XML Schemas

Derived from
http://www.w3.org/TR/xmlschema-0/

What is XML Schemas?

- Answer: An XML vocabulary for expressing your data's business rules

Example

```
<location>
        <latitude>32.904237</latitude>
        <longitude>73.620290</longitude>
        <uncertainty units="meters">2</uncertainty>
</location>
```

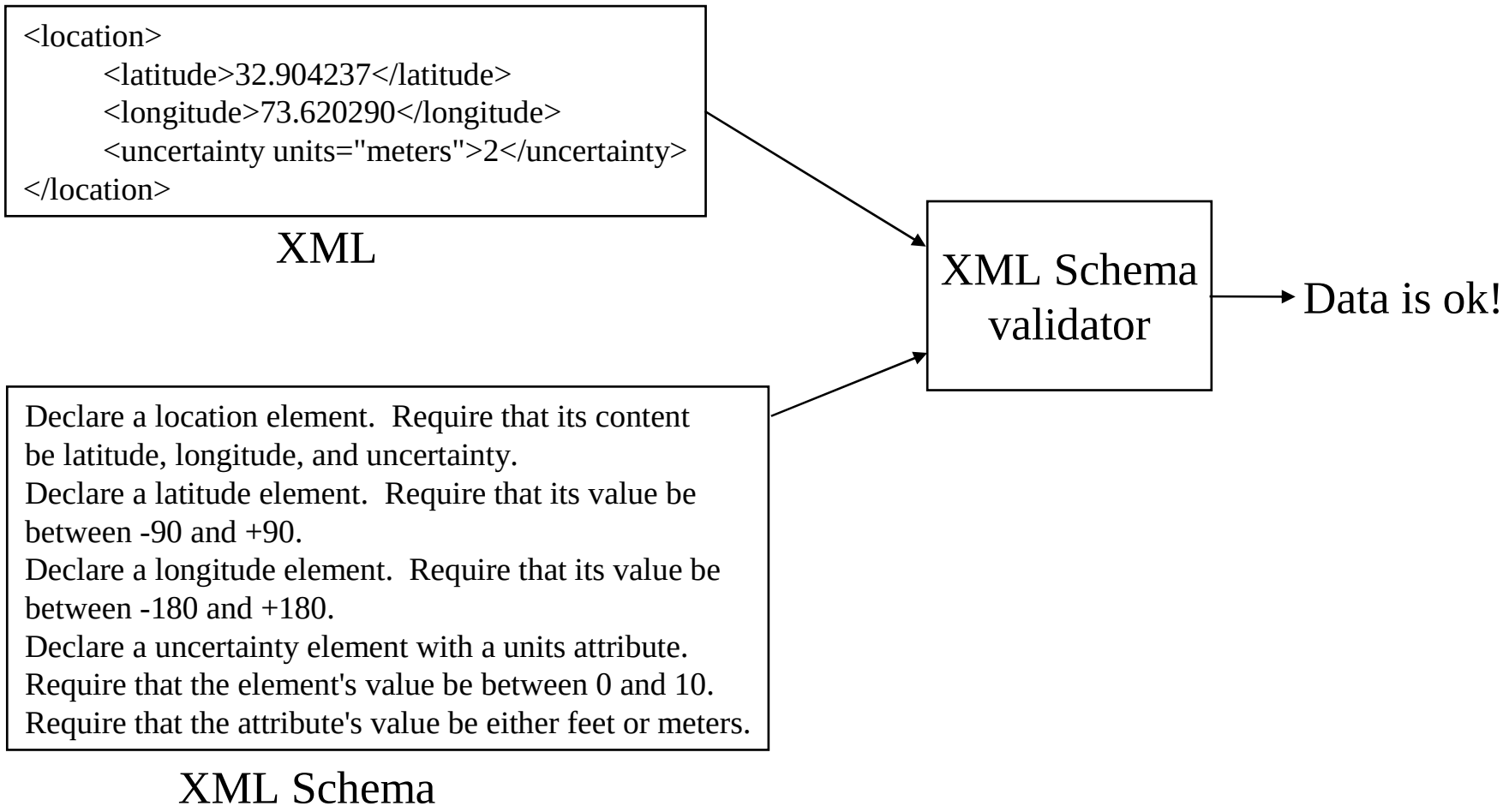
Is this data valid?

To be valid, it must meet these **constraints (data business rules)**:

1. The location must be comprised of a latitude, followed by a longitude, followed by an indication of the uncertainty of the lat/lon measurements.
2. The latitude must be a decimal with a value between -90 to +90
3. The longitude must be a decimal with a value between -180 to +180
4. For both latitude and longitude the number of digits to the right of the decimal point must be exactly six digits.
5. The value of uncertainty must be a non-negative integer
6. The uncertainty units must be either meters or feet.

We can express all these data constraints using XML Schemas

Validating your data



Schema Validators

- Command Line Only
  - XSV by Henry Thompson
    - ftp://ftp.cogsci.ed.ac.uk/pub/XSV/XSV12.EXE
- Has a Programmatic API
  - xerces by Apache
    - http://www.apache.org/xerces-j/index.html
  - IBM Schema Quality Checker (Note: this tool is only used to check your schema. It cannot be used to validate an instance document against a schema.)
    - http://www.alphaworks.ibm.com/tech/xmlsqc
  - MSXML4.0
    - http://www.microsoft.com
- GUI Oriented
  - XML Spy
    - http://www.xmlspy.com
  - Turbo XML
    - http://www.extensibility.com

What does an XML Schema accomplish?

Declare a location element. Require that its content be latitude, longitude, and uncertainty.
Declare a latitude element. Require that its value be between -90 and +90.
Declare a longitude element. Require that its value be between -180 and +180.
Declare a uncertainty element with a units attribute. Require that the element's value be between 0 and 10. Require that the attribute's value be either feet or meters.

XML Schema

Answer:
It creates an XML vocabulary:

<location>, <latitude>, <longitude>, <uncertainty>

It specifies the contents of each element, and the restrictions on the content.

It does one more thing ...

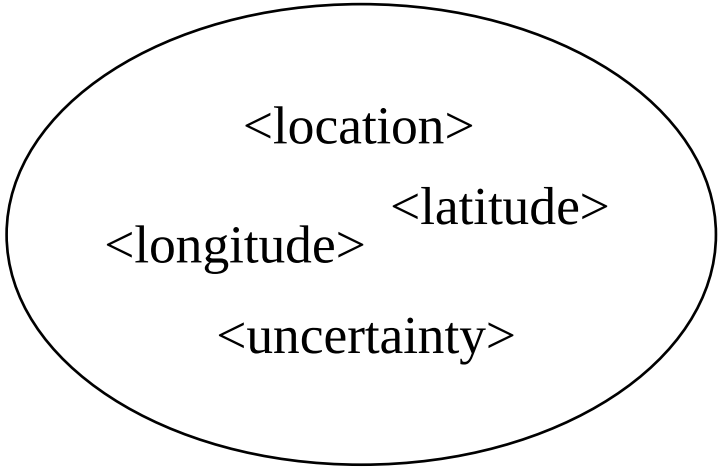
Namespace = http://www.example.org/target
Declare a location element. Require that its content be latitude, longitude, and uncertainty.
Declare a latitude element. Require that its value be between -90 and +90.
Declare a longitude element. Require that its value be between -180 and +180.
Declare a uncertainty element with a units attribute. Require that the element's value be between 0 and 10. Require that the attribute's value be either feet or meters.

XML Schema

An XML Schema specifies that the XML vocabulary that is being created shall be in a "namespace"

http://www.example.org/target Namespace

http://www.example.org/target



Purpose of XML Schemas (and DTDs)

- Specify:
  - the *structure* of instance documents
    - "this element contains these elements, which contains these other elements, etc"
  - the *datatype* of each element/attribute
    - "this element shall hold an integer with the range 0 to 12,000" (DTDs don't do too well with specifying datatypes like this)

Motivation for XML Schemas

- People are dissatisfied with DTDs
  - It's a different syntax
    - You write your XML (instance) document using one syntax and the DTD using another syntax --> bad, inconsistent
  - Limited datatype capability
    - DTDs support a very limited capability for specifying datatypes. You can't, for example, express "I want the <elevation> element to hold an integer with a range of 0 to 12,000"
    - Desire a set of datatypes compatible with those found in databases
      - DTD supports 10 datatypes; XML Schemas supports 44+ datatypes

Highlights of XML Schemas

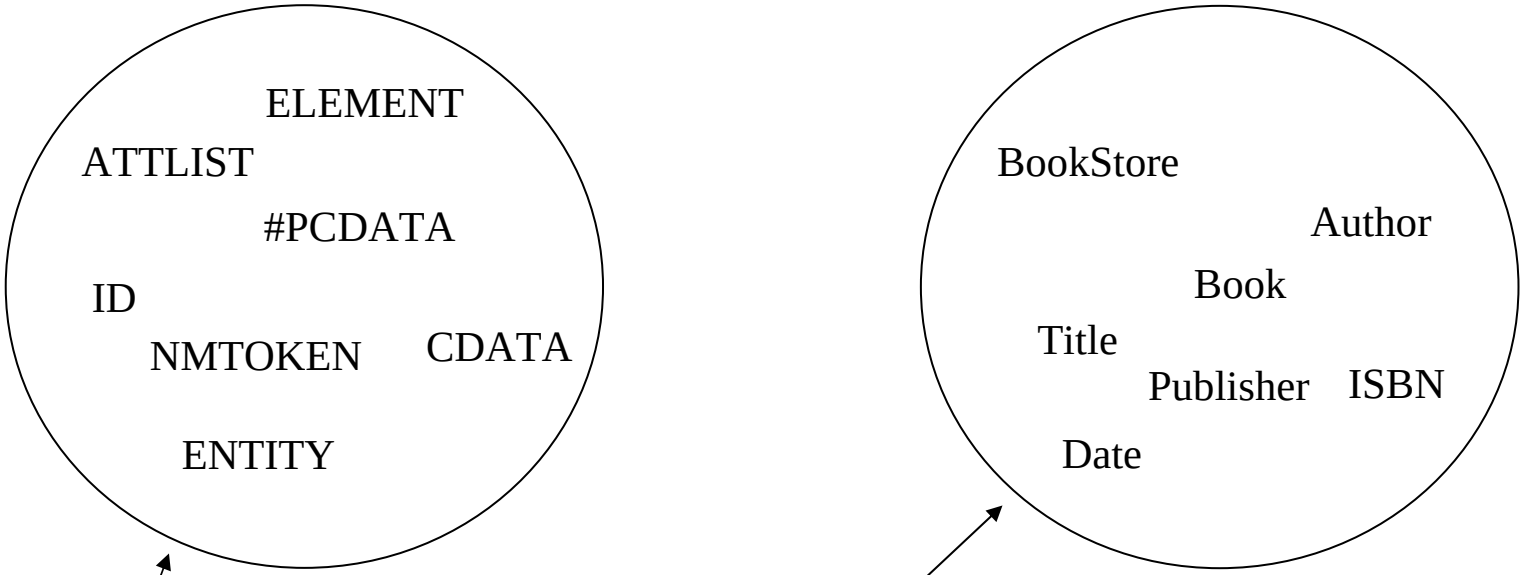
- XML Schemas are a tremendous advancement over DTDs:
  - Enhanced datatypes
    - 44+ versus 10
    - Can create your own datatypes
      - Example: "This is a new type based on the string type and elements of this type must follow this pattern: ddd-dddd, where 'd' represents a digit".
  - Written in the same syntax as instance documents
    - less syntax to remember
  - Object-oriented'ish
    - Can extend or restrict a type (derive new type definitions on the basis of old ones)
  - Can express sets, i.e., can define the child elements to occur in any order
  - Can specify element content as being unique (keys on content) and uniqueness within a region
  - Can define multiple elements with the same name but different content
  - Can define elements with nil content
  - Can define substitutable elements - e.g., the "Book" element is substitutable for the "Publication" element.

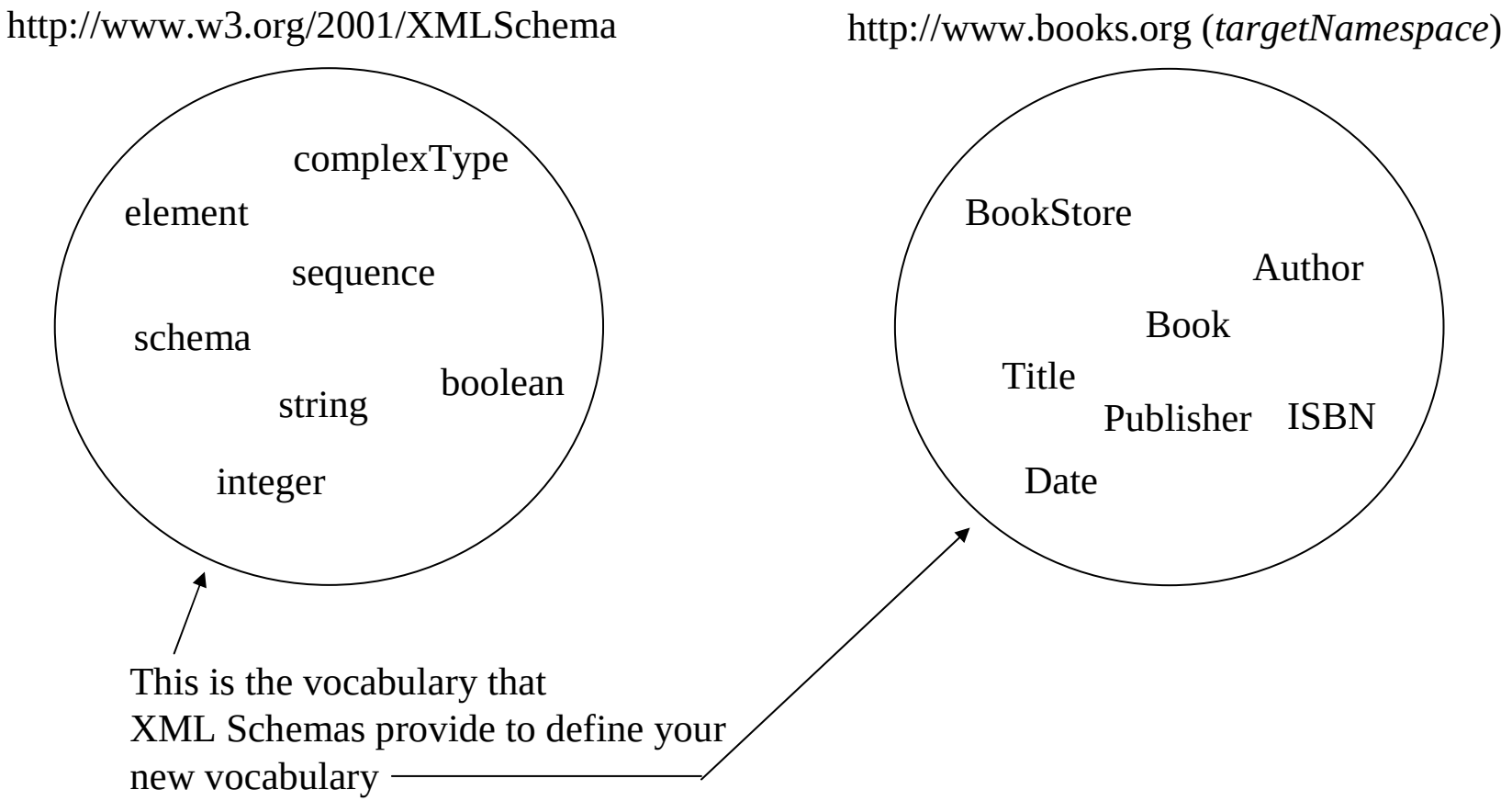
Let's Get Started!

- Convert the BookStore.dtd (next page) to the XML Schema syntax
  - for this first example we will make a straight, one-to-one conversion, i.e., Title, Author, Date, ISBN, and Publisher will hold strings, just like is done in the DTD
  - We will gradually modify the XML Schema to use stronger types

BookStore.dtd

```
<!ELEMENT BookStore (Book+)>
<!ELEMENT Book (Title, Author, Date, ISBN, Publisher)>
<!ELEMENT Title (#PCDATA)>
<!ELEMENT Author (#PCDATA)>
<!ELEMENT Date (#PCDATA)>
<!ELEMENT ISBN (#PCDATA)>
<!ELEMENT Publisher (#PCDATA)>
```





One difference between XML Schemas and DTDs is that the XML Schema vocabulary is associated with a name (namespace). Likewise, the new vocabulary that you define must be associated with a name (namespace). With DTDs neither set of vocabulary is associated with a name (namespace) [because DTDs pre-dated namespaces].

```
<?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>
```

<!ELEMENT BookStore (Book+)>

<!ELEMENT Book (Title, Author, Date, ISBN, Publisher)>

<!ELEMENT Title (#PCDATA)>

<!ELEMENT Author (#PCDATA)>

<!ELEMENT Date (#PCDATA)>

<!ELEMENT ISBN (#PCDATA)>

<!ELEMENT Publisher (#PCDATA)>

```
<?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>
```

All XML Schemas have "schema" as the root element.

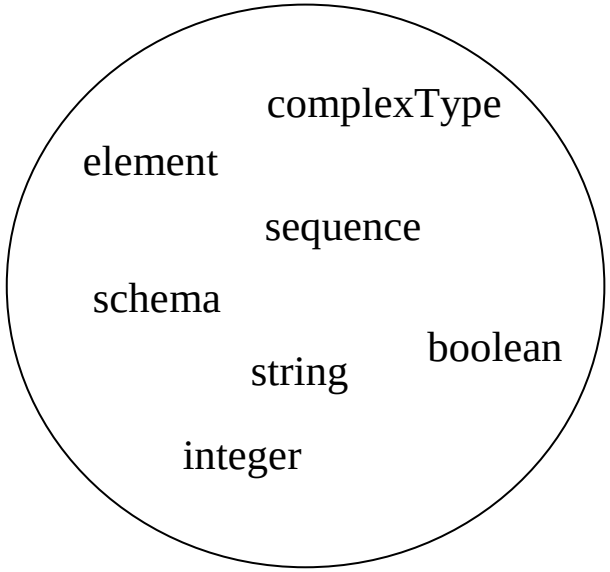
```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            targetNamespace="http://www.books.org"
            xmlns="http://www.books.org"
            elementFormDefault="qualified">
  <xsd:element name="BookStore">
    <xsd:complexType>
      <xsd:sequence>
        <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>
```

The elements and datatypes that are used to construct schemas
- schema
- element
- complexType
- sequence
- string

come from the http://…/XMLSchema namespace

XMLSchema Namespace

http://www.w3.org/2001/XMLSchema



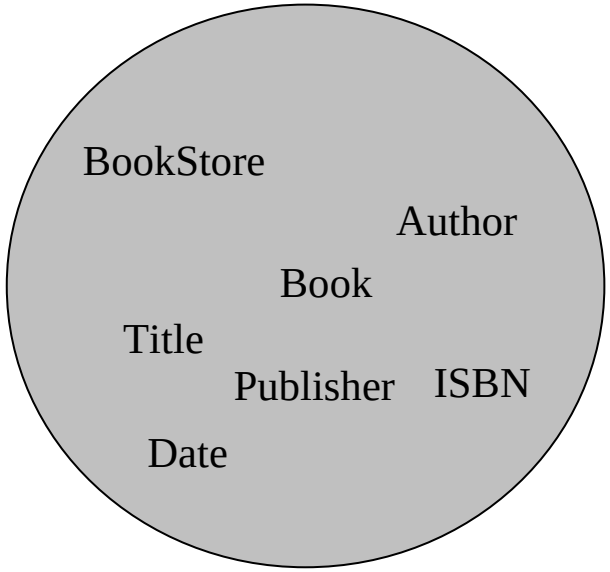
```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            targetNamespace="http://www.books.org"
            xmlns="http://www.books.org"
            elementFormDefault="qualified">
  <xsd:element name="BookStore">
    <xsd:complexType>
      <xsd:sequence>
        <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>
```

Indicates that the elements defined by this schema
- BookStore
- Book
- Title
- Author
- Date
- ISBN
- Publisher

are to go in the http://www.books.org namespace

Book Namespace (targetNamespace)

http://www.books.org (targetNamespace)



```
<?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>
```

The default namespace is http://www.books.org which is the targetNamespace!

This is referencing a Book element declaration. The Book in what namespace? Since there is no namespace qualifier it is referencing the Book element in the default namespace, which is the targetNamespace! Thus, this is a reference to the Book element declaration in this schema.

```
<?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>
```

This is a directive to any instance documents which conform to this schema: Any elements used by the instance document which were declared in this schema must be namespace qualified.

Referencing a schema in an XML instance document

```
<?xml version="1.0"?>
<BookStore xmlns ="http://www.books.org" (1)
           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" (3)
           xsi:schemaLocation="http://www.books.org
                               BookStore.xsd"> (2)
    <Book>
        <Title>My Life and Times</Title>
        <Author>Paul McCartney</Author>
        <Date>July, 1998</Date>
        <ISBN>94303-12021-43892</ISBN>
        <Publisher>McMillin Publishing</Publisher>
    </Book>
    ...
</BookStore>
```

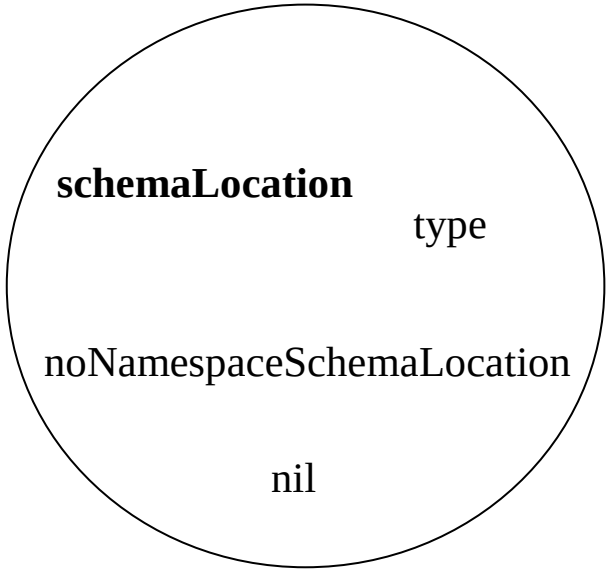
1. First, using a default namespace declaration, tell the schema-validator that all of the elements used in this instance document come from the *http://www.books.org* namespace.

2. Second, with schemaLocation tell the schema-validator that the *http://www.books.org* namespace is defined by BookStore.xsd (i.e., **schemaLocation contains a pair of values**).

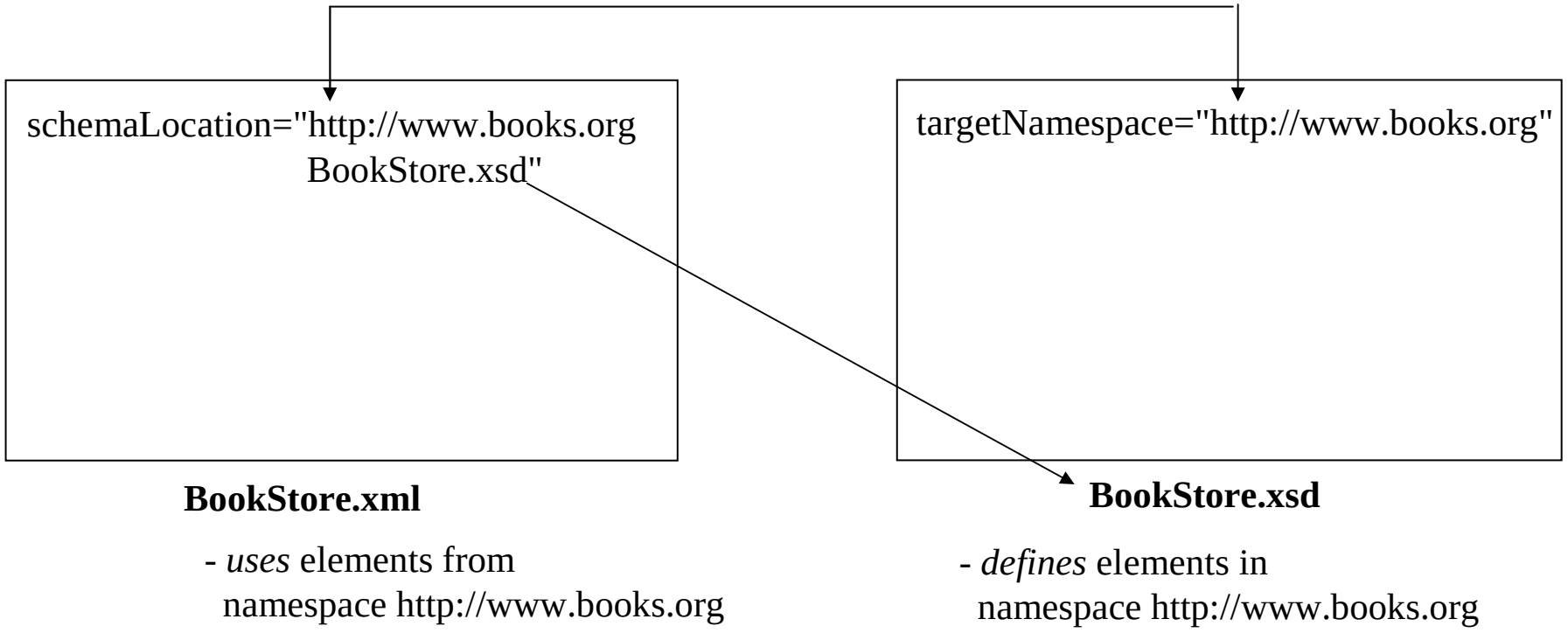
3. Third, tell the schema-validator that the schemaLocation attribute we are using is the one in the XMLSchema-instance namespace.

XMLSchema-instance Namespace

http://www.w3.org/2001/XMLSchema-instance

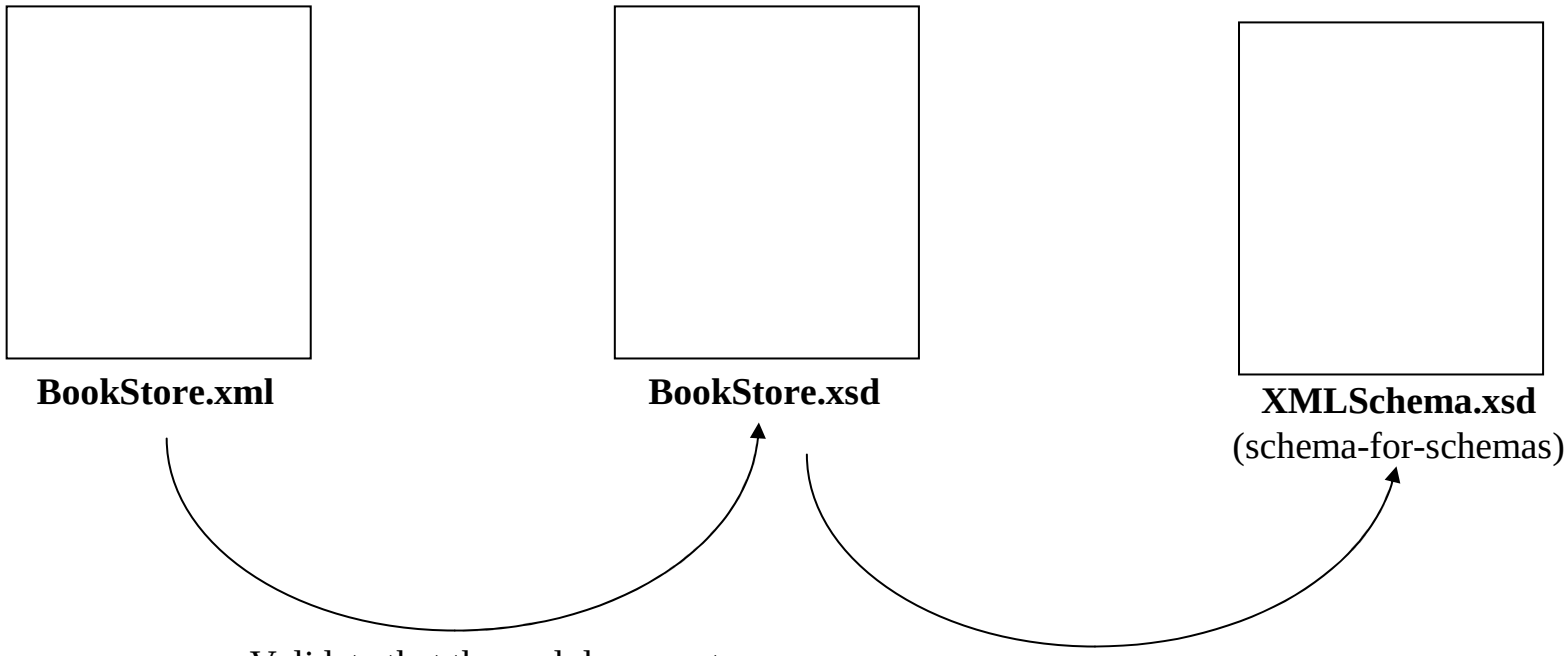


Referencing a schema in an XML instance document



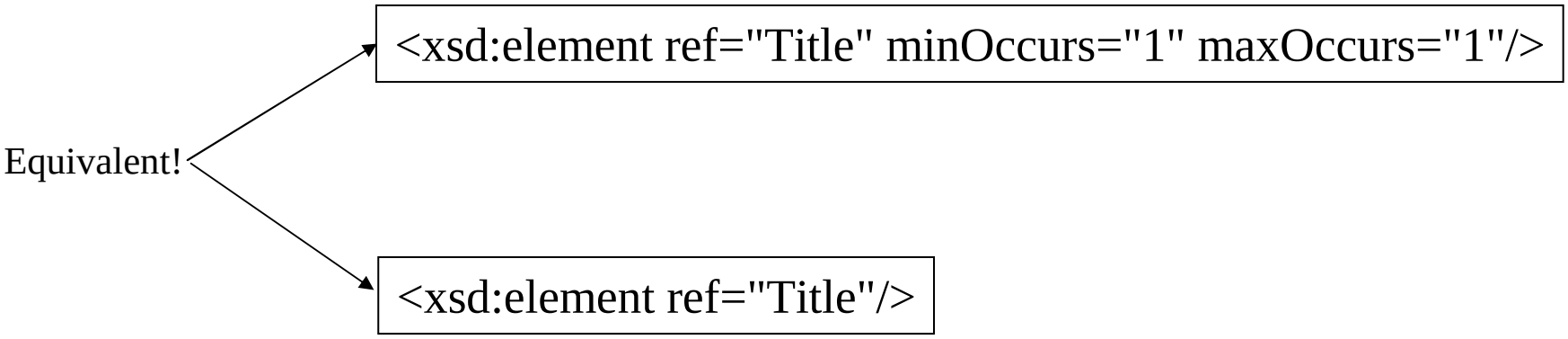
A schema *defines* a new vocabulary. Instance documents *use* that new vocabulary.

Note multiple levels of checking



Default Value for minOccurs and maxOccurs

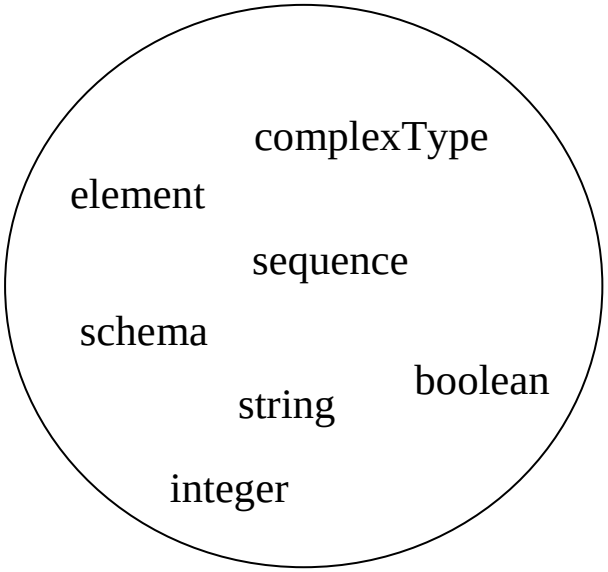
- The default value for minOccurs is "1"
- The default value for maxOccurs is "1"



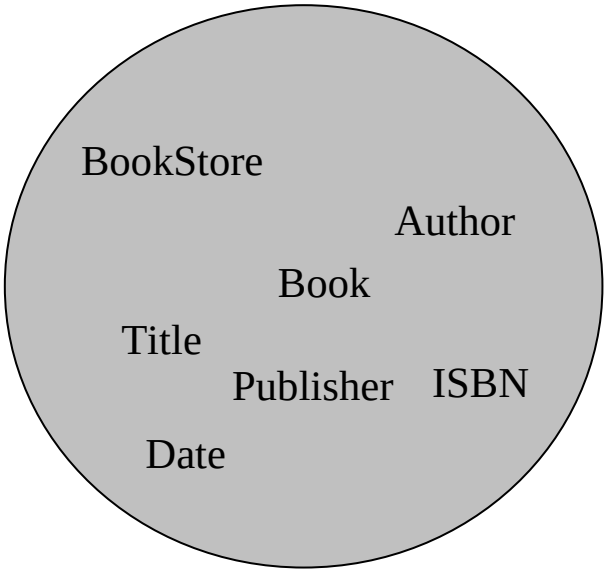
Qualify XMLSchema, Default targetNamespace

- In the first example, we explicitly qualified all elements from the XML Schema namespace. The targetNamespace was the default namespace.

http://www.w3.org/2001/XMLSchema



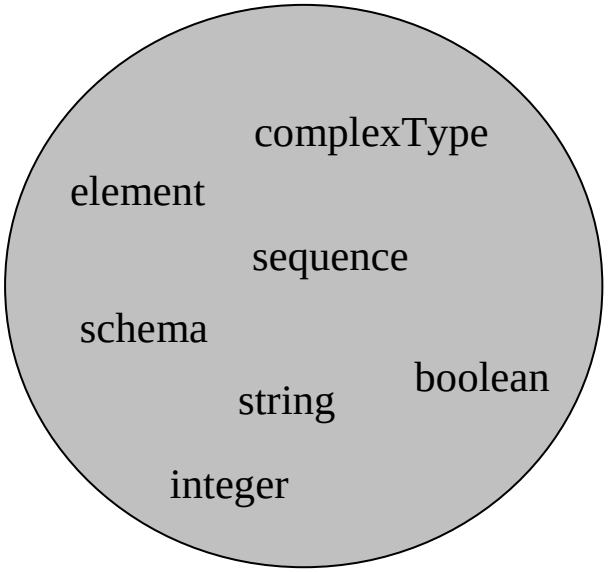
http://www.books.org (targetNamespace)



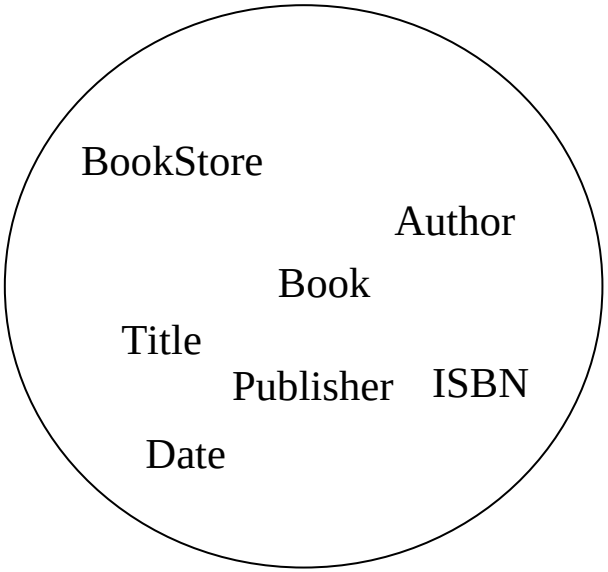
Default XMLSchema, Qualify targetNamespace

- Alternatively (equivalently), we can design our schema so that XMLSchema is the default namespace.

http://www.w3.org/2001/XMLSchema



http://www.books.org (targetNamespace)



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

Note that http://…/XMLSchema is the default namespace. Consequently, there are no namespace qualifiers on

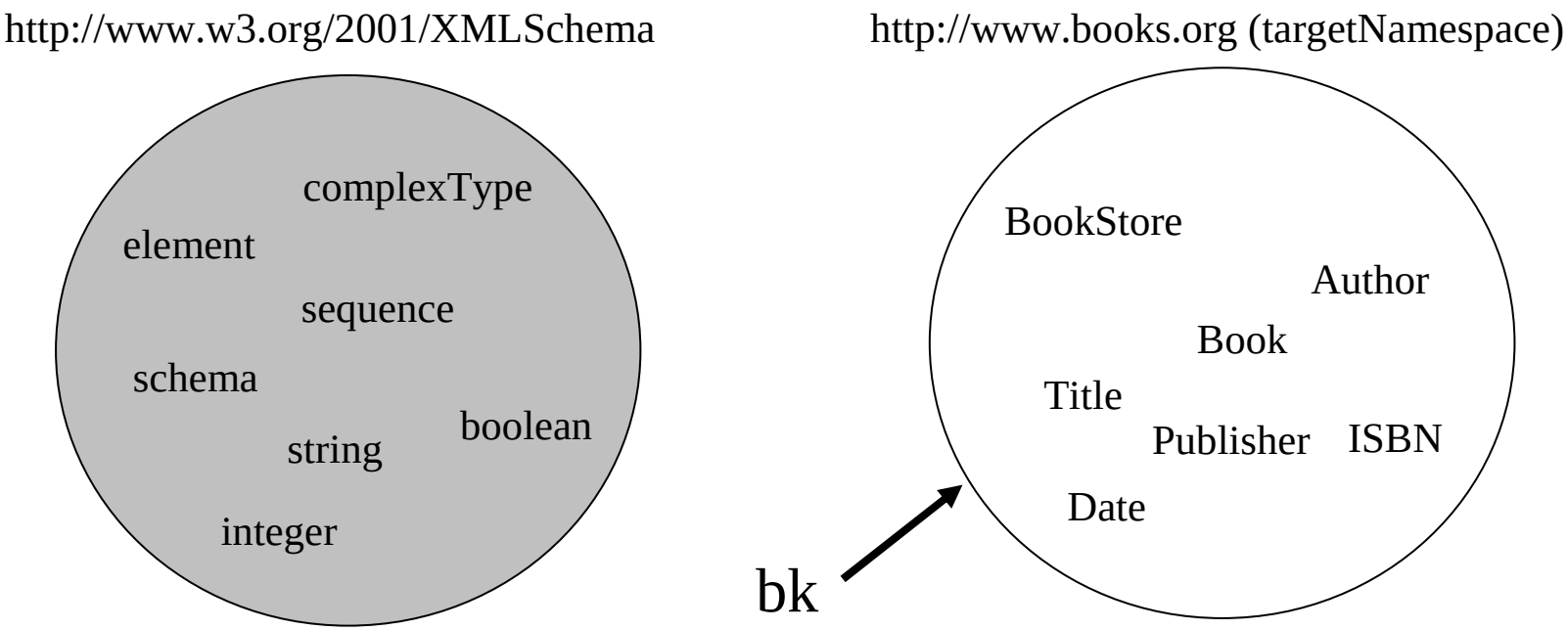
- schema
- element
- complexType
- sequence
- string

(see example02)

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

Here we are referencing a Book element. Where is that Book element defined? In what namespace? The bk: prefix indicates what namespace this element is in. bk: has been set to be the same as the targetNamespace.

"bk:" References the targetNamespace



Consequently, *bk:Book* refers to the Book element in the targetNamespace.

Inlining Element Declarations

- In the previous examples we declared an element and then we ref'ed 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*.

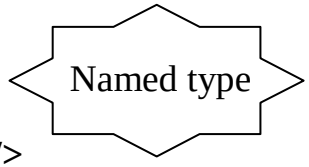
```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            targetNamespace="http://www.books.org"
            xmlns="http://www.books.org"
            elementFormDefault="qualified">
  <xsd:element name="BookStore">
    <xsd:complexType>
      <xsd:sequence>
        <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.

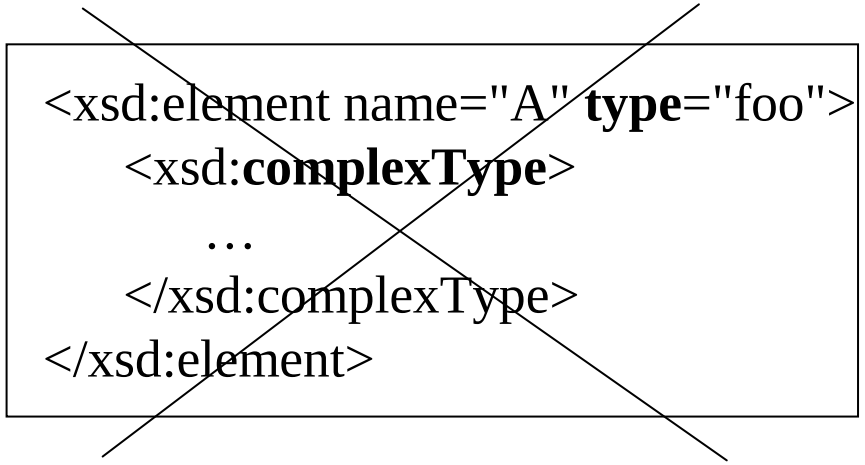
is equivalent to:

```
<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.



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>
```

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 gYear
    - 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)

(see example05)

```
<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:

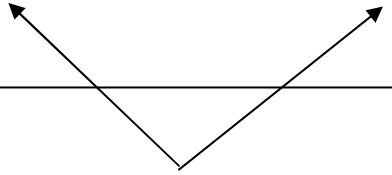
- First Pattern: 1 digit followed by a dash followed by 5 digits followed by another dash followed by 3 digits followed by another dash followed by 1 more digit, or
- Second Pattern: 1 digit followed by a dash followed by 3 digits followed by another dash followed by 5 digits followed by another dash 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**, **NAN**
  - double → **12.56E3**, **12**, **12560**, **0**, **-0**, **INF**, **-INF**, **NAN**
  - 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: 'T' is the date/time separator
INF = infinity
NAN = not-a-number

Built-in Datatypes (cont.)

- Primitive Datatypes
  - gDay
  - gMonth
  - hexBinary
  - base64Binary
  - anyURI
  - QName
  - NOTATION
- 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
  - a NOTATION from the XML spec

Built-in Datatypes (cont.)

- Derived types
  - normalizedString → Subtype of primitive datatype: A string without tabs, line feeds, or carriage returns
  - token → String w/o tabs, l/f, leading/trailing spaces, consecutive spaces
  - language → any valid xml:lang value, e.g., **EN**, **FR**, ...
  - IDREFS → must be used only with attributes
  - ENTITIES → must be used only with attributes
  - NMTOKEN → must be used only with attributes
  - NMTOKENS → must be used only with attributes
  - Name
  - NCName → **part** (no namespace qualifier)
  - ID → must be used only with attributes
  - IDREF → must be used only with attributes
  - ENTITY → must be used only with attributes
  - integer → **456**
  - nonPositiveInteger → negative infinity to 0

Built-in Datatypes (cont.)

Derived types	Subtype of primitive datatype
negativeInteger	negative infinity to -1
long	**-9223372036854775808** to **9223372036854775807**
int	**-2147483648** to **2147483647**
short	**-32768** to **32767**
byte	**-127** to **128**
nonNegativeInteger	0 to infinity
unsignedLong	**0** to **18446744073709551615**
unsignedInt	**0** to **4294967295**
unsignedShort	**0** to **65535**
unsignedByte	**0** to **255**
positiveInteger	**1** to infinity

Note: the following types can only be used with attributes:
ID, IDREF, IDREFS, NMTOKEN, NMTOKENS, ENTITY, and ENTITIES.

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

```
<xsd:simpleType name= "name">
   <xsd:restriction base= "xsd:source">
      <xsd:facet value= "value"/>
      <xsd:facet value= "value"/>
      …
   </xsd:restriction>
</xsd:simpleType>
```

Facets:
- length
- minlength
- maxlength
- pattern
- enumeration
- minInclusive
- maxInclusive
- minExclusive
- maxExclusive

...

Sources:
- string
- boolean
- number
- float
- double
- duration
- dateTime
- time

...

Multiple Facets - "*and*" them together, or "*or*" them together?

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

An element declared to be of type TelephoneNumber must be a string of length=8 *and* the string must follow the pattern: 3 digits, dash, 4 digits.

```
<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>
```

An element declared to be of type shape must be a string with a value of *either* circle, *or* triangle, *or* square.

Patterns, enumerations => "or" them together
All other facets => "and" them together

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) facet 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= "ClassSize">
   <xsd:restriction base="xsd:nonNegativeInteger">
      <xsd:minInclusive value="10" fixed="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"/>
```

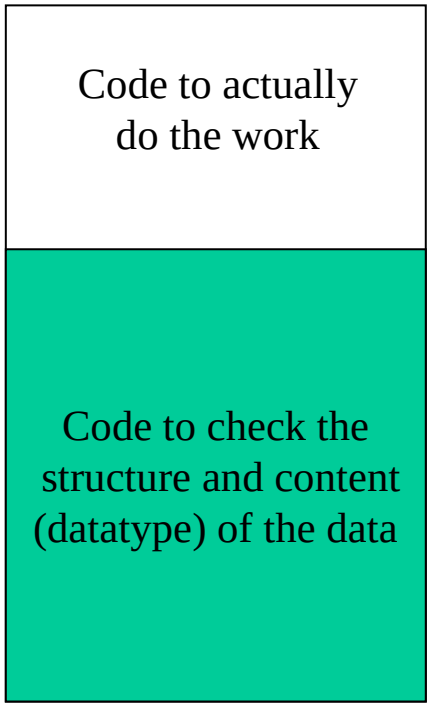
2

```
<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>
```

Save $$$ using XML Schemas

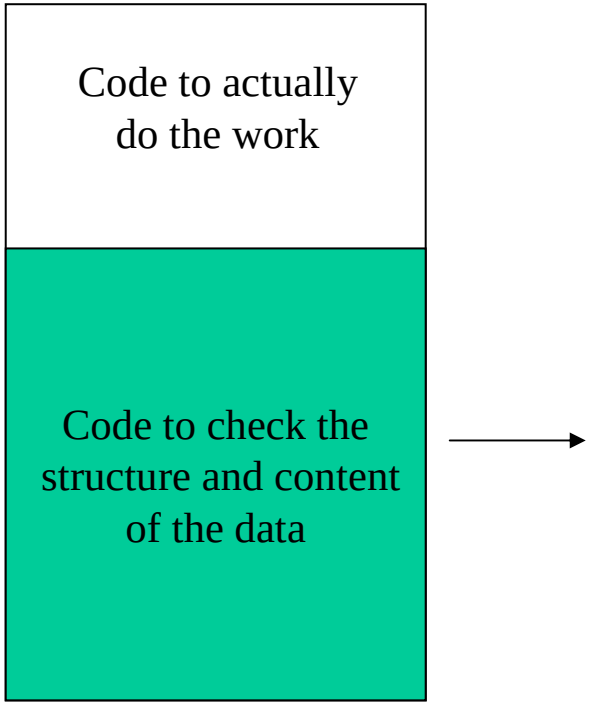


"In a typical program, up to 60% of the code is spent checking the data!"

- source unknown

Continued -->

Save $$$ using XML Schemas (cont.)

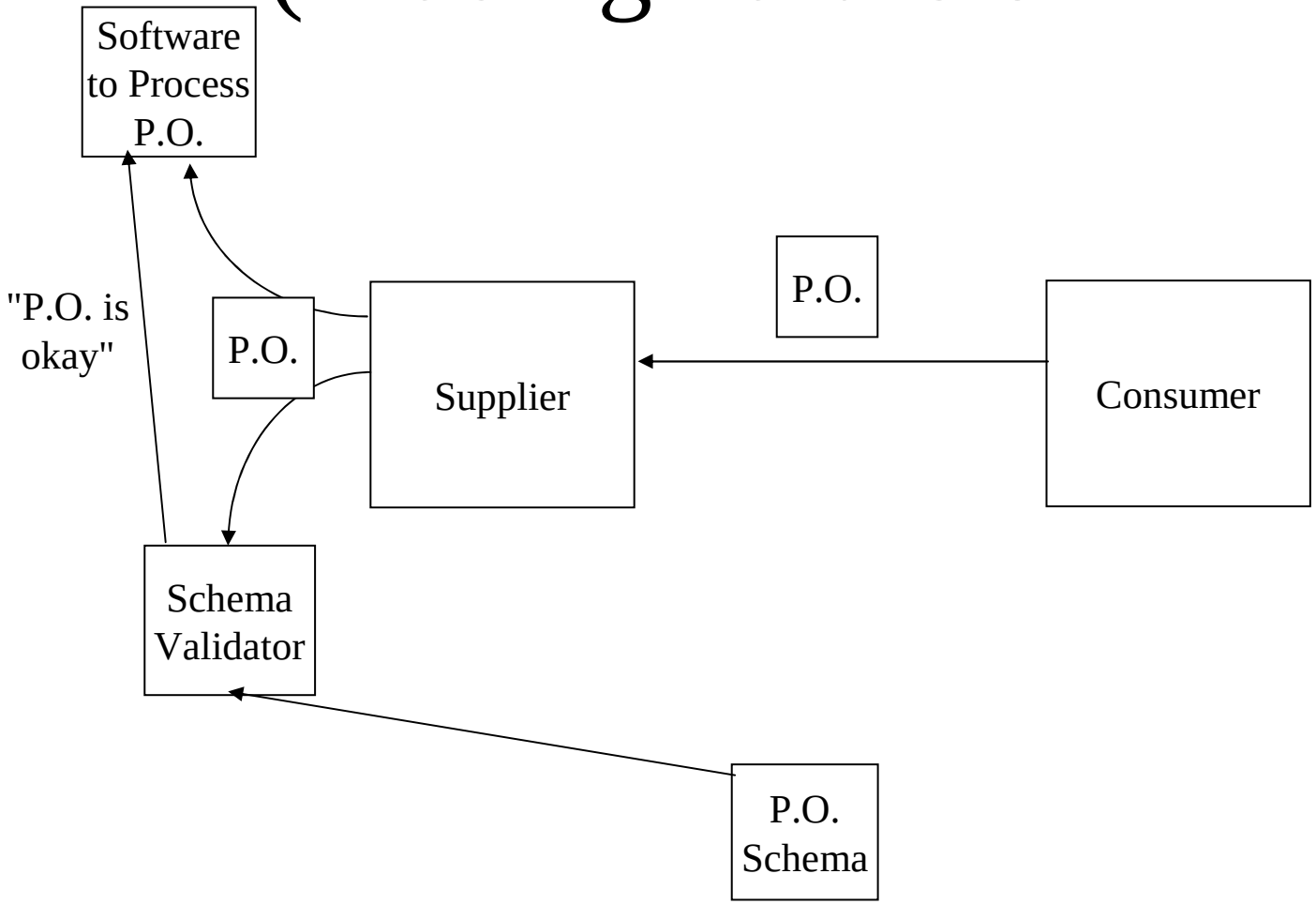


If your data is structured as XML, and there is a schema, then you can hand the data-checking task off to a schema validator.

Thus, your code is reduced by up to 60%!!!

Big $$ savings!

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 datatype 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	Example
- Chapter \d	- Chapter 1
- Chapter \d	- Chapter 1
- a*b	- b, ab, aab, aaab, …
- [xyz]b	- xb, yb, zb
- a?b	- b, ab
- a+b	- ab, aab, aaab, …

Regular Expressions (cont.)

Regular Expression	Example
[a-c]x	ax, bx, cx
[-ac]x	-x, ax, cx
[ac-]x	ax, cx, -x
[^0-9]x	any non-digit char followed by x
\Dx	any non-digit char followed by x
Chapter\s\d	Chapter followed by a blank followed by a digit
(ho){2} there	hoho there
(ho\s){2} there	ho ho there
.abc	any (*one*) char followed by abc
(a|b)+x	ax, bx, aax, bbx, abx, bax,...

Regular Expressions (cont.)

• a{1,3}x	• ax, aax, aaax
• a{2,}x	• aax, aaax, aaaax, …
• \w\s\w	• word **character** (alphanumeric plus dash) followed by a space followed by a word character
• [a-zA-Z-[Ol]]*	• A string comprised of any lower and upper case letters, except "O" and "l"
• \.	• The period "." (Without the backward slash the period means "any character")

Regular Expressions (cont.)

• \n	• linefeed
• \r	• carriage return
• \t	• tab
• \\	• The backward slash \
• |	• The vertical bar |
• \-	• The hyphen -
• \^	• The caret ^
• \?	• The question mark ?
• *	• The asterisk *
• \+	• The plus sign +
• \{	• The open curly brace {
• \}	• The close curly brace }
• \(	• The open paren (
• \)	• The close paren)
• \[	• The open square bracket [
• \]	• The close square bracket]

Regular Expressions (concluded)

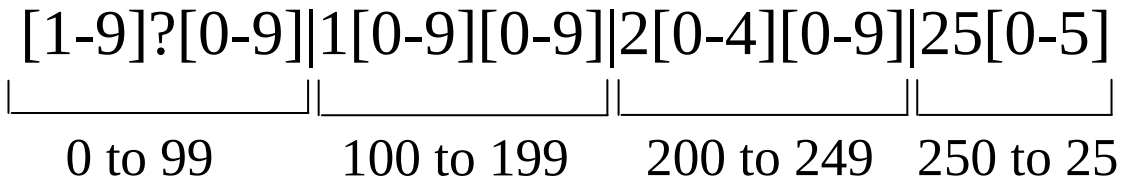
• \p{L}	• A letter, from any language
• \p{Lu}	• An uppercase letter, from any language
• \p{Ll}	• A lowercase letter, from any language
• \p{N}	• A number - Roman, fractions, etc
• \p{Nd}	• A digit from any language
• \p{P}	• A punctuation symbol
• \p{Sc}	• A currency sign, from any language

```
<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"/>
```

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

"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"

Example R.E.



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

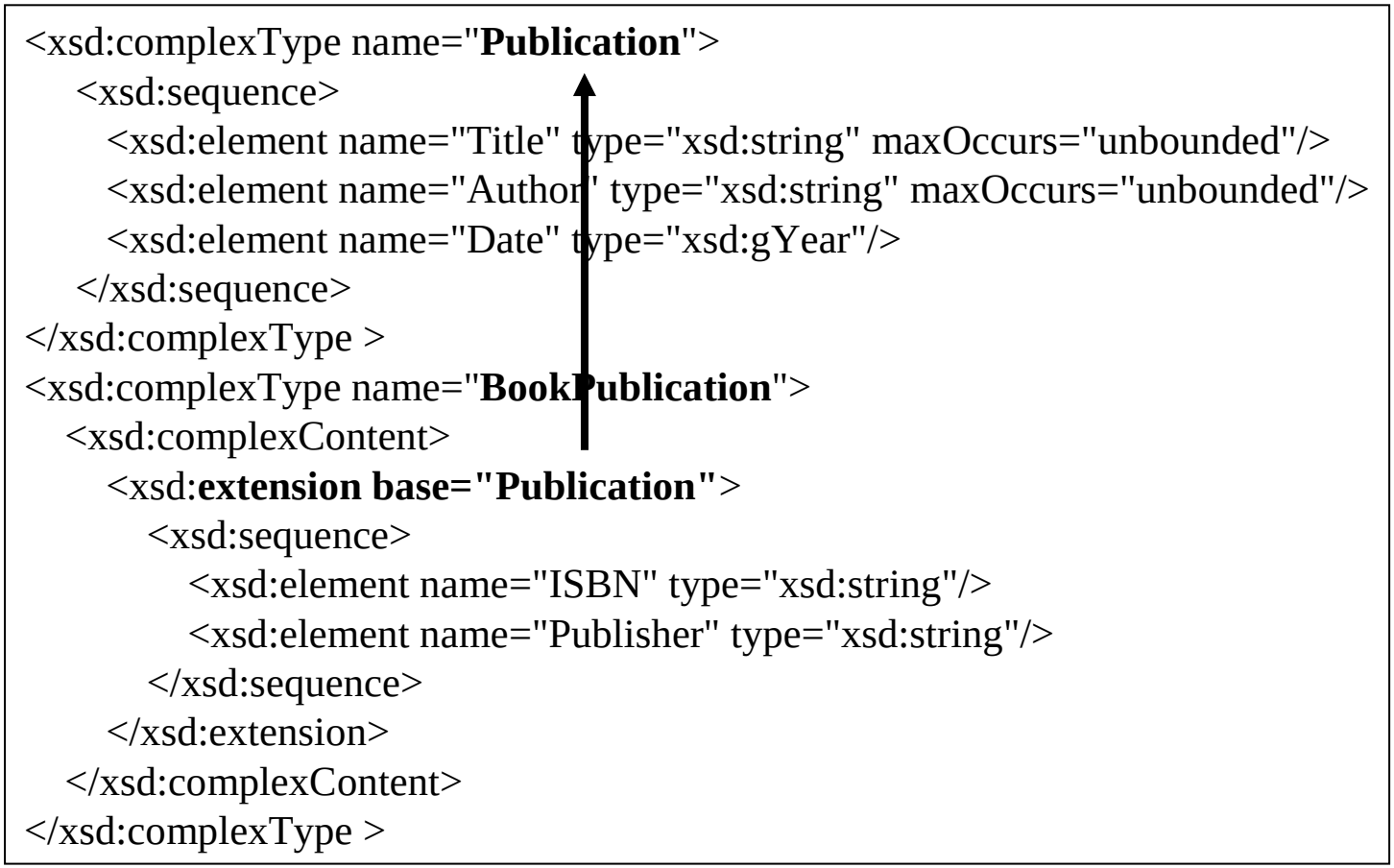
```
<xsd:simpleType name="IP">
   <xsd:restriction base="xsd:string">
      <xsd:pattern value="(([1-9]?[0-9]|1[0-9][0-9]|2[0-4][0-9]|25[0-5])\.){3}
                          ([1-9]?[0-9]|1[0-9][0-9]|2[0-4][0-9]|25[0-5])">
         <xsd:annotation>
            <xsd:documentation>
               Datatype for representing IP addresses.  Examples,
                 129.83.64.255, 64.128.2.71, etc.
               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]
               Note: in the value attribute (above) the regular
               expression has been split over two lines.  This is
               for readability purposes only.  In practice the R.E.
               would all be on one line.
            </xsd:documentation>
         </xsd:annotation>
      </xsd:pattern>
   </xsd:restriction>
</xsd:simpleType>
```

Derived Types

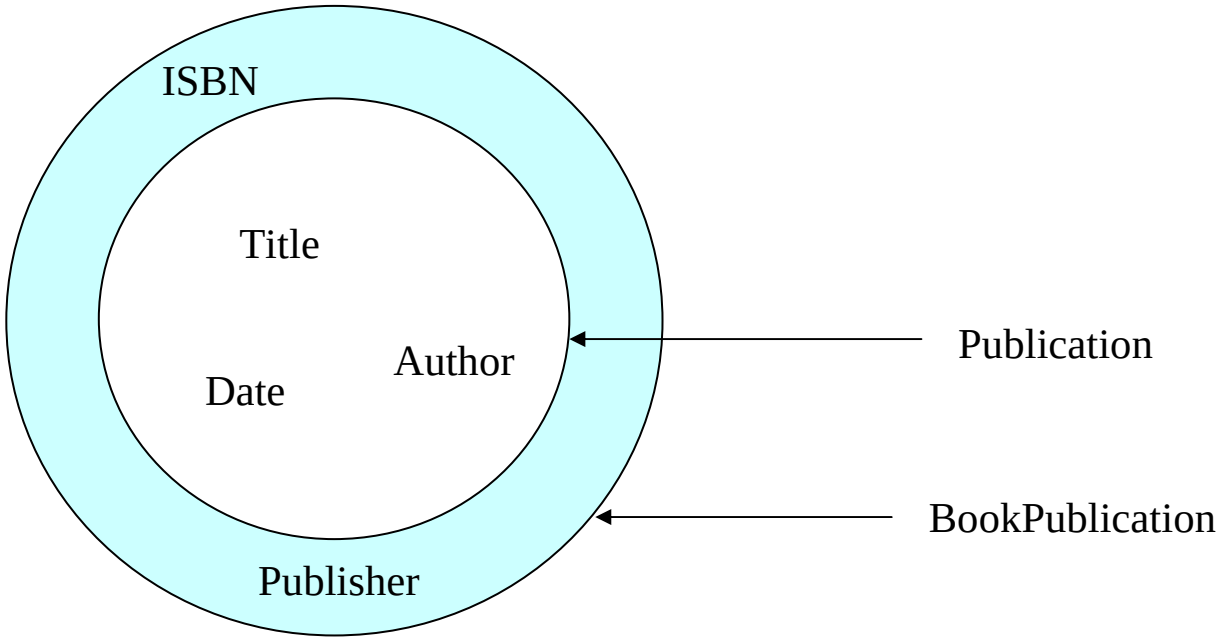
- We can do a form of subclassing complexType definitions. We call this "derived types"
  - derive by extension: extend the parent complexType with more elements
  - derive by restriction: create a type which is a subset of the base type. There are two ways to subset the elements:
    - redefine a base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

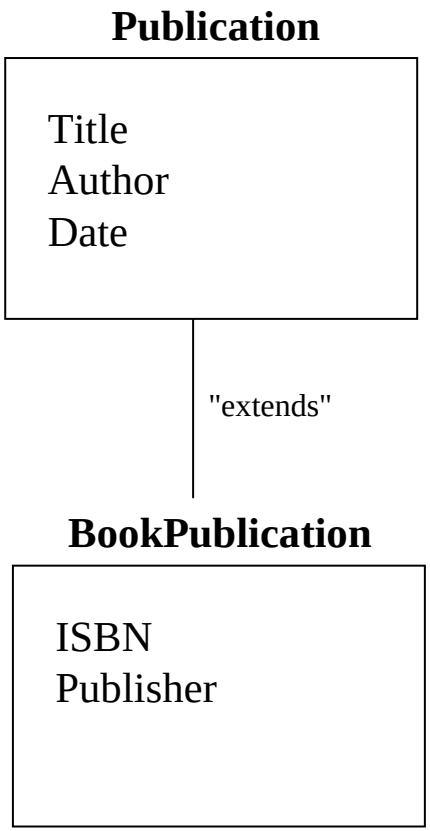
```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
                targetNamespace="http://www.books.org"
                xmlns="http://www.books.org"
               elementFormDefault="qualified">
  <xsd:complexType name="Publication">
    <xsd:sequence>
      <xsd:element 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



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>
```

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.

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.

Derive by Restriction (cont.)

- You might (legitimately) ask:
  - why do I have to repeat all the declarations from the base type? Why can't I simply show the delta (i.e., show those declarations that are changed)?
  - What's the advantage of doing derived by restriction if I have to repeat everything? I'm certainly not saving on typing.
- Answer:
  - Even though you have to retype everything in the base type by associating a type with a base type you can do *element substitution* - the ability to substitute one element for another. A restriction of element substitution is that the substituting element have a type that derives from the type of the element it is substituting. Thus, it is beneficial to link the type.
  - Also, an element's content model may be substituted by the content model of derived types. Thus, the content of an element that has been declared to be of type Publication can be substituted with a SingleAuthorPublication content since SingleAuthorPublication derives from Publication.

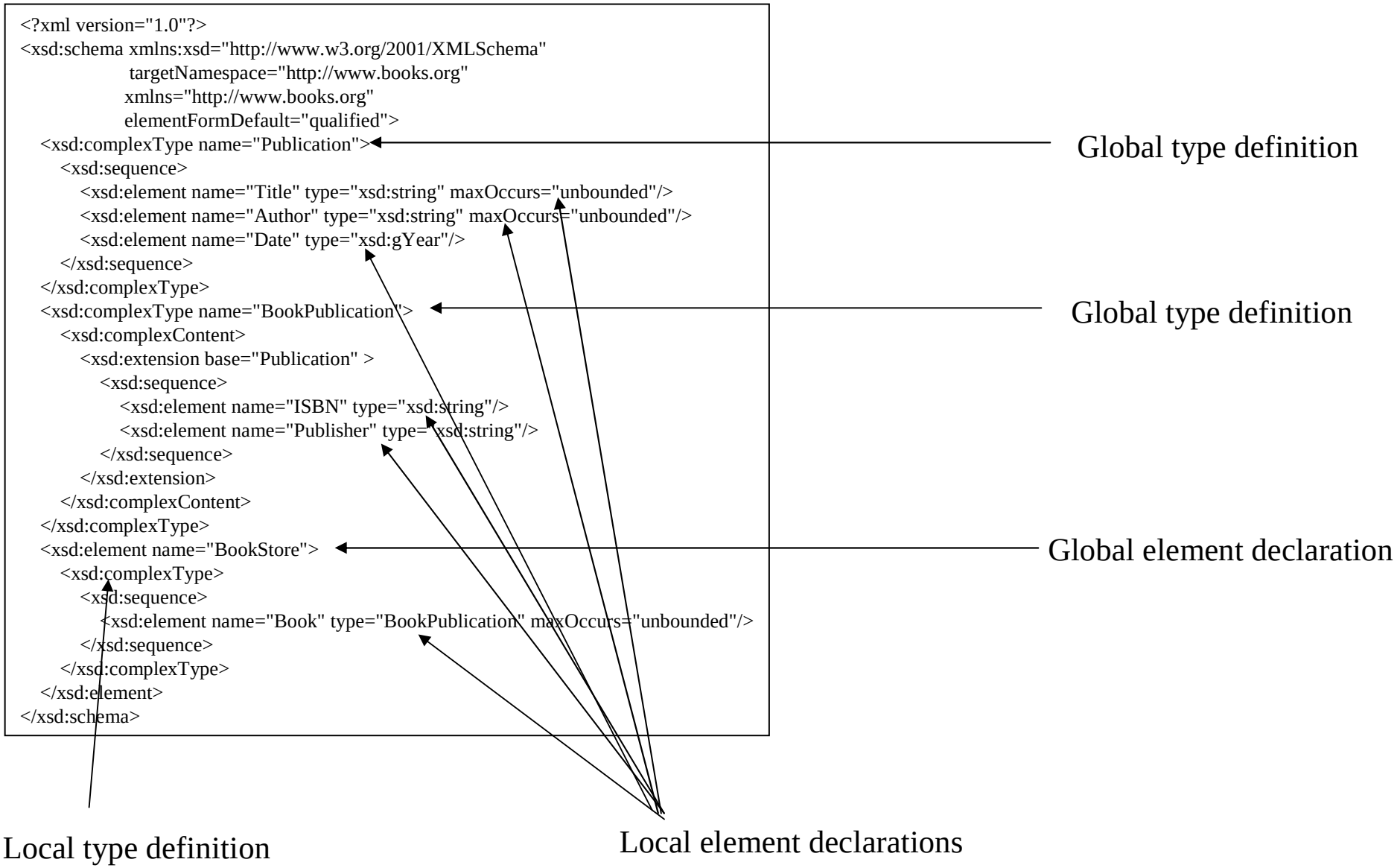
Terminology: Declaration vs Definition

- In a schema:
  - You *declare* elements and attributes. Schema components that are *declared* are those that have a representation in an XML instance document.
  - You *define* components that are used just within the schema document(s). Schema components that are *defined* are those that have no representation in an XML instance document.

Declarations:	**Definitions**:
- element declarations - attribute declarations	- type (simple, complex) definitions - attribute group definitions - model group definitions

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.



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

- On the next slide I show a version of the BookStore DTD that uses attributes. Then, on the following slide I show 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>
```

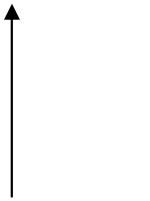
"Instance documents are required to have the Category attribute (as indicated by use="required"). The value of Category must be either autobiography, non-fiction, or fiction (as specified by the enumeration facets)."

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

Summary of Declaring Attributes (two ways to do it)

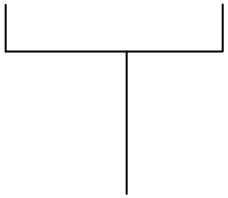
1

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



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

required
optional
prohibited



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.

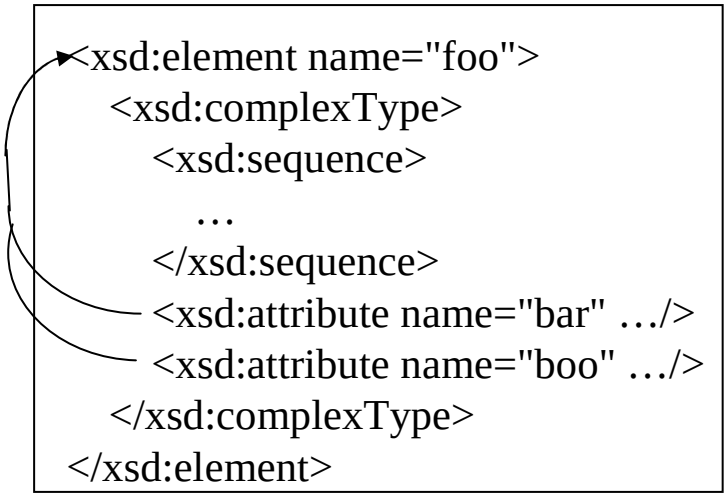
```
<xsd:element name="Book"  maxOccurs="unbounded">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element name="Title" type="xsd:string"/>
            <xsd:element name="Author" type="xsd:string" maxOccurs="unbounded"/>
            <xsd:element name="Date" type="xsd:string"/>
            <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>
```

(see example08)

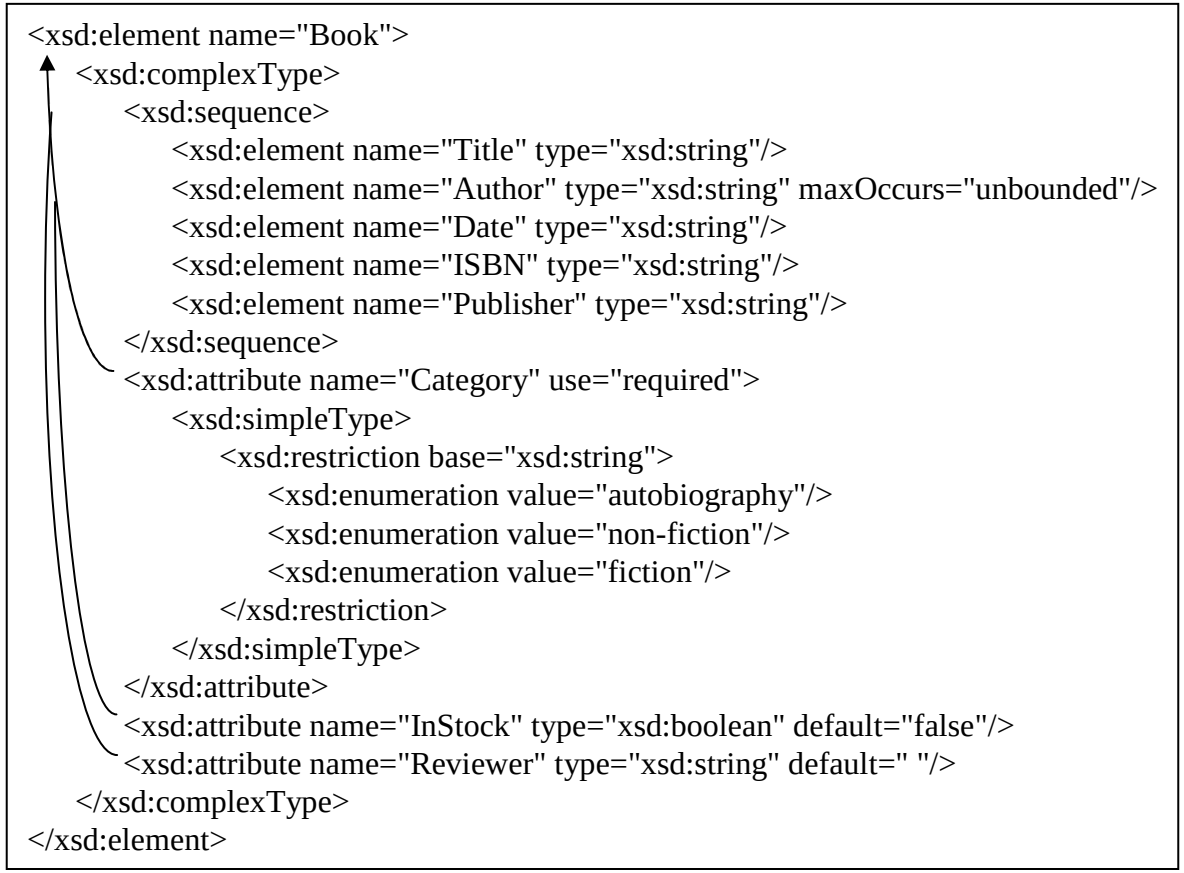
Notes about Attributes

- 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"



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>
```

Element with Simple Content and Attributes

Example. Consider this:

```
<elevation units="feet">5440</elevation>
```

The elevation element has these two constraints:

- it has a simple (integer) content
- it has an attribute called units

How do we declare elevation? (see next slide)

```
<xsd:element name="elevation">
   <xsd:complexType> (1)
      <xsd:simpleContent> (2)
         <xsd:extension base="xsd:integer"> (3)
            <xsd:attribute name="units" type="xsd:string" use="required"/> (4)
         </xsd:extension>
      </xsd:simpleContent>
   </xsd:complexType>
</xsd:element>
```

1. elevation contains an attribute.
   - therefore, we must use <xsd:complexType>
2. However, elevation does not contain child elements (which is what we generally use <complexType> to indicate). Instead, elevation contains simpleContent.
3. We wish to extend the simpleContent (an integer) ...
4. with an attribute.

elevation - use Stronger Datatype

- In the declaration for elevation we allowed it to hold any integer. Further, we allowed the units attribute to hold any string.
- Let's restrict elevation to hold an integer with a range 0 - 12,000 and let's restrict units to hold either the string "feet" or the string "meters"

```
<xsd:simpleType name="elevationType">
   <xsd:restriction base="xsd:integer">
      <xsd:minInclusive value="0"/>
      <xsd:maxInclusive value="12000"/>
   </xsd:restriction>
</xsd:simpleType>
<xsd:simpleType name="unitsType">
   <xsd:restriction base="xsd:string">
      <xsd:enumeration value="feet"/>
      <xsd:enumeration value="meters"/>
   </xsd:restriction>
</xsd:simpleType>
<xsd:element name="elevation">
   <xsd:complexType>
      <xsd:simpleContent>
         <xsd:extension base="elevationType">
            <xsd:attribute name="units" type="unitsType" use="required"/>
         </xsd:extension>
      </xsd:simpleContent>
   </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"/>
```

Summary of Declaring Elements (concluded)

5. **Element Contains Simple Content and Attributes**

```
<xsd:element name="apple">
  <xsd:complexType>
    <xsd:simpleContent>
      <xsd:extension base="xsd:string">
        <xsd:attribute name="variety" type="xsd:string" use="required"/>
      </xsd:extension>
    </xsd:simpleContent>
  </xsd:complexType>
</xsd:element>
```

Example. <apple variety="Cortland">Large, green, sour</apple>

complexContent versus simpleContent

- With complexContent you extend or restrict a complexType
- With simpleContent you extend or restrict a simpleType

```
<xsd:complexType name="…">
   <xsd:complexContent>
      <xsd:extension base="X">
         …
      </xsd:extension>
   </xsd:complexContent>
</xsd:complexType>
```

versus

```
<xsd:complexType name="…">
   <xsd:simpleContent>
      <xsd:extension base="Y">
         …
      </xsd:extension>
   </xsd:simpleContent>
</xsd:complexType>
```

X must be a complexType

Y must be a simpleType

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>
```

Expressing Any Order

Problem: create an element, Book, which contains Author, Title, Date, ISBN, and Publisher, *in any order* (Note: this is very difficult and ugly with DTDs).

XML Schema:

```
<?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:all>
              <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:all>
          </xsd:complexType>
        </xsd:element>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
```

<all> means that Book must contain all five child elements, but they may occur in any order.

Constraints on using <all>

- Elements declared within <all> must have a maxOccurs value of "1" (minOccurs can be either "0" or "1")
- If a complexType uses <all> and it extends another type, then that parent type must have empty content.
- The <all> element cannot be nested within either <sequence>, <choice>, or another <all>
- The contents of <all> must be just elements. It cannot contain <sequence> or <choice>

Empty Element

DTD:

```
<!ELEMENT image EMPTY>
<!ATTLIST image href CDATA #REQUIRED>
```

Schema:

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            targetNamespace="http://www.photography.org"
            xmlns="http://www.photography.org"
            elementFormDefault="qualified">
  <xsd:element name="gallery">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="image" maxOccurs="unbounded">
          <xsd:complexType>
            <xsd:attribute name="href" type="xsd:anyURI" use="required"/>
          </xsd:complexType>
        </xsd:element>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

Instance doc (snippet):

```
<image href="http://www.xfront.com/InSubway.gif"/>
```

Assembling an Instance Document from Multiple Schema Documents

- An instance document may be composed of elements from multiple schemas.
- Validation can apply to the entire XML instance document, or to a single element.

```
<?xml version="1.0"?>
<Library xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation=
              "http://www.book.org
               Book.xsd
               http://www.employee.org
               Employee.xsd">
  <Books>
    <Book xmlns="http://www.book.org">
      <Title>My Life and Times</Title>
      <Author>Paul McCartney</Author>
      <Date>1998</Date>
      <ISBN>1-56592-235-2</ISBN>
      <Publisher>Macmillan Publishing</Publisher>
    </Book>
    <Book xmlns="http://www.book.org">
      <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>
    <Book xmlns="http://www.book.org">
      <Title>The First and Last Freedom</Title>
      <Author>J. Krishnamurti</Author>
      <Date>1954</Date>
      <ISBN>0-06-064831-7</ISBN>
      <Publisher>Harper &amp;amp; Row</Publisher>
    </Book>
  </Books>
  <Employees>
    <Employee xmlns="http://www.employee.org">
      <Name>John Doe</Name>
      <SSN>123-45-6789</SSN>
    </Employee>
    <Employee xmlns="http://www.employee.org">
      <Name>Sally Smith</Name>
      <SSN>000-11-2345</SSN>
    </Employee>
  </Employees>
</Library>
```

Validating against two schemas

The <Book> elements are defined in Book.xsd, and the <Employee> elements are defined in Employee.xsd. The <Library>, <Books>, and <Employees> elements are not defined in any schema!

1. A schema validator will validate each Book element against Book.xsd.
2. It will validate each Employee element against Employee.xsd.
3. It will not validate the other elements.

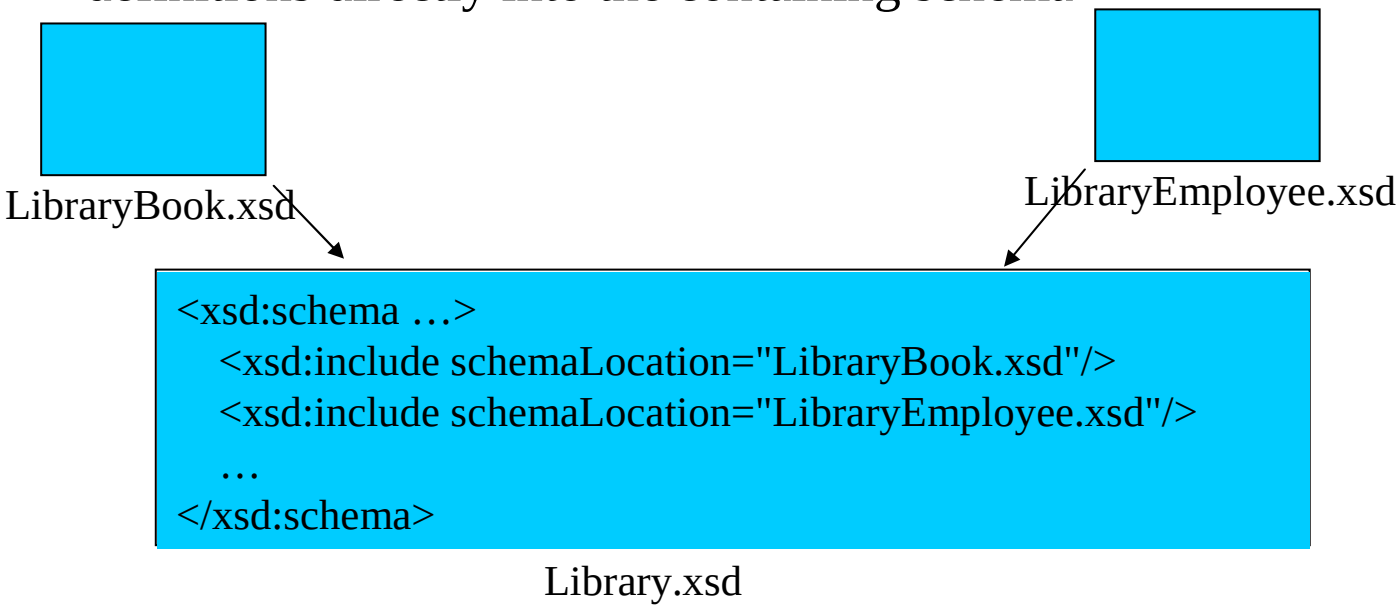
Library.xml

Lax Validation vs Strict Validation

- On the previous slide there were elements (Library, Books, and Employees) for which there was no schema to validate against.
- Lax validation is where the schema validator skips over elements for which no schema is available.
- Strict validation is where the schema validator requires validation of every element
- xsv performs lax validation. Thus, it will accept the instance document on the previous slide (but it will note validation="lax" in its output)
- All the other validators do strict validation. Consequently, they will reject the instance document on the previous slide.

Assembling a Schema from Multiple Schema Documents

- The include element allows you to access components in other schemas
  - All the schemas you include **must** have the **same namespace** as your schema (i.e., the schema that is doing the include)
  - The net effect of include is as though you had typed all the definitions directly into the containing schema



```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            targetNamespace="http://www.library.org"
            xmlns="http://www.library.org"
            elementFormDefault="qualified">
  <xsd:include schemaLocation="LibraryBook.xsd"/>
  <xsd:include schemaLocation="LibraryEmployee.xsd"/>
  <xsd:element name="Library">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="Books">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element ref="Book" maxOccurs="unbounded"/>
            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
        <xsd:element name="Employees">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element ref="Employee" maxOccurs="unbounded"/>
            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

These are referencing element declarations in the other schemas. Nice!

Library.xsd

Note about using Include

- The <include> elements must come before any element declarations or type definitions.

Creating Lists

- There are times when you will want an element to contain a list of values, e.g., "The contents of the Numbers element is a list of numbers".

Example: For a document containing a Lottery drawing we might have

<Numbers>12 49 37 99 20 67</Numbers>

How do we declare the element Numbers ...

(1) To contain a list of integers, and
(2) Each integer is restricted to be between 1 and 99, and
(3) The total number of integers in the list is exactly six.

```
<?xml version="1.0"?>
<LotteryDrawings xmlns="http://www.lottery.org"
                 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                 xsi:schemaLocation=
                         "http://www.lottery.org
                          Lottery.xsd">
    <Drawing>
        <Week>July 1</Week>
        <Numbers>21 3 67 8 90 12</Numbers>
    </Drawing>
    <Drawing>
        <Week>July 8</Week>
        <Numbers>55 31 4 57 98 22</Numbers>
    </Drawing>
    <Drawing>
        <Week>July 15</Week>
        <Numbers>70 77 19 35 44 11</Numbers>
    </Drawing>
</LotteryDrawings>
```

Lottery.xml (see example19)

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            targetNamespace="http://www.lottery.org"
            xmlns="http://www.lottery.org"
            elementFormDefault="qualified">
  <xsd:simpleType name="LotteryNumbers">
    <xsd:list itemType="xsd:positiveInteger"/>
  </xsd:simpleType>
  <xsd:element name="LotteryDrawings">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="Drawing" minOccurs="0" maxOccurs="unbounded">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name="Week" type="xsd:string"/>
              <xsd:element name="Numbers" type="LotteryNumbers"/>
            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

Lottery.xsd

LotteryNumbers --> Need Stronger Datatyping

- The list in the previous schema has two problems:
  - It allows <Numbers> to contain an arbitrarily long list
  - The numbers in the list may be any positiveInteger
- We need to:
  - Restrict the list to length value="6"
  - Restrict the numbers to maxInclusive value="99"

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
                targetNamespace="http://www.lottery.org"
                xmlns="http://www.lottery.org"
               elementFormDefault="qualified">
  <xsd:simpleType name="OneToNinetyNine">
    <xsd:restriction base="xsd:positiveInteger">
      <xsd:maxInclusive value="99"/>
    </xsd:restriction>
  </xsd:simpleType>
  <xsd:simpleType name="NumbersList">
    <xsd:list itemType="OneToNinetyNine"/>
  </xsd:simpleType>
  <xsd:simpleType name="LotteryNumbers">
    <xsd:restriction base="NumbersList">
      <xsd:length value="6"/>
    </xsd:restriction>
  </xsd:simpleType>
  <xsd:element name="LotteryDrawings">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="Drawing" minOccurs="0" maxOccurs="unbounded">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name="Week" type="xsd:string"/>
              <xsd:element name="Numbers" type="LotteryNumbers"/>
            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

Lottery.xsd (see example19)

```
<xsd:simpleType name="OneToNinetyNine">
   <xsd:restriction base="xsd:positiveInteger">
      <xsd:maxInclusive value="99"/>
   </xsd:restriction>
</xsd:simpleType>
```

```
<xsd:simpleType name="NumbersList">
   <xsd:list itemType="OneToNinetyNine"/>
</xsd:simpleType>
```

```
<xsd:simpleType name="LotteryNumbers">
   <xsd:restriction base="NumbersList">
      <xsd:length value="6"/>
   </xsd:restriction>
</xsd:simpleType>
```

NumbersList is a list where the type of each item is OneToNinetyNine. LotteryNumbers restricts NumbersList to a length of six (i.e., an element declared to be of type LotteryNumbers must hold a list of numbers, between 1 and 99, and the length of the list must be exactly six).

```
<xsd:simpleType name="OneToNinetyNine">
   <xsd:restriction base="xsd:positiveInteger">
      <xsd:maxInclusive value="99"/>
   </xsd:restriction>
</xsd:simpleType>
<xsd:simpleType name="NumbersList">
   <xsd:list itemType="OneToNinetyNine"/>
</xsd:simpleType>
<xsd:simpleType name="LotteryNumbers">
   <xsd:restriction base="NumbersList">
      <xsd:length value="6"/>
   </xsd:restriction>
</xsd:simpleType>
```

Alternatively,

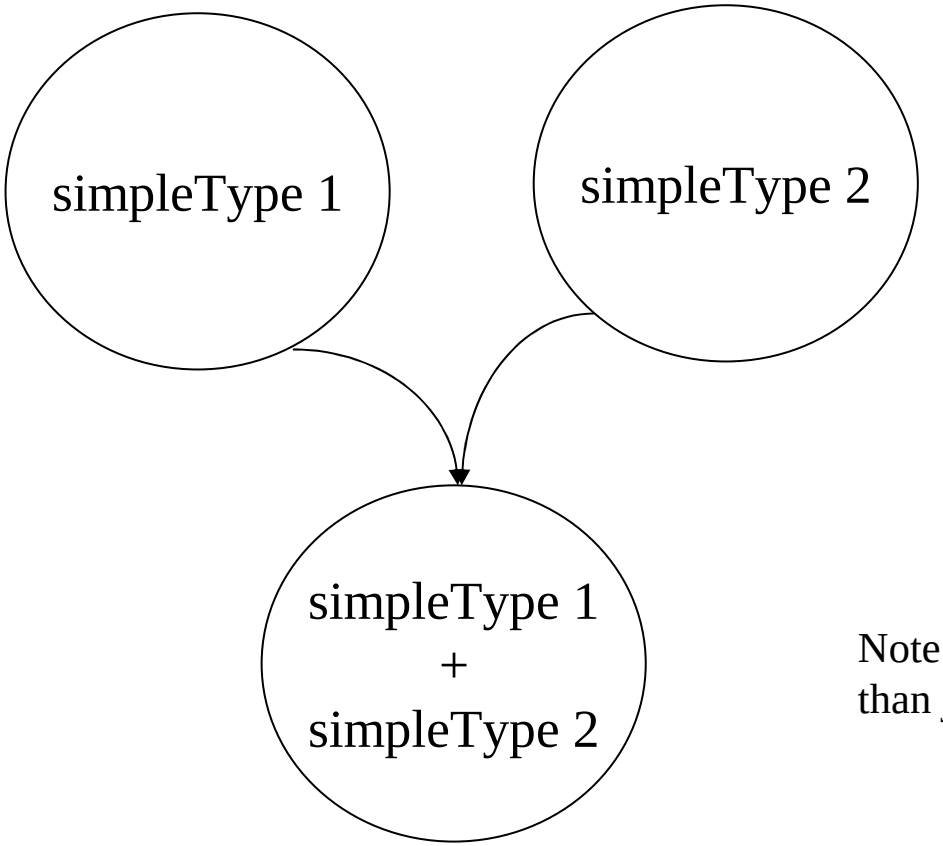
```
<xsd:simpleType name="LotteryNumbers">
    <xsd:restriction>
       <xsd:simpleType>
          <xsd:list itemType="OneToNinetyNine"/>
       </xsd:simpleType>
       <xsd:length value="6"/>
    </xsd:restriction>
</xsd:simpleType>
```

This is read as: "We are creating a new type called LotteryNumbers. It is a restriction. At this point we can either use the base attribute or a simpleType child element to indicate the type that we are restricting (you **cannot** use **both** the base attribute and the simpleType child element). We want to restrict the type that is a list of OneToNinetyNine. We will restrict that type to a length of 6."

Notes about the list type

- You **cannot create a list of lists**
  - i.e., you cannot create a list type from another list type.
- You cannot create a list of complexTypes
  - i.e., **lists only apply to simpleTypes**
- In the instance document, you must **separate each item in a list with white space** (blank space, tab, or carriage return)
- The only facets that you may use with a list type are:
  - **length**: use this to specify the **length of the list**
  - **minLength**: use this to specify the **minimum length of the list**
  - **maxLength**: use this to specify the **maximum length of the list**
  - **enumeration**: use this to specify the **values that the list may have**
  - **pattern**: use this to specify **the values that the list may have**

Creating a simpleType that is a Union of Types



Note: you can create a union of more than just two simpleTypes

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            targetNamespace="http://www.CostelloReunion.org"
            xmlns="http://www.CostelloReunion.org"
            elementFormDefault="qualified">
  <xsd:simpleType name="Parent">
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="Mary"/>
    </xsd:restriction>
  </xsd:simpleType>
  <xsd:simpleType name="PatsFamily">
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="Pat"/>
      <xsd:enumeration value="Patti"/>
      <xsd:enumeration value="Christopher"/>
      <xsd:enumeration value="Elizabeth"/>
    </xsd:restriction>
  </xsd:simpleType>
  <xsd:simpleType name="BarbsFamily">
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="Barb"/>
      <xsd:enumeration value="Greg"/>
      <xsd:enumeration value="Dan"/>
      <xsd:enumeration value="Kimberly"/>
    </xsd:restriction>
  </xsd:simpleType>
```

Cont. -->

```
<xsd:simpleType name="JudysFamily">
   <xsd:restriction base="xsd:string">
      <xsd:enumeration value="Judy"/>
      <xsd:enumeration value="Peter"/>
   </xsd:restriction>
</xsd:simpleType>
<xsd:simpleType name="TomsFamily">
   <xsd:restriction base="xsd:string">
      <xsd:enumeration value="Tom"/>
      <xsd:enumeration value="Cheryl"/>
      <xsd:enumeration value="Marc"/>
      <xsd:enumeration value="Joe"/>
      <xsd:enumeration value="Brian"/>
   </xsd:restriction>
</xsd:simpleType>
<xsd:simpleType name="RogersFamily">
   <xsd:restriction base="xsd:string">
      <xsd:enumeration value="Roger"/>
   </xsd:restriction>
</xsd:simpleType>
<xsd:simpleType name="JohnsFamily">
   <xsd:restriction base="xsd:string">
      <xsd:enumeration value="John"/>
   </xsd:restriction>
</xsd:simpleType>
<xsd:simpleType name="CostelloFamily">
   <xsd:union memberTypes="Parent PatsFamily BarbsFamily
                           JudysFamily TomsFamily RogersFamily
                           JohnsFamily"/>
</xsd:simpleType>
```

Cont. -->

```
   <xsd:element name="Y2KFamilyReunion">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element name="Participants">
               <xsd:complexType>
                  <xsd:sequence>
                     <xsd:element name="Name" type="CostelloFamily" minOccurs="0" maxOccurs="unbounded"/>
                  </xsd:sequence>
               </xsd:complexType>
            </xsd:element>
         </xsd:sequence>
      </xsd:complexType>
   </xsd:element>
</xsd:schema>
```

Y2KFamilyReunion.xsd (see example 20)

```
<?xml version="1.0"?>
<Y2KFamilyReunion xmlns="http://www.CostelloReunion.org"
                  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                  xsi:schemaLocation=
                                "http://www.CostelloReunion.org
                                 Y2KFamilyReunion.xsd">
    <Participants>
      <Name>Mary</Name>
      <Name>Pat</Name>
      <Name>Patti</Name>
      <Name>Christopher</Name>
      <Name>Elizabeth</Name>
      <Name>Judy</Name>
      <Name>Peter</Name>
      <Name>Tom</Name>
      <Name>Cheryl</Name>
      <Name>Marc</Name>
      <Name>Joe</Name>
      <Name>Roger</Name>
    </Participants>
</Y2KFamilyReunion>
```

Y2KFamilyReunion.xml

Alternative

```
<xsd:simpleType name="CostelloFamily">
    <xsd:union>
       <xsd:simpleType>
          <xsd:restriction base="xsd:string">
             <xsd:enumeration value="Mary"/>
          </xsd:restriction>
       </xsd:simpleType>
       <xsd:simpleType>
          <xsd:restriction base="xsd:string">
             <xsd:enumeration value="Pat"/>
             <xsd:enumeration value="Patti"/>
             <xsd:enumeration value="Christopher"/>
             <xsd:enumeration value="Elizabeth"/>
          </xsd:restriction>
       </xsd:simpleType>
       …
    </xsd:union>
</xsd:simpleType>
```

Version 2 of Y2KFamilyReunion.xsd (see example 21)

A union of anonymous simpleTypes

The disadvantage of creating the union type in this manner is that none of the simpleTypes are reusable.

Review of Union simpleType

```
<xsd:simpleType name="name">
    <xsd:union memberTypes="space delimited simpleTypes"/>
</xsd:simpleType>
```

Alternatively,

```
<xsd:simpleType name="name">
  <xsd:union>
    <xsd:simpleType>
      …
    </xsd:simpleType>
    <xsd:simpleType>
      …
    </xsd:simpleType>
    …
  </xsd:union>
</xsd:simpleType>
```

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>
```

Summary of Defining simpleTypes

3. **simpleType that defines a list type:**

```
<xsd:simpleType name= "LotteryNumbers">
   <xsd:list itemType="OneToNinetyNine"/>
</xsd:simpleType>
```

where the datatype OneToNinetyNine is declared as:

```
<xsd:simpleType name= "OneToNinetyNine">
   <xsd:restriction base="xsd:positiveInteger">
      <xsd:maxInclusive value="99"/>
   </xsd:restriction>
</xsd:simpleType>
```

4. **An alternate form of the above, where the list's datatype is specified using an inlined simpleType:**

```
<xsd:simpleType name= "LotteryNumbers">
   <xsd:list>
      <xsd:simpleType>
         <xsd:restriction base="xsd:positiveInteger">
            <xsd:maxInclusive value="99"/>
         </xsd:restriction>
      </xsd:simpleType>
   </xsd:list>
</xsd:simpleType>
```

Summary of Defining simpleTypes

5. **simpleType that defines a union type:**

```
<xsd:simpleType name= "maxOccurs">
   <xsd:union memberTypes="xsd:nonNegativeInteger  UnboundedType"/>
</xsd:simpleType>
```

where the datatype UnboundedType is declared as:

```
<xsd:simpleType name= "UnboundedType">
   <xsd:restriction base="xsd:string">
      <xsd:enumeration value="unbounded"/>
   </xsd:restriction>
</xsd:simpleType>
```

6. **An alternate form of the above, where the datatype UnboundedType is specified using an inline simpleType:**

```
<xsd:simpleType name= "maxOccurs">
   <xsd:union memberTypes="xsd:nonNegativeInteger">
      <xsd:simpleType>
         <xsd:restriction base="xsd:string">
            <xsd:enumeration value="unbounded"/>
         </xsd:restriction>
      </xsd:simpleType>
   </xsd:union>
</xsd:simpleType>
```