

XML Schemas

<http://www.w3.org/TR/xmlschema-0/> (Primer)

<http://www.w3.org/TR/xmlschema-1/> (Structures)

<http://www.w3.org/TR/xmlschema-2/> (Datatypes)

What is XML Schema?

- XML Schema is vocabulary for expressing constraints for the validity of an XML document.
- A piece of XML is valid if it satisfies the constraints expressed in another XML file, the schema file.
- The idea is to check if the XML file is fit for a certain purpose.

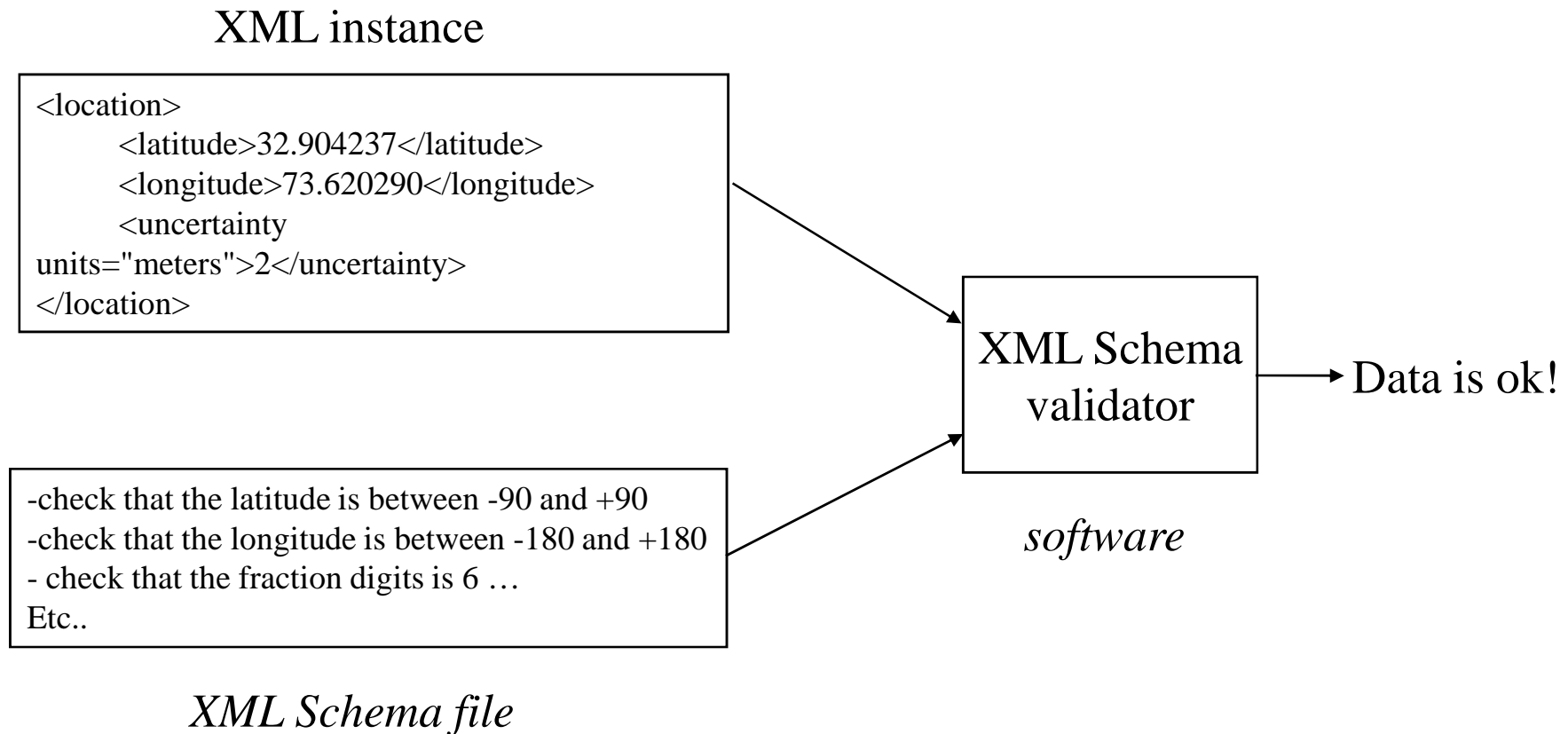
Example

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

To be valid, this XML snippet must meet all the following constraints:

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.

Validating your data



History of Schema

- Once upon a time, there was SGML
- SGML has a “schema” language called a DTD.
- It is crap
 - Different syntax than SGML
 - Main focus on presence and absence of elements
 - Very limited capabilities to check contents of elements (datatypes)

DTD versus Schema

Limitations of DTD

- No support for basic data types like integers, doubles, dates, times, ...
- No structured, self-definable data types
- No type derivation
- id/idref links are quite loose (target is not specified)
- No constraints on character data
 - Not using XML syntax
 - No support for namespace
 - Very limited for reusability and extensibility

Advantages of Schema

- Syntax in XML Style
- Supporting Namespace and import/include
- More data types
- Able to create complex data type by inheritance
- Inheritance by extension or restriction
- More ...

XML Schemas can constrain

- 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"

Highlights of XML Schemas

- 44 built-in datatypes
- Can create your own datatypes by extending or restricting existing datatypes
- Written in the same syntax as instance documents
- 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

Important schema concepts

- Simple types: types that can not have child elements
 - elements that only have text contents and no attributes
 - attributes
- Complex type: type of anything that can have child attributes

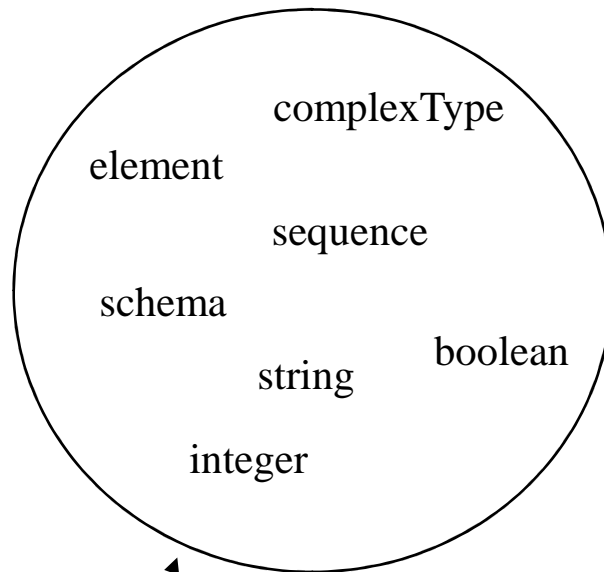
Important schema concepts

- Global declarations are direct children of the root schema element. They are visible everywhere.
- Local declarations are limited in scope to the element that they appear within

Namespaces

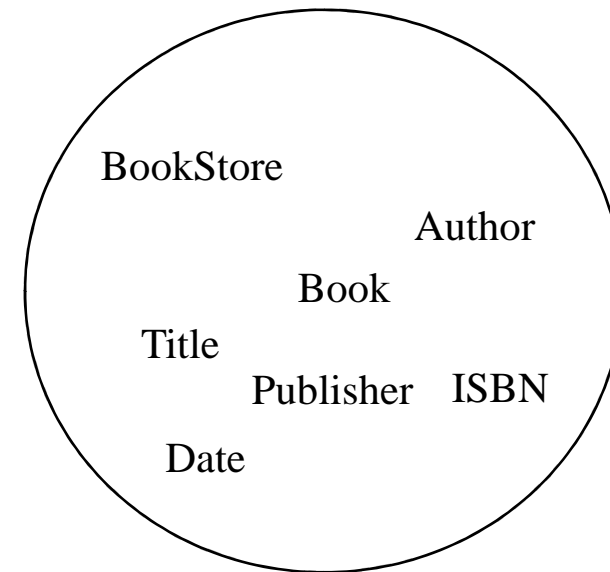
- XML Schema file mixes vocabulary from the XML Schema language with own vocabulary to be created.
- Has to keep both separate using namespaces.
- Namespaces associate a URI with names.

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



↖
This is the vocabulary that
XML Schemas provide to define your
new vocabulary

<http://www.books.org> (*targetNamespace*)



↖
This is the vocabulary for our book
store xml description.

```

<?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

xsd = Xml-Schema Definition

```

<?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>

```

```

<?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>

```

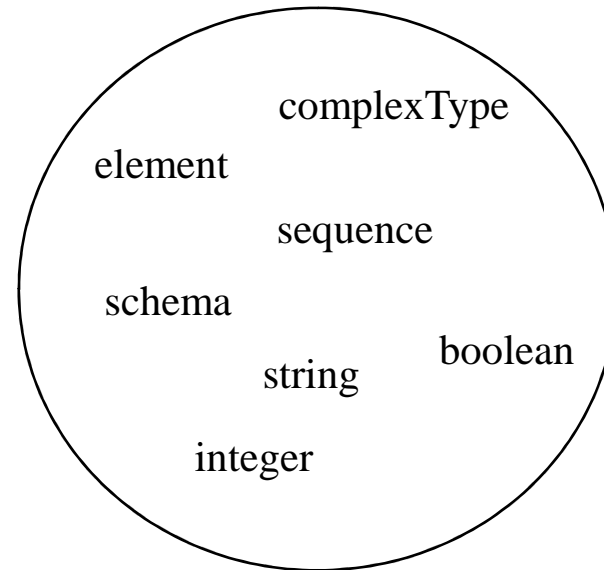
The elements and datatypes that are used to construct schemas

- 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>

```

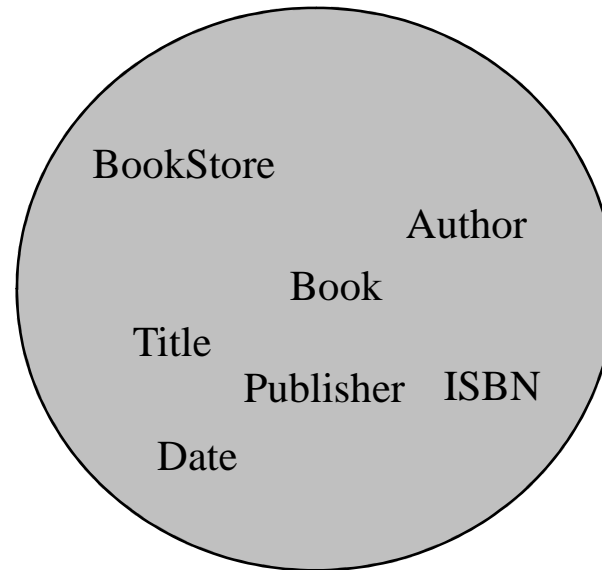
Says that the elements defined by this schema

- BookStore
- Book
- Title
- Author
- Date
- ISBN
- Publisher

are to go in this 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?

```

<?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 that are defined in this schema must be namespace-qualified when used in instance documents.

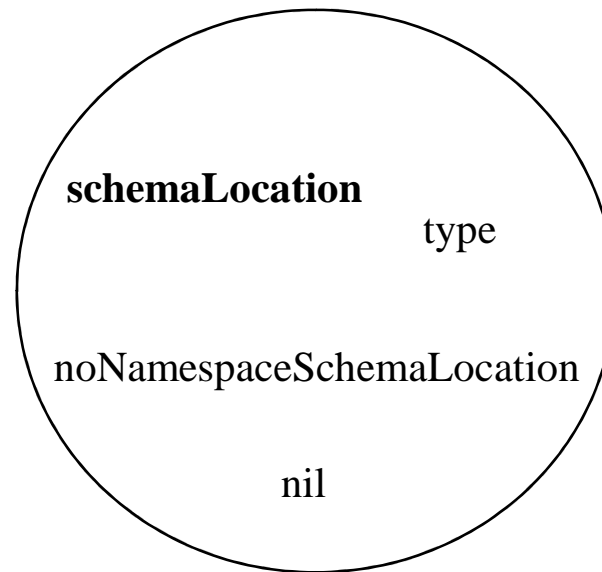
Referencing a schema in an XML instance document

```
<?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>
```

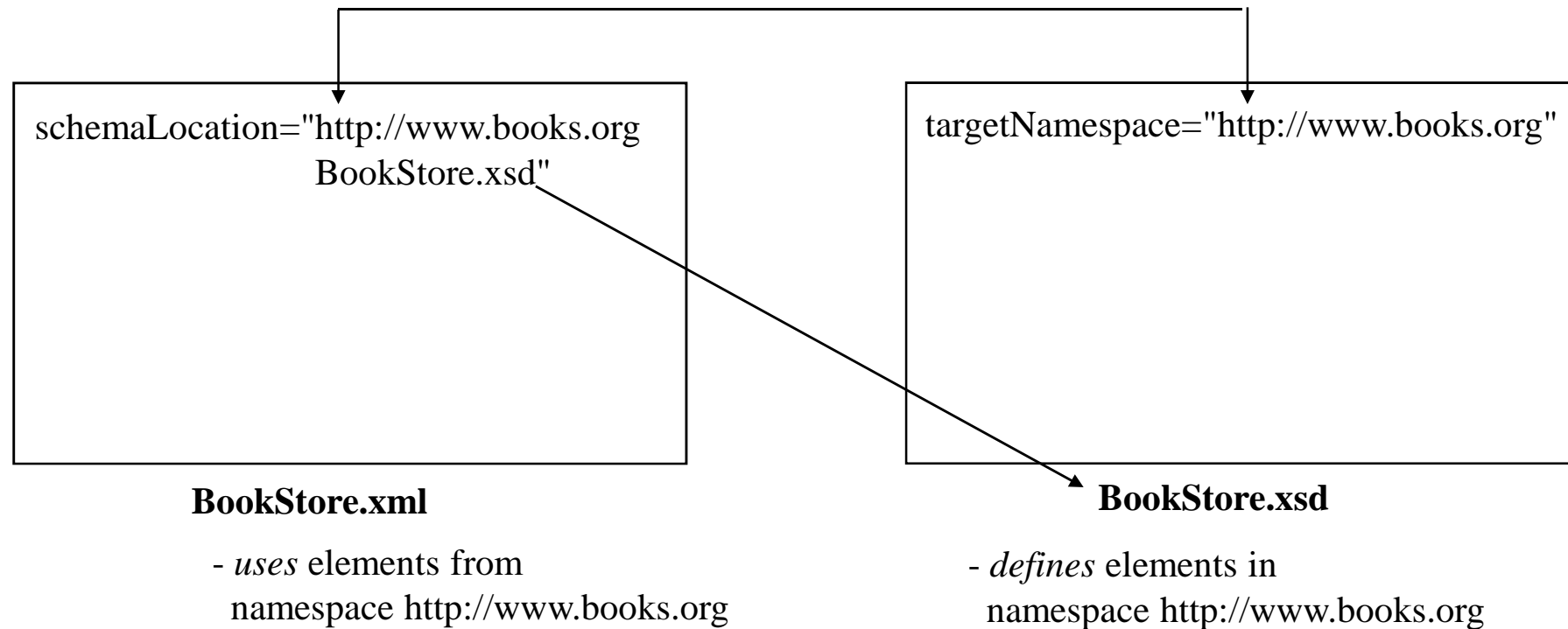
1. First, using a default namespace declaration, tell the schema-validator that all of the elements used in this instance document come from the *http://www.books.org* namespace.
2. Second, with `schemaLocation` tell the schema-validator that the *http://www.books.org* namespace is defined by `BookStore.xsd` (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 XML Schema-instance namespace.

XMLSchema-instance Namespace

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

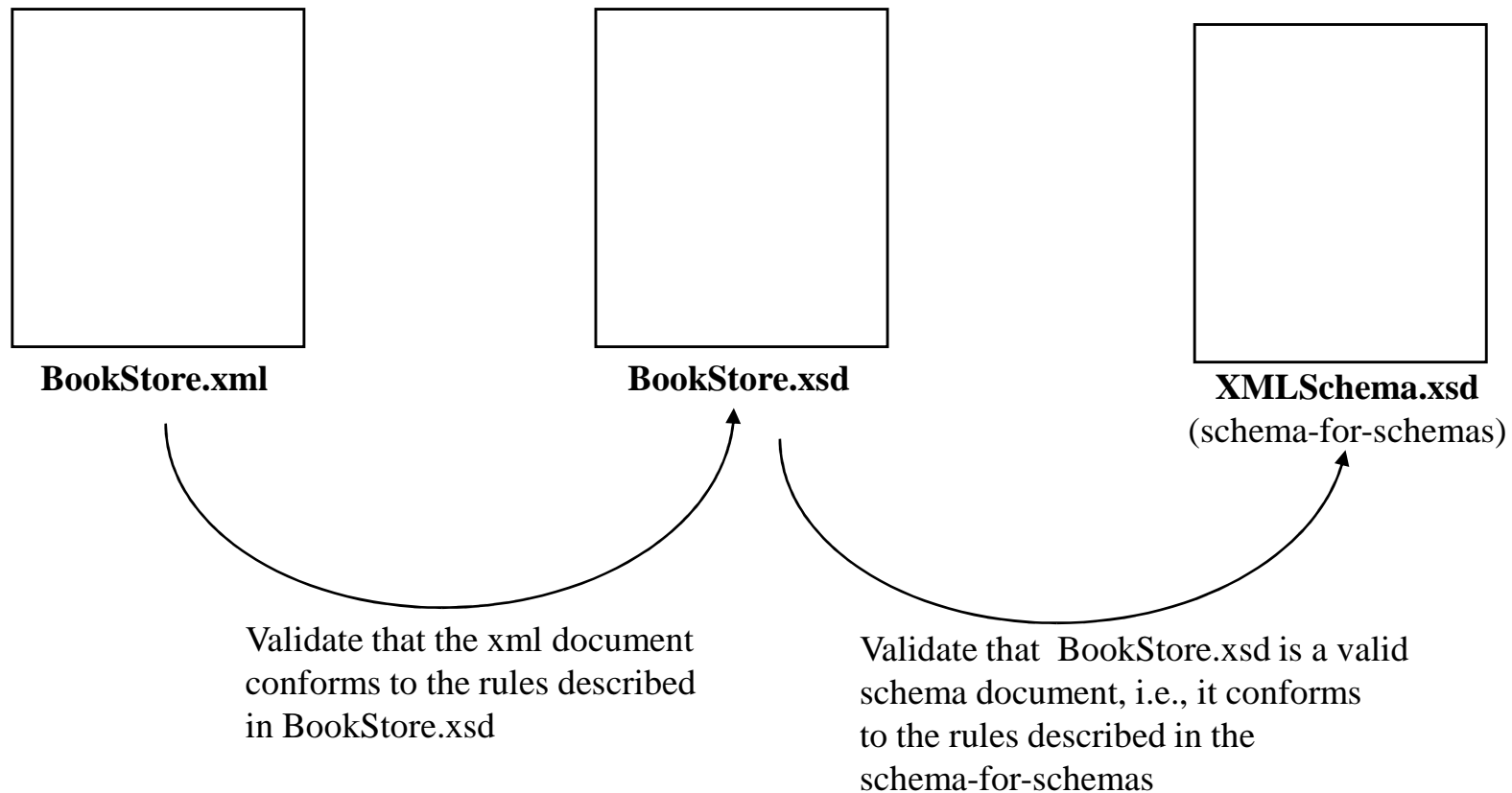


Referencing a schema in an XML instance document



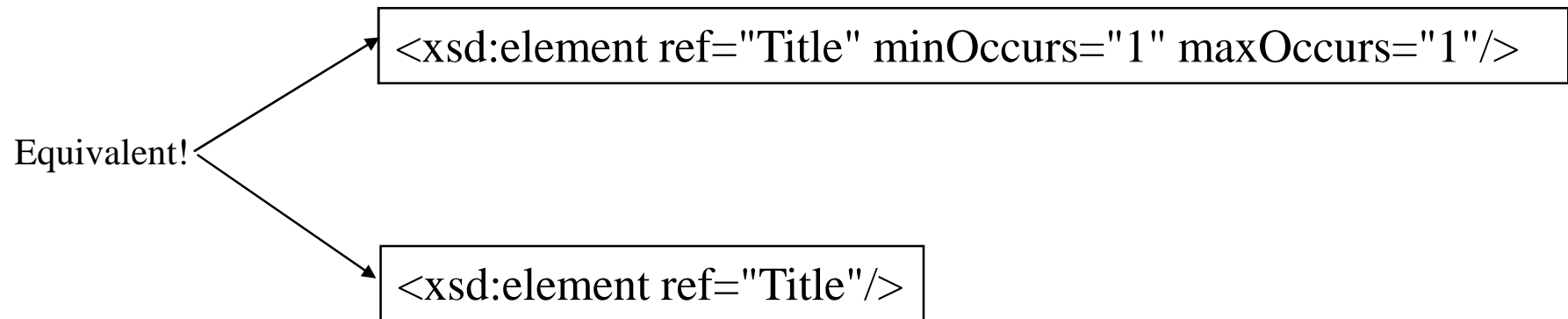
A schema *defines* a new vocabulary. Instance documents *use* that new vocabulary.

Note multiple levels of checking



Default Value for minOccurs and maxOccurs

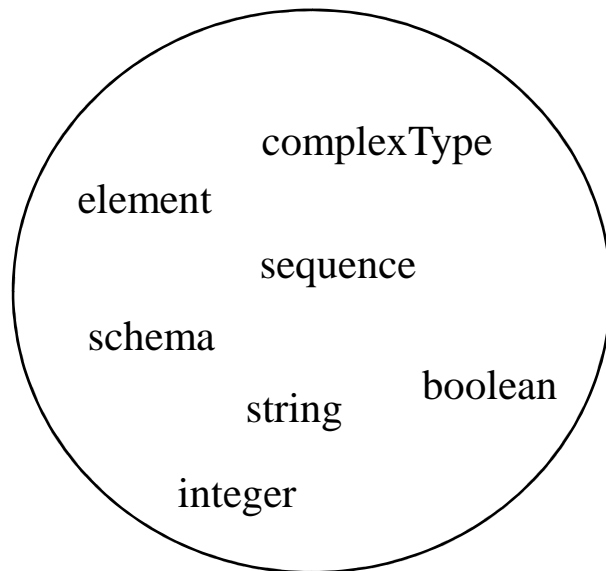
- The default value for minOccurs is "1"
- The default value for maxOccurs is "1"



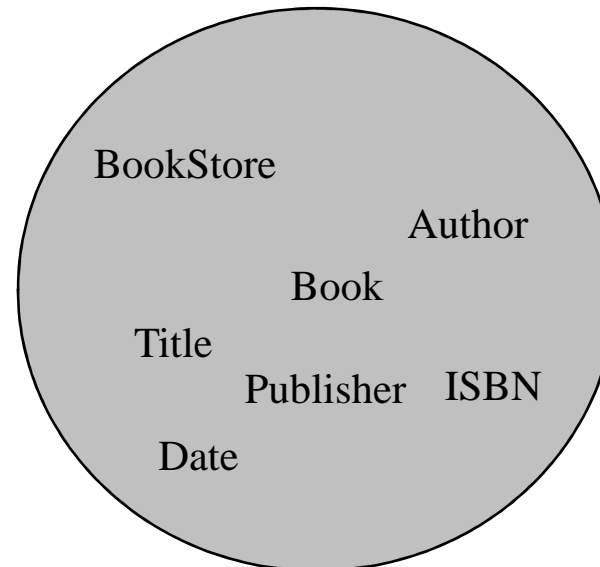
Qualify XMLSchema, Default targetNamespace

- 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>



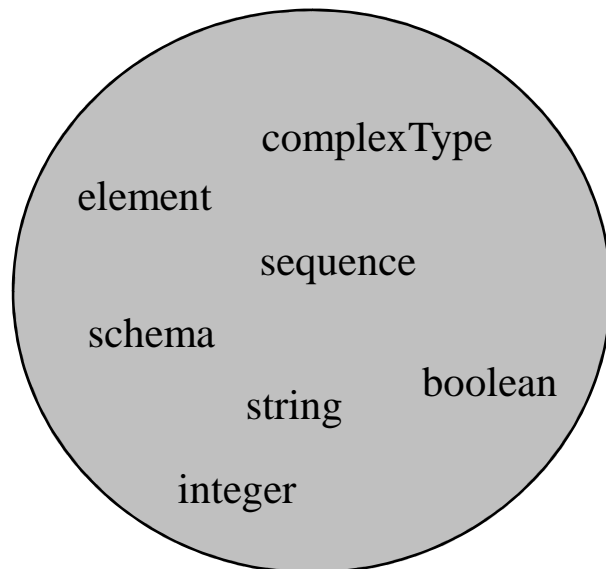
<http://www.books.org> (targetNamespace)



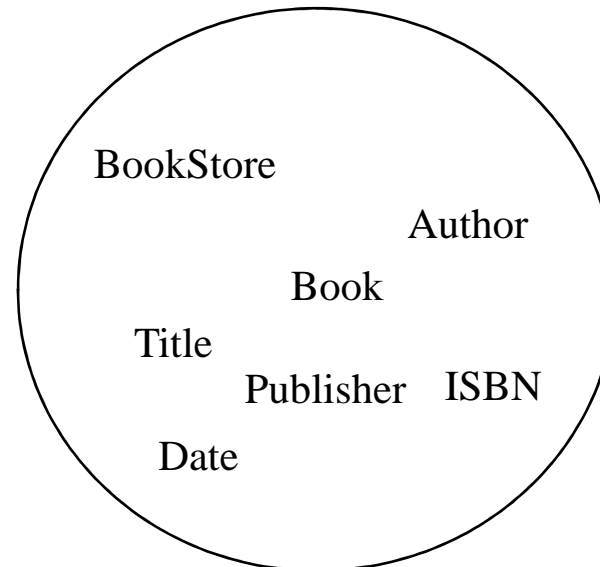
Default XMLSchema, Qualify targetNamespace

- 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://www.w3.org/2001/XMLSchema>
 is the default
 namespace.
 Consequently, there
 are no namespace
 qualifiers on

- schema
- element
- complexType
- sequence
- string

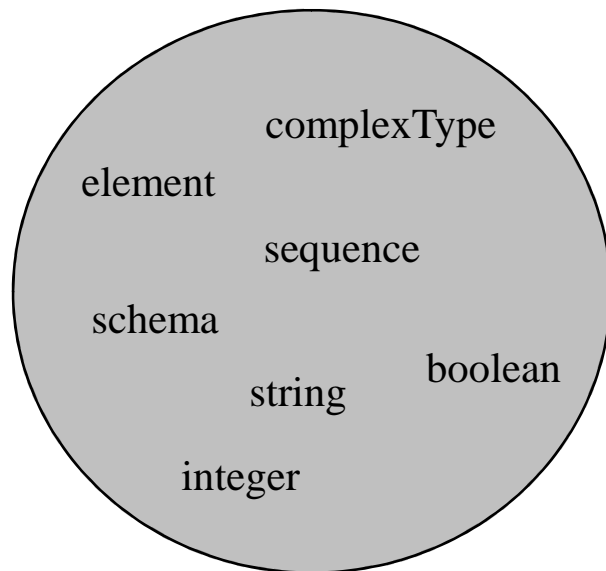
(see example02)

```
<?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>
```

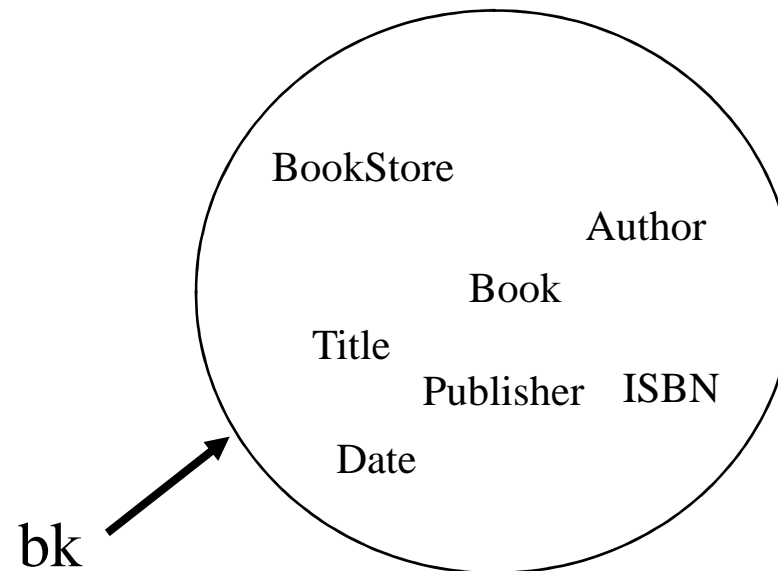
Here we are
referencing a
Book element.
Where is that
Book element
defined? In
what namespace?

"bk:" References the targetNamespace

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



<http://www.books.org> (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.


```

<?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!

```
<?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

```
<?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>
```



Named type

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>
```

<xsd:complexType> and <xsd:simpleType>

- 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)

Refining our data

- 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).
 - Done in the 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 gYear built-in 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

The date built-in Datatype

- Elements declared to be of type date must follow this form: CCYY-MM-DD
 - range for CC and 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

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)

Primitive Built-in 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*

Note: 'T' is the date/time separator
 INF = infinity
 NAN = not-a-number

Primitive Built-in Datatypes

- 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

Derived Built-in Datatypes (cont.)

- `normalizedString` – A string without tabs, line feeds, or carriage returns
- `Token` – String w/o tabs, 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` – **part** (no namespace qualifier)
- `NCName`
- `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
 - negativeInteger
 - Long
 - int
 - short
 - byte
 - nonNegativeInteger
 - unsignedLong
 - unsignedInt
 - unsignedShort
 - unsignedByte
 - positiveInteger
- Subtype of primitive datatype
 - negative infinity to -1
 - **-9223372036854775808** to **9223372036854775807**
 - **-2147483648** to **2147483647**
 - **-32768** to **32767**
 - **-127** to **128**
 - 0 to infinity
 - **0** to **18446744073709551615**
 - **0** to **4294967295**
 - **0** to **65535**
 - **0** to **255**
 - **1** to infinity

```

<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>

```

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

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

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"

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'.

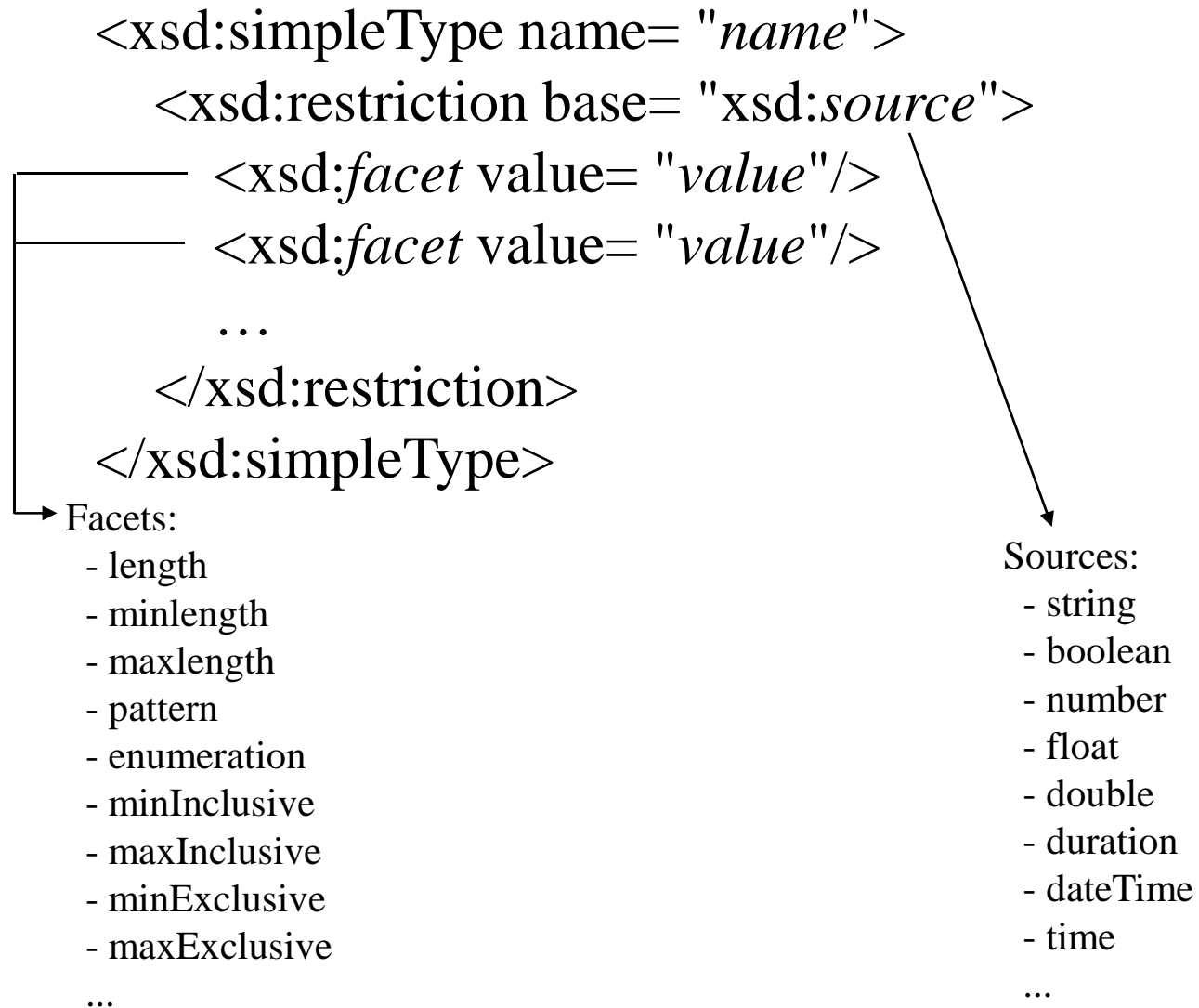
Another Example

```
<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.

General Form of Creating a New Datatype by Specifying Facet Values



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.

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 others facets => "*and*" them

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.


```
<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>
```

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" />
```

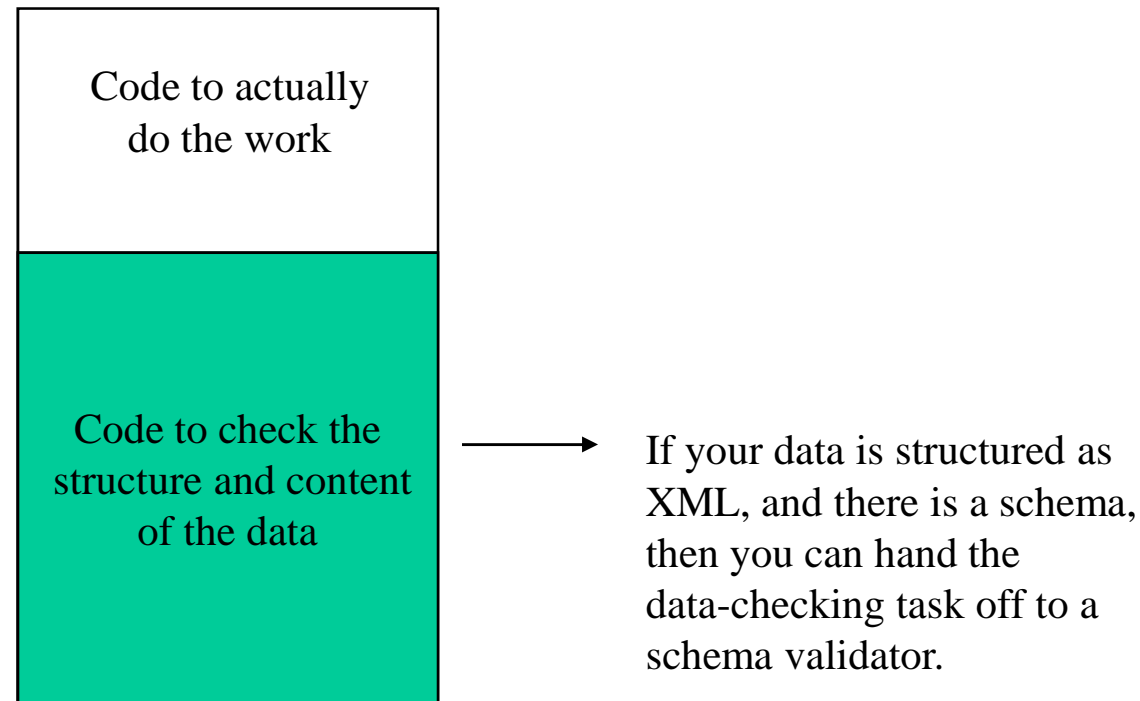
2

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

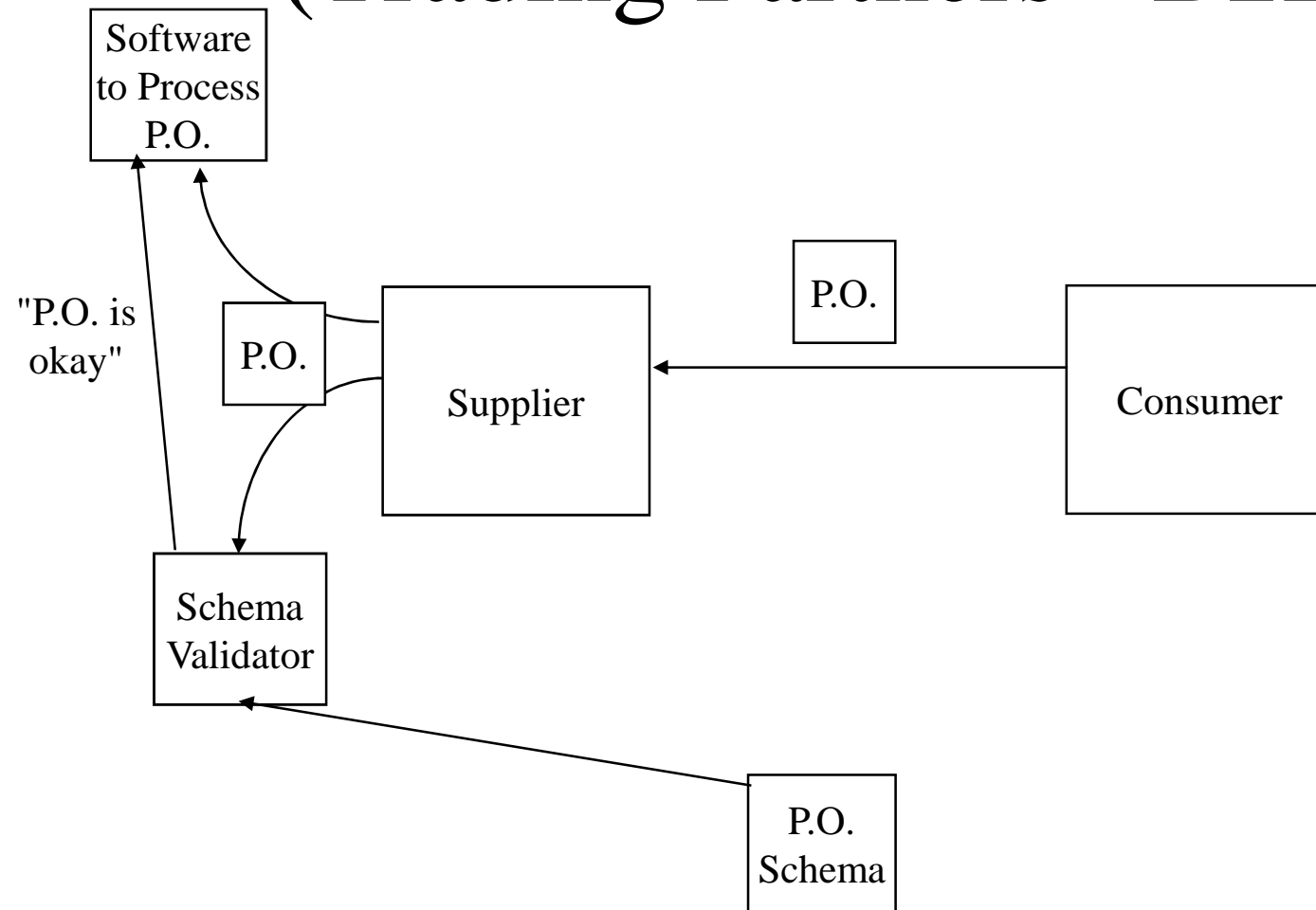
3

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

In a typical program, 60% of code is used to check the input.



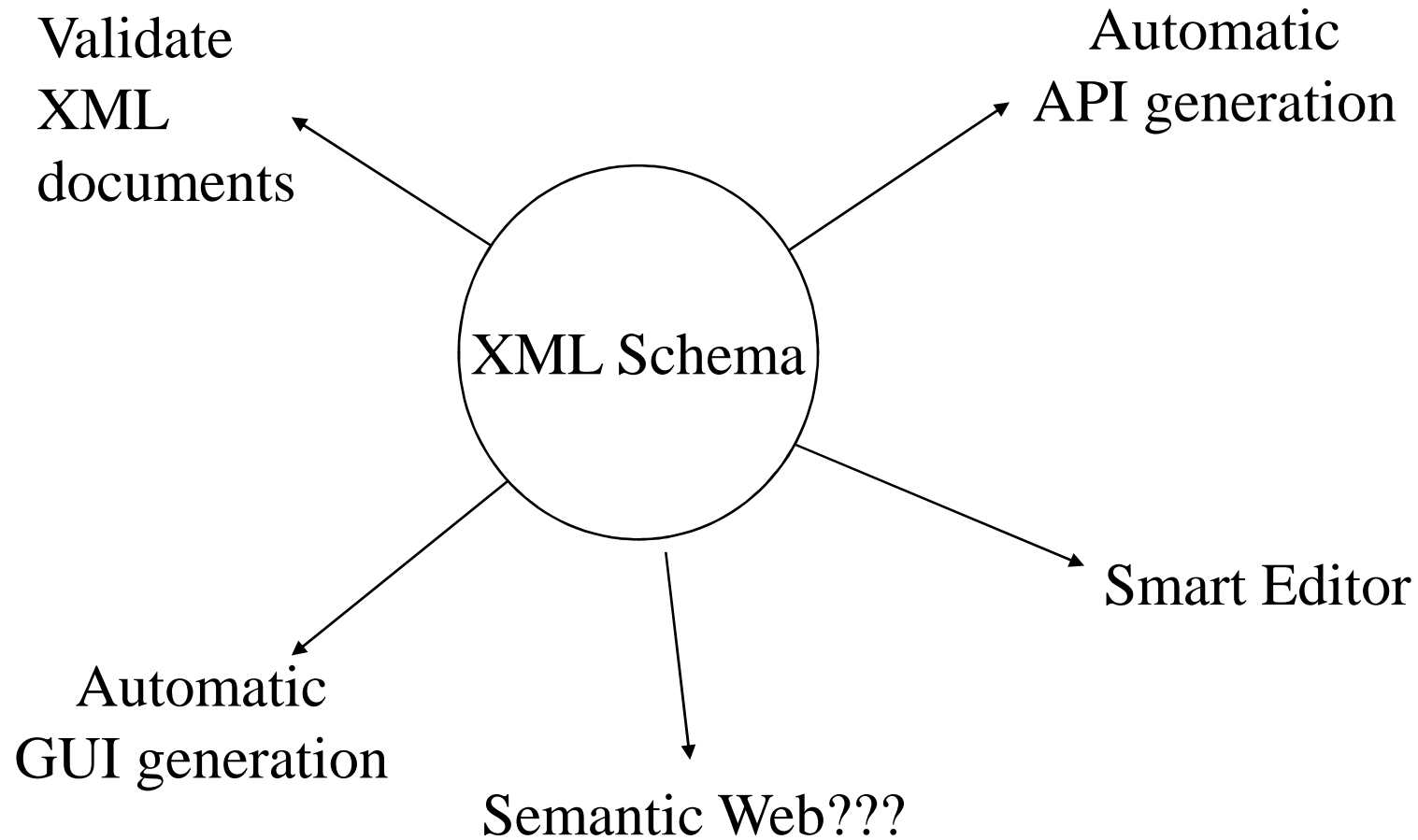
Classic use of XML Schemas (Trading Partners - B2B)



Other aspects of XML schemas

- Organizations agree to structure their XML documents in conformance with an XML Schema. Thus, the XML Schema acts as a contract between the organizations.
- 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

And more ...



Describing Metadata using Schemas

- XML Schema Strategy - two documents are used to provide metadata:
 - a schema document specifies the properties (metadata) for a class of resources (objects);
 - each instance document provides specific values for the properties.

XML Schema: Specifies the Properties for a Class of Resources

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

"For the class of Book resources, we identify five properties - Title, Author, Date, ISBN, and Publisher"

XML Instance Document: Specifies Values for the Properties

```
<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>
```

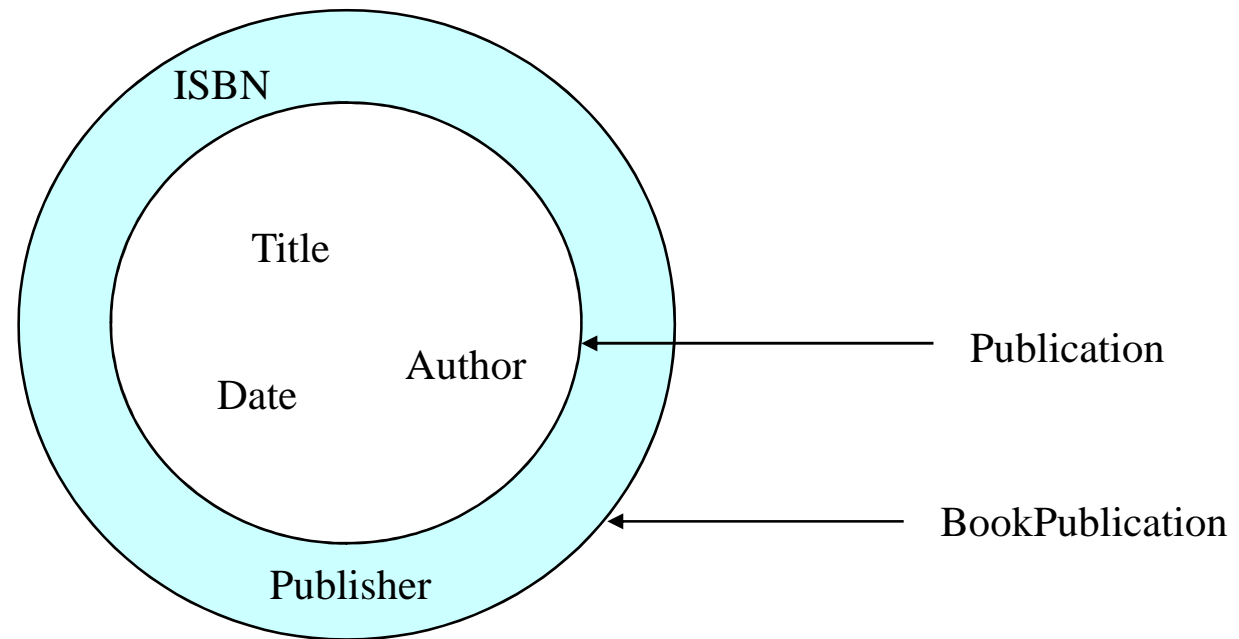
"For a specific instance of a Book resource, here are the values for the properties.

Use schemaLocation to identify the companion document (i.e., the schema) which defines the Book class of resources."

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.
 - 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**.

Derived Types by extension





```

<?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.

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 there are advantages to explicitly associating a type with a base type. 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`.

Prohibiting Derivations

```
<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: Declaration vs Definition

- 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: <ul style="list-style-type: none">- element declarations- attribute declarations	Definitions: <ul style="list-style-type: none">- type (simple, complex) definitions- attribute group definitions- model group definitions
--	--

Summary of Declaring 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>
```

Declaring Attributes

- Declare a required attribute Category that can take the value “autobiography” “fiction” and “non-fiction”.
- Declare an optional attribute inStock that can take the values “true” or “false”, true by default.
- Declare an optional argument Reviewer.


```

<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" 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: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>

```

Diagram illustrating the structure of the **BookStore** XML schema:

- The **BookStore** element contains a sequence of **Book** elements.
- The **Book** element is a complex type containing a sequence of elements: **Title**, **Author**, **Date**, **ISBN**, and **Publisher**.
- The **Book** element also has an **attributeGroup** named **BookAttributes**.
- The **BookAttributes** group contains three attributes:
 - Category**: A required attribute of type **xsd:string** with a restriction to the values **autobiography**, **non-fiction**, and **fiction**.
 - InStock**: A boolean attribute with a default value of **false**.
 - Reviewer**: A string attribute with a default value of an empty string.

Arrows indicate the mapping from the attribute definitions to their respective values:

- Category → Category
- InStock → InStock
- Reviewer → Reviewer

```
<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>
```

Attributes are simpleTypes because they can not 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

└─┬─┘
│
Do not use the "use"
attribute if you use either
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>
```

Inlining Attributes

- 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>
```

```

<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>

```

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

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?


```

<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.

Summary: declaring elements

1 Elements with simple contents

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="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="square"/>  
    </xsd:restriction>  
  </xsd:simpleType>  
</xsd:element>
```

2 Elements that have children

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

```
<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"/>
```

3. Extending 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"/>
```

4. Restricting 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"/>
```

5. Simple content with 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 / simpleContent

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

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

versus

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

X must be a complexType

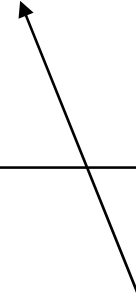
Y must be a simpleType

Expressing Repeatable Choice

Example: define the element binary-string as a repetition of elements zero and 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/>
```

Xsd:sequence and xsd:choice

Repeat work and eat, then once work or play, the whole thing any number of times

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="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,..., *in any order*

XML Schema:

```
<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

Problem: declare an empty element with an attribute

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>

```

(see example 13)

Instance doc (snippet):

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

Thank You