:string
xsd:integer
xsd:boolean
...

↑
required
optional
...

└─┬─┘
│
The "use" attribute must be
optional if you use
default or fixed.

2

```
<xsd:attribute name="name" use="how-its-used" default/fixed="value">
```

```
  <xsd:simpleType>
```

```
    <xsd:restriction base="simple-type">
```

```
      <xsd:facet value="value"/>
```

```
      ...
```

```
    </xsd:restriction>
```

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

```
</xsd:attribute>
```

use --> use it only with Local Attribute Declarations

- The "use" attribute only makes sense in the context of an element declaration. Example: "for each Book element, the Category attribute is required".
- When declaring a global attribute do not specify a "use"


```
<xsd:element name="Book">
  <xsd:complexType>
    <xsd:sequence>
      ...
    </xsd:sequence>
    <xsd:attribute ref="Category" use="required"/>
    ...
  </xsd:complexType>
</xsd:element>
<xsd:attribute name="Category">
  <xsd:simpleType>
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="autobiography"/>
      <xsd:enumeration value="fiction"/>
      <xsd:enumeration value="non-fiction"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:attribute>
```

← Local attribute declaration. Use the "use" attribute here.

← Global attribute declaration. Must NOT have a "use" ("use" only makes sense in the context of an element)

Inlining Attributes

- On the next slide is another way of expressing the last example - the attributes are inlined within the Book declaration rather than being separately defined in an attributeGroup.

```
<xsd:element name="Book" maxOccurs="unbounded">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="Title" type="xsd:string"/>
      <xsd:element name="Author" type="xsd:string" maxOccurs="unbounded"/>
      <xsd:element name="Date" type="xsd:string"/>
      <xsd:element name="ISBN" type="xsd:string"/>
      <xsd:element name="Publisher" type="xsd:string"/>
    </xsd:sequence>
    <xsd:attribute name="Category" use="required">
      <xsd:simpleType>
        <xsd:restriction base="xsd:string">
          <xsd:enumeration value="autobiography"/>
          <xsd:enumeration value="non-fiction"/>
          <xsd:enumeration value="fiction"/>
        </xsd:restriction>
      </xsd:simpleType>
    </xsd:attribute>
    <xsd:attribute name="InStock" type="xsd:boolean" default="false"/>
    <xsd:attribute name="Reviewer" type="xsd:string" default="" />
  </xsd:complexType>
</xsd:element>
```

Notes about Attributes

- The attribute declarations always come last, after the element declarations.
- *The attributes are always with respect to the element that they are defined (nested) within.*

"bar and boo are
attributes of foo"

```
<xsd:element name="foo">
  <xsd:complexType>
    <xsd:sequence>
      ...
    </xsd:sequence>
    <xsd:attribute name="bar" .../>
    <xsd:attribute name="boo" .../>
  </xsd:complexType>
</xsd:element>
```

These attributes
apply to the
element they are
nested within (Book)
That is, Book has three
attributes - Category,
InStock, and Reviewer.

```

<xsd:element name="Book">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="Title" type="xsd:string"/>
      <xsd:element name="Author" type="xsd:string" maxOccurs="unbounded"/>
      <xsd:element name="Date" type="xsd:string"/>
      <xsd:element name="ISBN" type="xsd:string"/>
      <xsd:element name="Publisher" type="xsd:string"/>
    </xsd:sequence>
    <xsd:attribute name="Category" use="required">
      <xsd:simpleType>
        <xsd:restriction base="xsd:string">
          <xsd:enumeration value="autobiography"/>
          <xsd:enumeration value="non-fiction"/>
          <xsd:enumeration value="fiction"/>
        </xsd:restriction>
      </xsd:simpleType>
    </xsd:attribute>
    <xsd:attribute name="InStock" type="xsd:boolean" default="false"/>
    <xsd:attribute name="Reviewer" type="xsd:string" default=" "/>
  </xsd:complexType>
</xsd:element>

```

Summary of Declaring Elements

1. Element with Simple Content.

Declaring an element using a **built-in type**:

```
<xsd:element name="numStudents" type="xsd:positiveInteger"/>
```

Declaring an element using a **user-defined simpleType**:

```
<xsd:simpleType name="shapes">  
  <xsd:restriction base="xsd:string">  
    <xsd:enumeration value="triangle"/>  
    <xsd:enumeration value="rectangle"/>  
    <xsd:enumeration value="square"/>  
  </xsd:restriction>  
</xsd:simpleType>  
<xsd:element name="geometry" type="shapes"/>
```

An alternative formulation of the above shapes example is to **inline the simpleType** definition:

```
<xsd:element name="geometry">  
  <xsd:simpleType>  
    <xsd:restriction base="xsd:string">  
      <xsd:enumeration value="triangle"/>  
      <xsd:enumeration value="rectangle"/>  
      <xsd:enumeration value="square"/>  
    </xsd:restriction>  
  </xsd:simpleType>  
</xsd:element>
```

Summary of Declaring Elements (cont.)

2. Element Contains Child Elements

Defining the child elements **inline**:

```
<xsd:element name="Person">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="Title" type="xsd:string"/>
      <xsd:element name="FirstName" type="xsd:string"/>
      <xsd:element name="Surname" type="xsd:string"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

An alternate formulation of the above Person example is to create a **named complexType** and then use that type:

```
<xsd:complexType name="PersonType">
  <xsd:sequence>
    <xsd:element name="Title" type="xsd:string"/>
    <xsd:element name="FirstName" type="xsd:string"/>
    <xsd:element name="Surname" type="xsd:string"/>
  </xsd:sequence>
</xsd:complexType>
<xsd:element name="Person" type="PersonType"/>
```

Summary of Declaring Elements (cont.)

3. Element Contains a complexType that is an Extension of another complexType

```
<xsd:complexType name="Publication">
  <xsd:sequence>
    <xsd:element name="Title" type="xsd:string" maxOccurs="unbounded"/>
    <xsd:element name="Author" type="xsd:string" maxOccurs="unbounded"/>
    <xsd:element name="Date" type="xsd:gYear"/>
  </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="BookPublication">
  <xsd:complexContent>
    <xsd:extension base="Publication" >
      <xsd:sequence>
        <xsd:element name="ISBN" type="xsd:string"/>
        <xsd:element name="Publisher" type="xsd:string"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
<xsd:element name="Book" type="BookPublication"/>
```


Summary of Declaring Elements (cont.)

4. Element Contains a complexType that is a Restriction of another complexType

```
<xsd:complexType name="Publication">
  <xsd:sequence>
    <xsd:element name="Title" type="xsd:string" maxOccurs="unbounded"/>
    <xsd:element name="Author" type="xsd:string" maxOccurs="unbounded"/>
    <xsd:element name="Date" type="xsd:gYear"/>
  </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="SingleAuthorPublication">
  <xsd:complexContent>
    <xsd:restriction base="Publication">
      <xsd:sequence>
        <xsd:element name="Title" type="xsd:string" maxOccurs="unbounded"/>
        <xsd:element name="Author" type="xsd:string"/>
        <xsd:element name="Date" type="xsd:gYear"/>
      </xsd:sequence>
    </xsd:restriction>
  </xsd:complexContent>
</xsd:complexType>
<xsd:element name="Catalogue" type="SingleAuthorPublication"/>
```

Expressing Alternates

DTD: `<!ELEMENT transportation (train | plane | automobile)>`

XML Schema:

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.travel.org"
  xmlns="http://www.travel.org"
  elementFormDefault="qualified">
  <xsd:element name="transportation">
    <xsd:complexType>
      <xsd:choice>
        <xsd:element name="train" type="xsd:string"/>
        <xsd:element name="plane" type="xsd:string"/>
        <xsd:element name="automobile" type="xsd:string"/>
      </xsd:choice>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

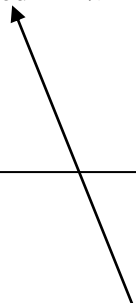
Note: the choice is an exclusive-or, that is, transportation can contain only **one** element - either train, or plane, or automobile.

Expressing Repeatable Choice

DTD: `<!ELEMENT binary-string (zero | one)*>`

XML Schema:

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.binary.org"
  xmlns="http://www.binary.org"
  elementFormDefault="qualified">
  <xsd:element name="binary-string">
    <xsd:complexType>
      <xsd:choice minOccurs="0" maxOccurs="unbounded">
        <xsd:element name="zero" type="xsd:unsignedByte" fixed="0"/>
        <xsd:element name="one" type="xsd:unsignedByte" fixed="1"/>
      </xsd:choice>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```



Notes:

1. An element can fix its value, using the fixed attribute.
2. When you don't specify a value for minOccurs, it defaults to "1".
Same for maxOccurs. See the last example (transportation) where we used a <choice> element with no minOccurs or maxOccurs.

fixed/default Element Values

- When you declare an element, you can give it a fixed or default value.
 - Then, in the instance document, you can leave the element empty.

```
<element name="zero" fixed="0"/>
```

```
...
```

```
<zero>0</zero>
```

or equivalently:

```
<zero/>
```

```
<element name="color" default="red"/>
```

```
...
```

```
<color>red</color>
```

or equivalently:

```
<color/>
```

Using <sequence> and <choice>

DTD:

```
<!ELEMENT life ((work, eat)*, (work | play), sleep)* >
```

XML Schema:

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.life.org"
  xmlns="http://www.life.org"
  elementFormDefault="qualified">
  <xsd:element name="life">
    <xsd:complexType>
      <xsd:sequence minOccurs="0" maxOccurs="unbounded">
        <xsd:sequence minOccurs="0" maxOccurs="unbounded">
          <xsd:element name="work" type="xsd:string"/>
          <xsd:element name="eat" type="xsd:string"/>
        </xsd:sequence>
        <xsd:choice>
          <xsd:element name="work" type="xsd:string"/>
          <xsd:element name="play" type="xsd:string"/>
        </xsd:choice>
        <xsd:element name="sleep" type="xsd:string"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

Summary of Defining simpleTypes

1. simpleType that uses a built-in base type:

```
<xsd:simpleType name= "EarthSurfaceElevation">  
  <xsd:restriction base="xsd:integer">  
    <xsd:minInclusive value="-1290"/>  
    <xsd:maxInclusive value="29035"/>  
  </xsd:restriction>  
</xsd:simpleType>
```

2. simpleType that uses another simpleType as the base type:

```
<xsd:simpleType name= "BostonSurfaceElevation">  
  <xsd:restriction base="EarthSurfaceElevation">  
    <xsd:minInclusive value="0"/>  
    <xsd:maxInclusive value="120"/>  
  </xsd:restriction>  
</xsd:simpleType>
```

any Element

- The `<any>` element enables the instance document author to extend the document with elements not specified by the schema.

```
<xsd:element name="Book">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="Title" type="xsd:string"/>
      <xsd:element name="Author" type="xsd:string"/>
      <xsd:element name="Date" type="xsd:string"/>
      <xsd:element name="ISBN" type="xsd:string"/>
      <xsd:element name="Publisher" type="xsd:string"/>
      <xsd:any minOccurs="0"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Now an instance document author can optionally extend (after `<Publisher>`) the content of `<Book>` elements with any element.

```

<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            targetNamespace="http://www.repository.org"
            xmlns="http://www.repository.org"
            elementFormDefault="qualified">
  <xsd:element name="Reviewer">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="Name">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name="First" type="xsd:string"/>
              <xsd:element name="Last" type="xsd:string"/>
            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>

```

SchemaRepository.xsd

Suppose that the instance document author discovers this schema repository, and wants to extend the <Book> elements with a <Reviewer> element. This can be done - thus, the instance document will be extended with an element never anticipated by the schema author.


```

<?xml version="1.0"?>
<BookStore xmlns="http://www.BookRetailers.org"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation=
    "http://www. BookRetailers.org
    BookSeller.xsd
    http://www. repository.org
    SchemaRepository.xsd">

  <Book>
    <Title>My Life and Times</Title>
    <Author>Paul McCartney</Author>
    <Date>1998</Date>
    <ISBN>94303-12021-43892</ISBN>
    <Publisher>McMillin Publishing</Publisher>
    <Reviewer xmlns="http://www.repository.org">
      <Name>
        <First>Roger</First>
        <Last>Costello</Last>
      </Name>
    </Reviewer>
  </Book>
  <Book>
    <Title>Illusions: The Adventures of a Reluctant Messiah</Title>
    <Author>Richard Bach</Author>
    <Date>1977</Date>
    <ISBN>0-440-34319-4</ISBN>
    <Publisher>Dell Publishing Co.</Publisher>
  </Book>
</BookStore>

```

This instance document
uses components from
two different schemas.

Extensible Instance Documents

- The <any> element enables instance document authors to create instance documents containing elements above and beyond what was specified by the schema. The instance documents are said to be *extensible*. Contrast this schema with previous schemas where the content of all our elements were always fixed and *static*.
- We are empowering the instance document author with the ability to define what data makes sense to them.