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)

Roger L. Costello

XML Technologies Course

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

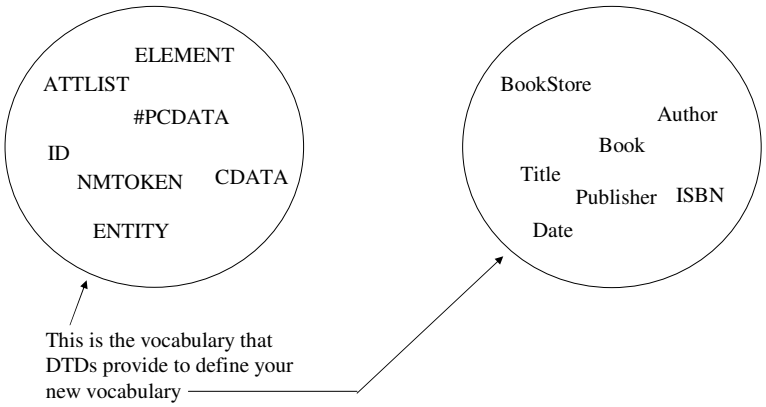
- XML Schemas are a tremendous advancement over DTDs:
  - Enhanced datatypes
    - 44+ versus 10
    - Can create your own datatypes
      - Example: "This is a new type based on the string type and elements of this type must follow this pattern: ddd-dddd, where 'd' represents a digit".
  - Written in 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.

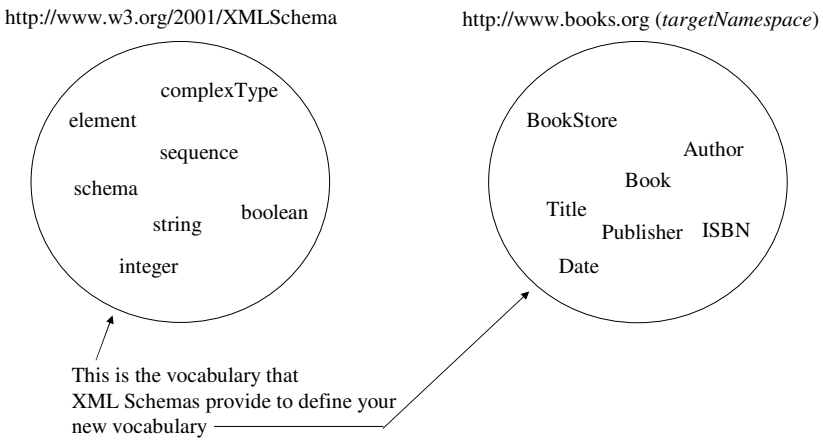
Let's Get Started!

- 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>
```

(explanations on succeeding pages)

BookStore.xsd (see example01)

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>
```

<!ELEMENT BookStore (Book)+>

```
  <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>
```

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

```
  <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 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.

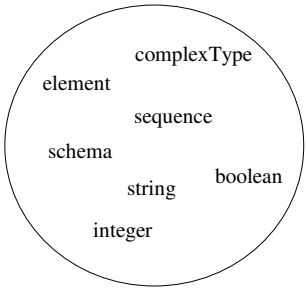

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            targetNamespace="http://www.books.org"
            xmlns="http://www.books.org"
            elementFormDefault="qualified">
  <xsd:element name="BookStore">
    <xsd:complexType>
      <xsd:sequence>
     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



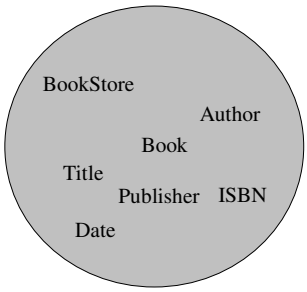

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            targetNamespace="http://www.books.org"
            xmlns="http://www.books.org"
            elementFormDefault="qualified">
  <xsd:element name="BookStore">
    <xsd:complexType>
      <xsd:sequence>
     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

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? 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>
```

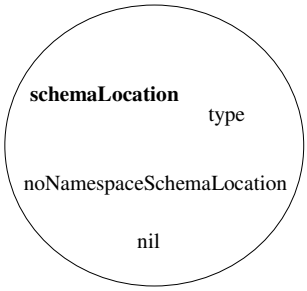
1. First, using a default namespace declaration, tell the schema-validator that all of the elements used in this instance document come from the *http://www.books.org* namespace.

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**).

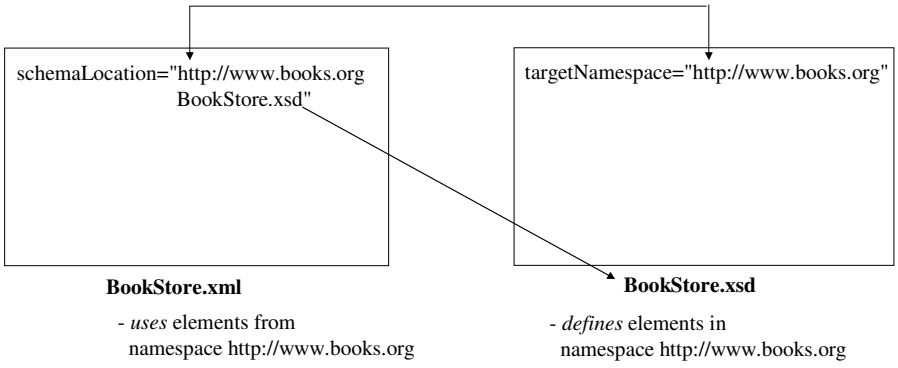
3. Third, tell the schema-validator that the schemaLocation attribute we are using is the one in the XMLSchema-instance namespace.

XMLSchema-instance Namespace

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

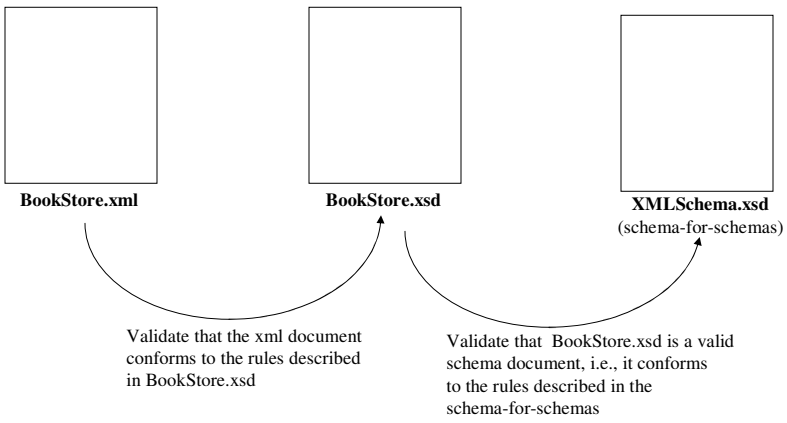


Referencing a schema in an XML instance document



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"

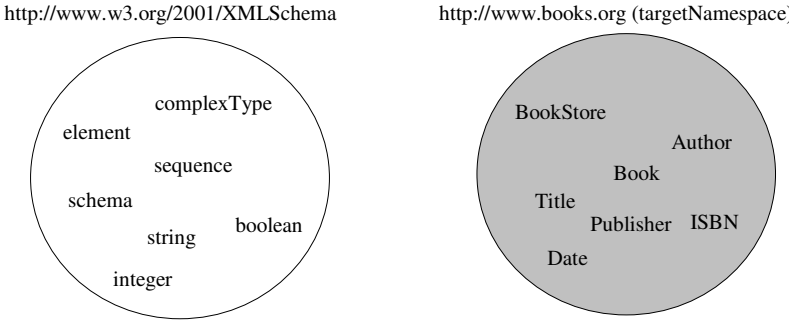
Equivalent!

```
<xsd:element ref="Title" minOccurs="1" maxOccurs="1"/>
```

```
<xsd:element ref="Title"/>
```

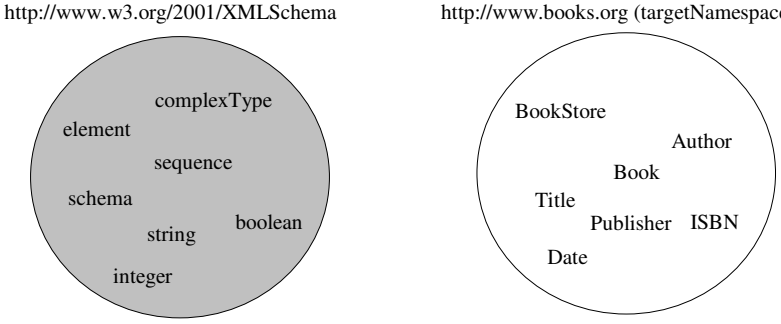
Qualify XMLSchema, Default targetNamespace

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



Default XMLSchema, Qualify targetNamespace

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



```
<?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>
```

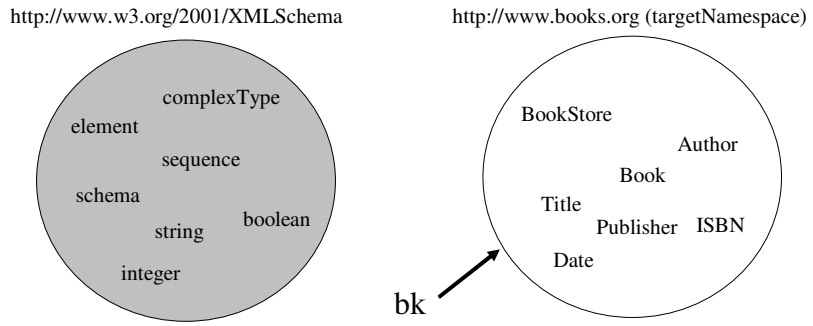
Note that http://…/XMLSchema is the default namespace. Consequently, there are no namespace qualifiers on
- schema
- element
- complexType
- sequence
- string

(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.

Do Lab1.1

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>
```

(see example03)

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)

(see example03)

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>
```

Named type

(see example04)

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>
```


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	Atomic, built-in
string	"**Hello World**"
boolean	{**true**, **false**}
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

Built-in Datatypes (cont.)

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

Built-in Datatypes (cont.)

Derived types	Subtype of primitive datatype
normalizedString	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 **9223372036854775808**
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 (which we will discuss later):
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
- ...

See DatatypeFacets.html for a mapping of datatypes to their facets.

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>
```

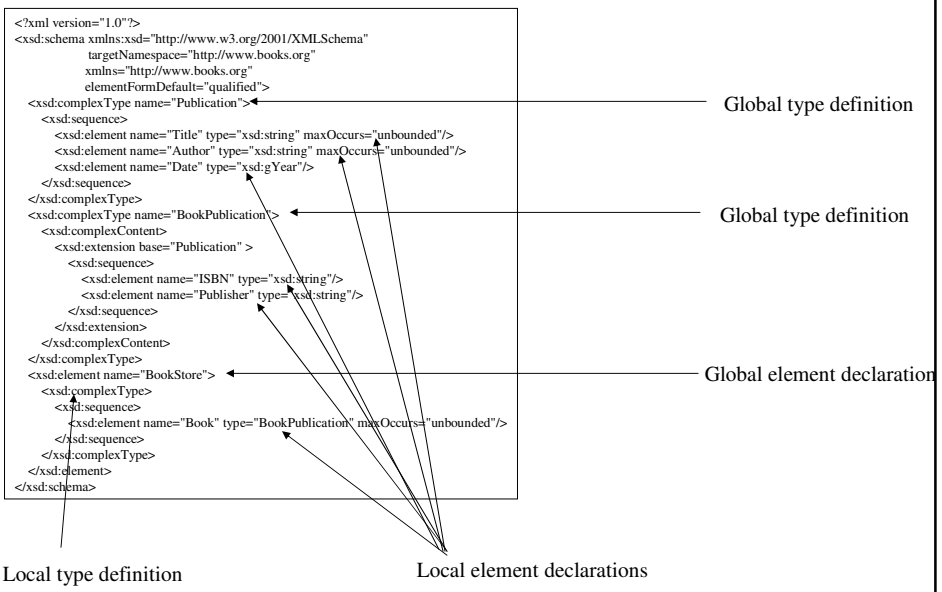
Terminology: Declaration vs Definition

- 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)>
<!ELEMENT Month (#PCDATA)>
<!ELEMENT Year (#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" 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>
```

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

→ InStock (true | false) "false"

→ Reviewer CDATA " "

(see example07)


```
<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"/>
```

- *simple-type*: xsd:string, xsd:integer, xsd:boolean, ...
- *how-its-used*: required, optional, prohibited
- default/fixed: 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>
```


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. (I only show a portion of the schema - the Book element declaration.)

```
<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"

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

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

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

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 flag 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>

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