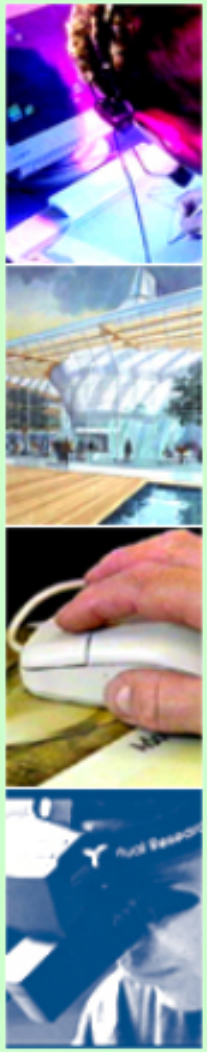
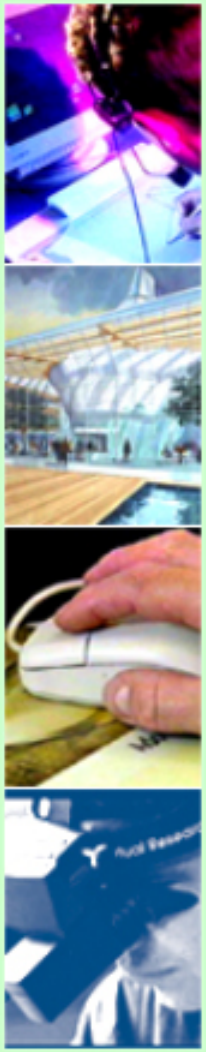
XML schemas

- XML schema is an XML based **alternative** to DTD.
- An XML vocabulary for expressing your data's business rules.
- XML schemas:
  - Are extensible for future use.
  - Are written in xml.
  - Support non textual data types.
- XML document that conforms to an XML schema is said to be "***schema valid*** "



XML schemas

- XML Schemas are a tremendous advancement over DTDs:
  - Enhanced datatypes
    - 44+
    - Can create your own datatypes
  - 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 .(xs:sequence, xs:all, xs:choice)



XML Document

```
<?xml version="1.0"?>
<note>
<to>Tove</to>
<from>Jani</from>
<subject>Reminder</subject>
<message>
Don't forget me this weekend!
</message>
</note>
```

DTD Document

```
<!ELEMENT note(to,from,subject,message)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT subject(#PCDATA)>
<!ELEMENT message (#PCDATA)>
```

Simple Xml Schema Document


```
<xs:schema
xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:element name="note">
   <xs:complexType>
     <xs:sequence>
       <xs:element name="to" type="xs:string"/>
       <xs:element name="from" type="xs:string"/>
       <xs:element name="subject" type="xs:string"/>
       <xs:element name="message" type="xs:string"/>
     </xs:sequence>
  </xs:complexType>
</xs:element>
</xs:schema>
```

Schema Data Types

- **Simple type :**

-Do not have sub-elements

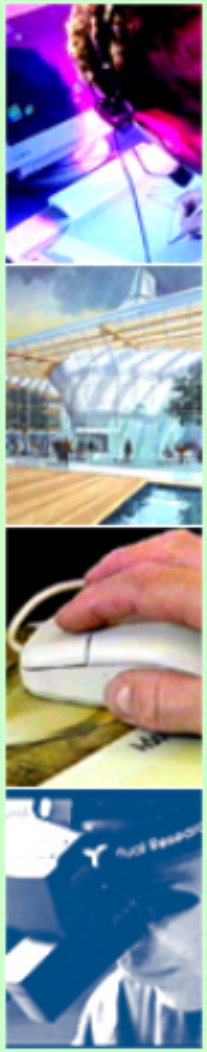
Do not have "*element*" sub-elements .

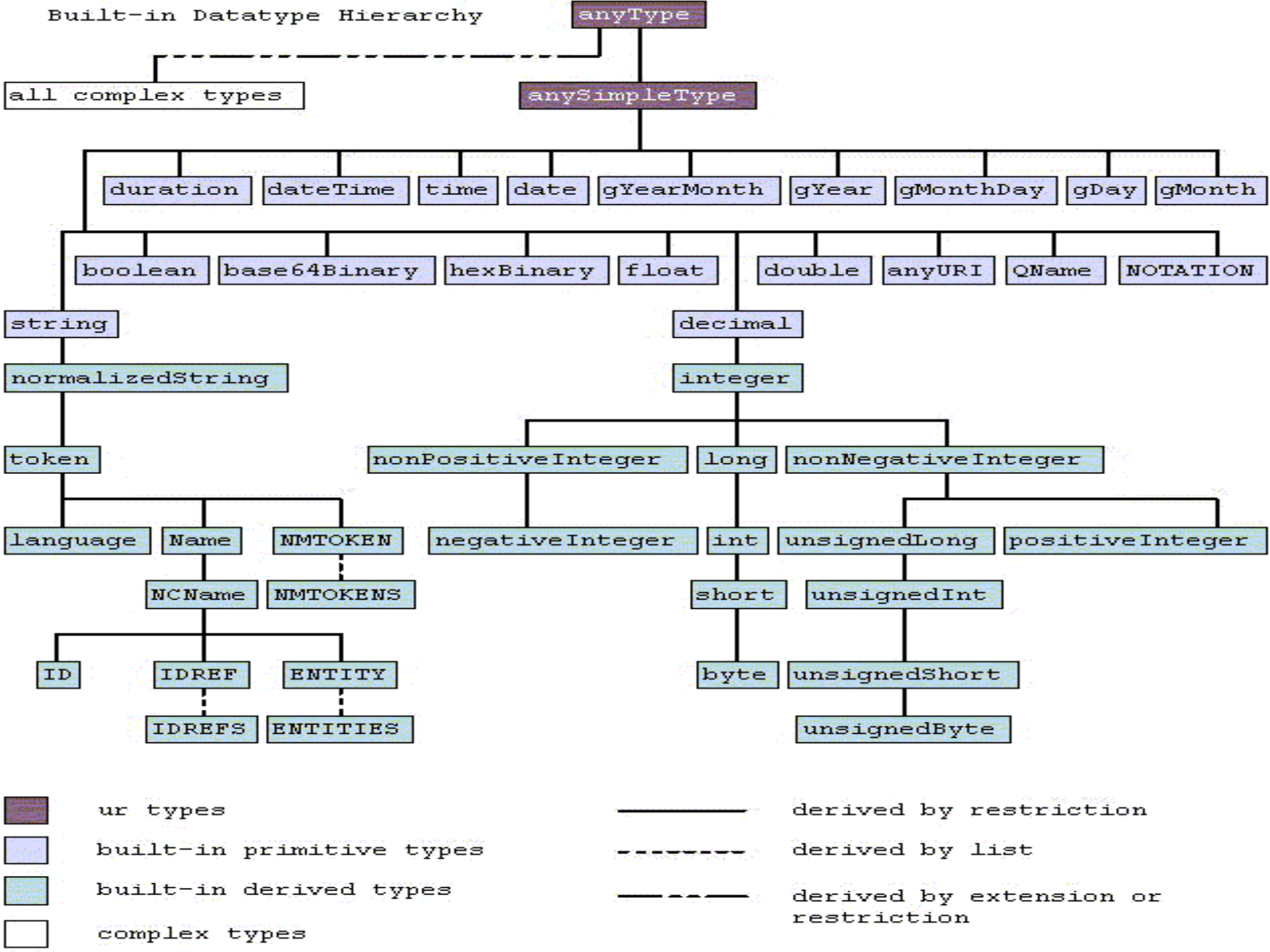
Do not have "*attribute*" sub-elements .

- Predefined type or derived from predefined type.

- **Complex type :**

-Have either "*elements*" sub-elements or "*attributes*"

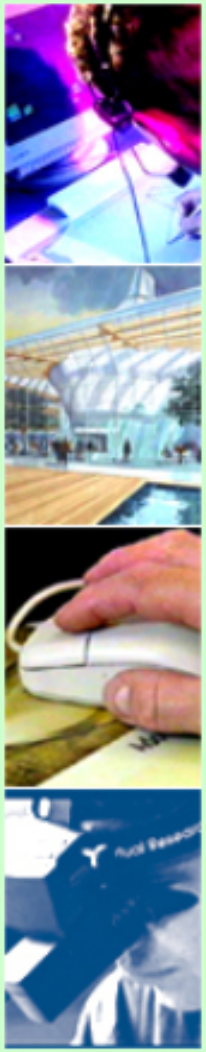



Built-in Datatype Hierarchy
anyType
all complex types
anySimpleType
duration
dateTime
time
date
gYearMonth
gYear
gMonthDay
gDay
gMonth
boolean
base64Binary
hexBinary
float
double
anyURI
QName
NOTATION
string
decimal
normalizedString
integer
token
nonPositiveInteger
long
nonNegativeInteger
language
Name
NMTOKEN
negativeInteger
int
unsignedLong
positiveInteger
NCName
NMTOKENS
short
unsignedInt
ID
IDREF
ENTITY
byte
unsignedShort
IDREFS
ENTITIES
unsignedByte
ur types
built-in primitive types
built-in derived types
complex types
derived by restriction
derived by list
derived by extension or restriction

Built-in Simple Types

Predefined Datatypes

String	"Hello World"
boolean	{true, false, 1, 0}
Decimal	7.08
dateTime	format:*CCYY-MM-DD hh:mm:ss*
time	format:*hh:mm:ss.sss*
date	format:*CCYY-MM-DD*
gYearMonth	format:*CCYY-MM*
gYear	format:*CCYY*
gMonthDay	format: *MM-DD*

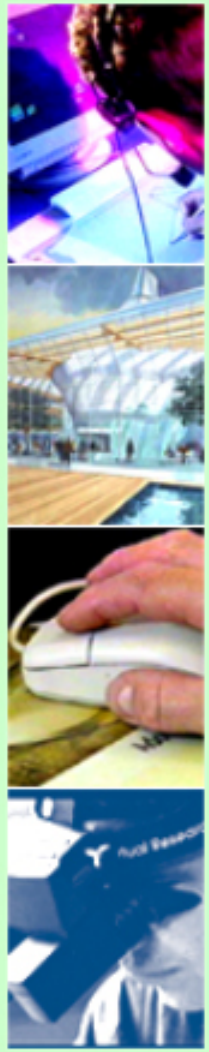


Built-in Simple Types (cont.)

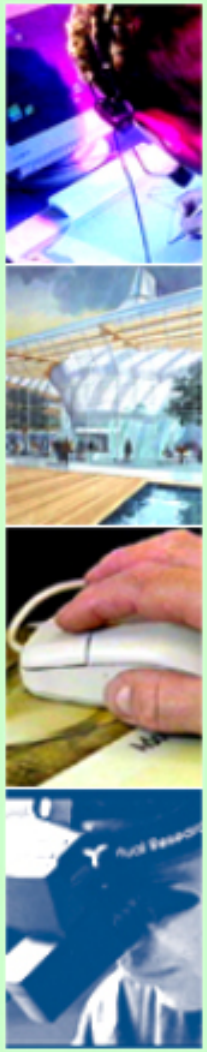
- **EX. :**

```
<xs:element name="price"
                    type="xs:decimal"/>
```

- Schemas provide data types that can be combined together to produce tailored data types.



Schema Data Types



Creating New Data Type:

- Definition :

  Create new types (both simple and complex types)

- Declaration :

  Enable elements and attributes with specific names and types (both simple and complex) to appear in document instances.

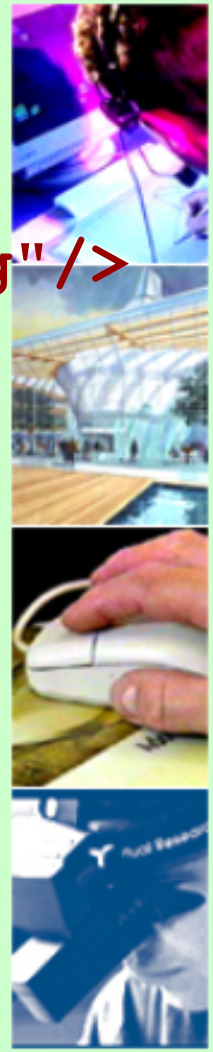
Simple Type Example

- <!-- Definition --> (In schema file)

```
<xs:simpleType name="mytype">
<xs:union memberTypes="xs:integer xs:string" />
</xs:simpleType>
```

- <!-- Declaration --> (In schema file)

```
<xs:element name="age" type="mytype"/>
```



Simple Type Example

- <!-- Definition --> (In schema file)

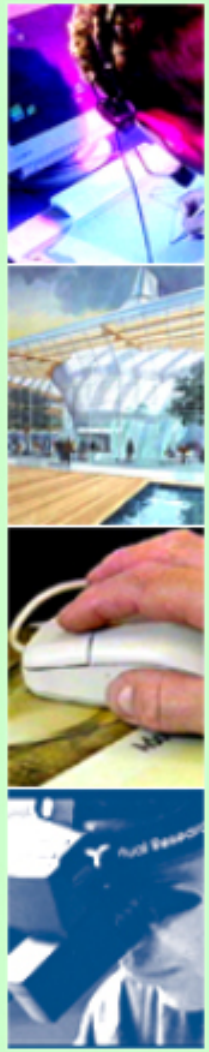
```
<xs:simpleType name="valuelist">
    <xs:list itemType="xs:integer"/>
</xs:simpleType>
```

- <!-- Declaration --> (In schema file)

```
<xs:element name="intvalues"
  type="valuelist">
```

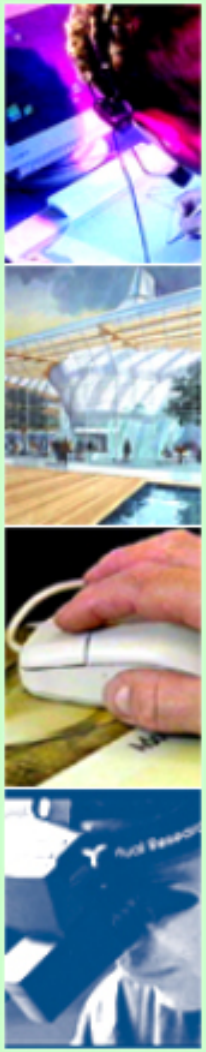
- In Xml file

```
<intvalues>100 34 56 -23 15 67</intvalues>
```



Derived Simple Type

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

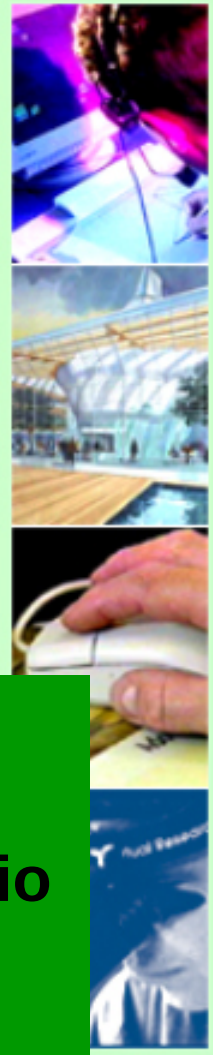


Derived Simple Type General Form

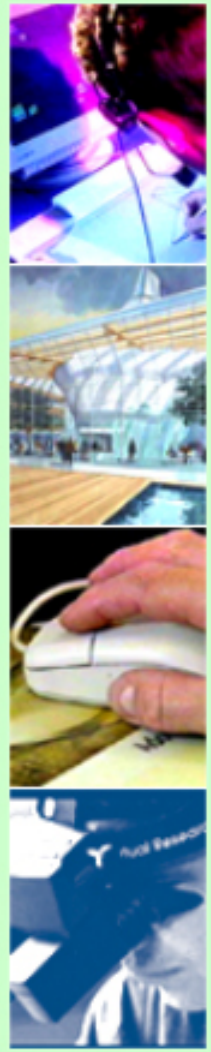
```
<xs:simpleType name= "name">
 <xs:restriction base= "xs:source">
      <xs:facet value= "value"/>
      <xs:facet value= "value"/>
            …
    </xs:restriction>
</xs:simpleType>
```

Facets:
length, maxlength, minlength
minInclusive, maxInclusive
minExclusive,maxExclusive,
Pattern, enumeration ….

Sources:
String, boolean,
float,double,duratio
n,
dateTime,time …



Derived simple type (cont.)



☞ **Ex.:**

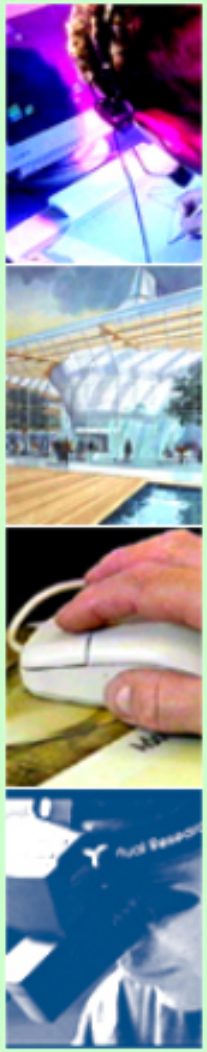
The string primitive datatype optional facets:

- **length**
- **minLength**
- **maxLength**
- **pattern**
- **enumeration**

Derived simple type (cont.)

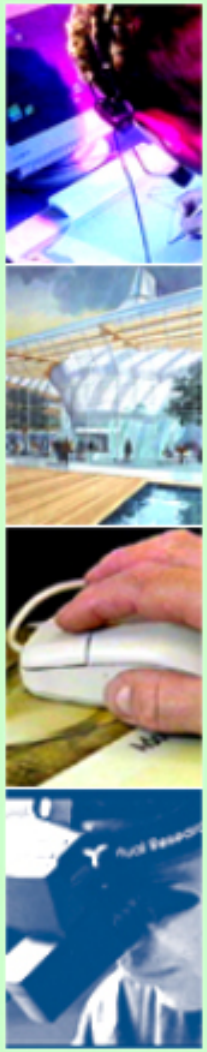
```
<xs:simpleType name="TelephoneNumber">                    ①
 <xs:restriction base="xs:string">                        ②
    <xs:length value="8"/>                                ③
    <xs:pattern value="\d{3}-\d{4}"/>                     ④
 </xs:restriction>
</xs:simpleType>
```

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.
4. The string must follow the pattern: ddd-dddd,
   where 'd' represents a 'digit'.



Derived simple type (cont.)

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

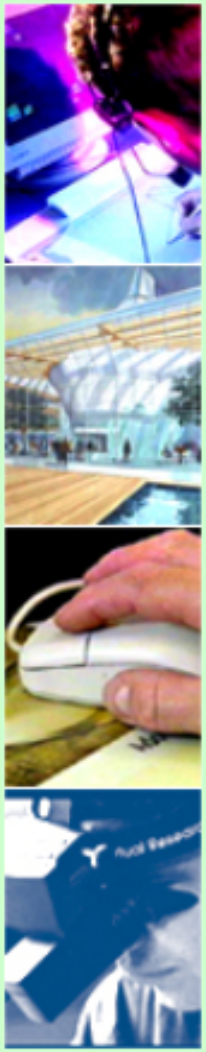


Patterns, enumerations => "or" them together

All other facets => "and" them together

Facets of the integer Datatype

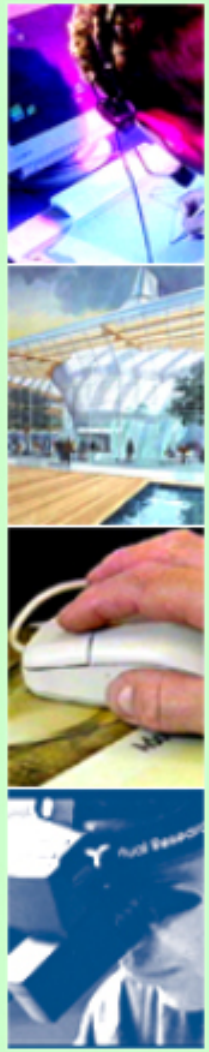
- The integer data type optional facets:
  - totalDigits
  - pattern
  - enumeration
  - maxInclusive
  - maxExclusive
  - minInclusive
  - minExclusive



Example(numeric range)

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

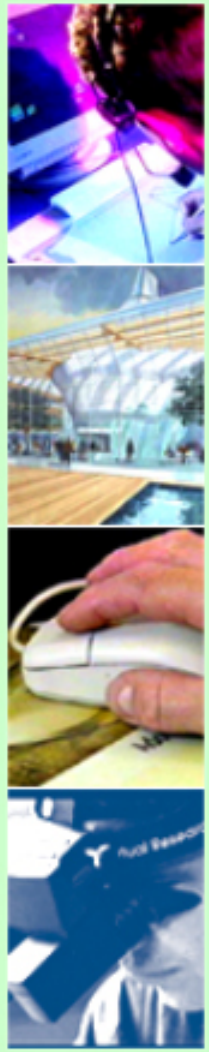
- minInclusive & maxInclusive are ***facets*** to ***integer*** type



Example(numeric range)

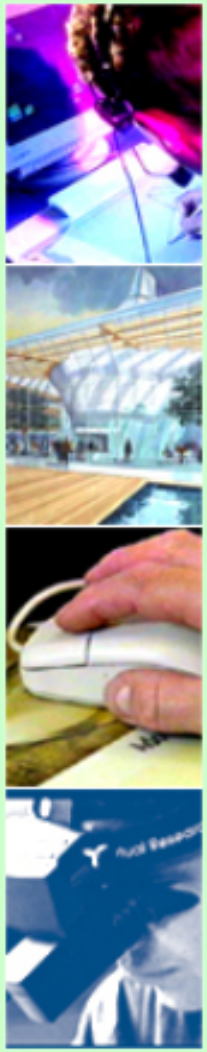
```
<simpleType name='lessthanonehundred-and-one'>
<restriction base="xs:integer">
 <maxExclusive value='101'/>
 </restriction>
</simpleType>
```

Limits values to integers less than or equal to 100.



Example(numeric range)

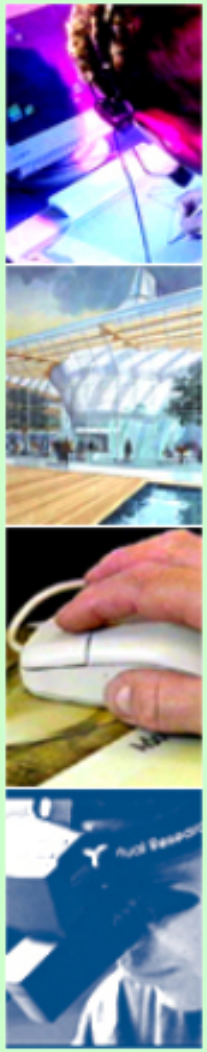
```
<simpleType name='more-than-ninety-nine'>
  <restriction base='integer'>
   <minExclusive value='99'/>
   </restriction>
</simpleType>
```



Limits values to integers greater than or equal to 100.

Creating a simpleType from another simpleType

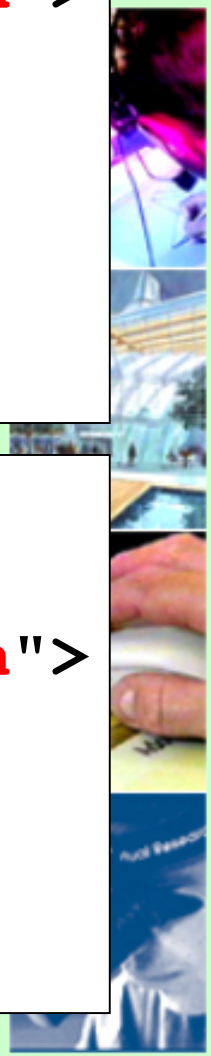
- So far we have created a simpleType using one of the built-in data types as our base type.
- However, we can create a simpleType that uses another simpleType as the base.



Example

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

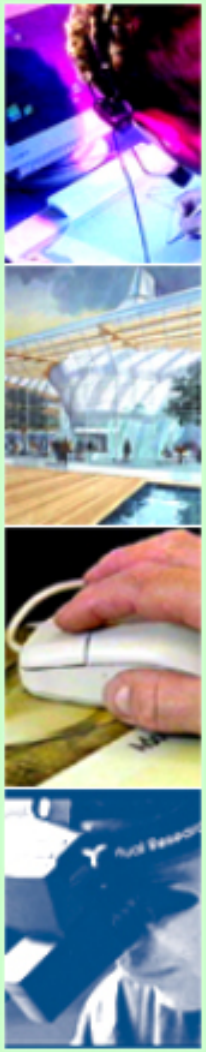
```
<xs:simpleType
     name= "BostonAreaSurfaceElevation">
<xs:restriction base="EarthSurfaceElevation">
<xs:minInclusive value="0"/>
<xs:maxInclusive value="120"/>
</xs:restriction>
</xs:simpleType>
```



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.

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

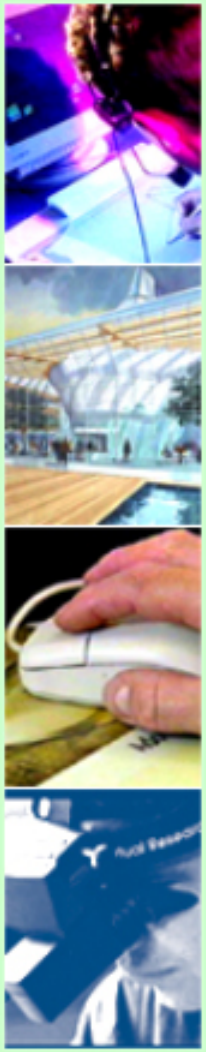


Complex types

- Defined using "*complexType*" element

typically contain :

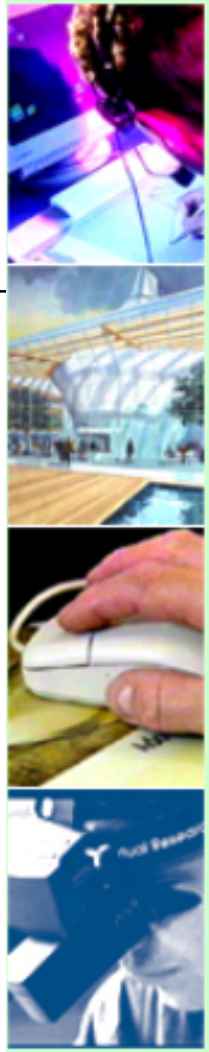
- element declarations .
- *attribute* declarations .
- *element* references .



Complex Types contain Nested Elements only

```
<person>
   <frstname>Mohamed</frstname>
   <lstname>Ahmed</lstname>
</person>
```

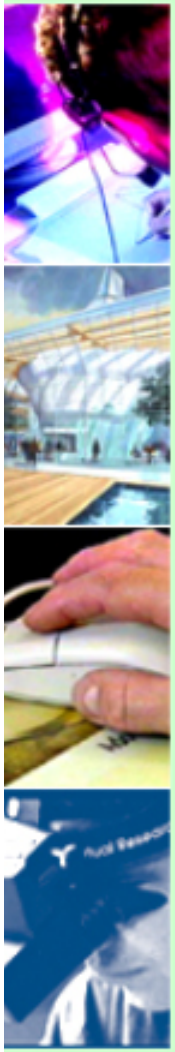
```
<xs:element name="person">
<xs:complexType>
   <xs:sequence>(xs:all)(xs:choice)
      <xs:element name="frstname"
                  type="xs:string"/>
      <xs:element name="lstname"
                  type="xs:string"/>
   </xs:sequence>
</xs:complexType>
</xs:element>
```



Complex Types contain text + attributes only

- `<son age="24"> ahmed </son>`

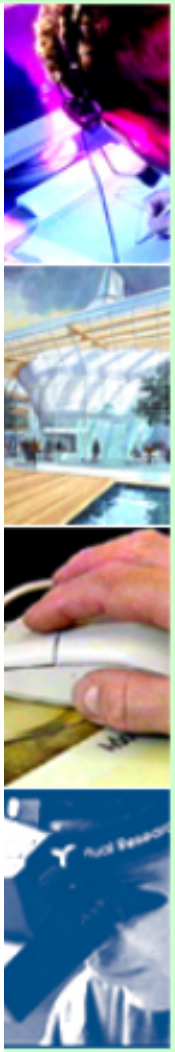
```
<xs:element name="son" >
  <xs:complexType>
  <xs:simpleContent>
  <xs:extension base="xs:string">
  <xs:attribute name="age"
                type="xs:integer"/>
 </xs:extension>
 </xs:simpleContent>
</xs:complexType>
</xs:element >
```



Complex Types (mixed content)

```
<salutation>DearMr
        <name>RobertSmith</name>
    </salutation>
```

```
<xs:element name="salutation">
  <xs:complexType mixed="true">
    <xs:sequence>(xs:all)
    <xs:element name="name"
                type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

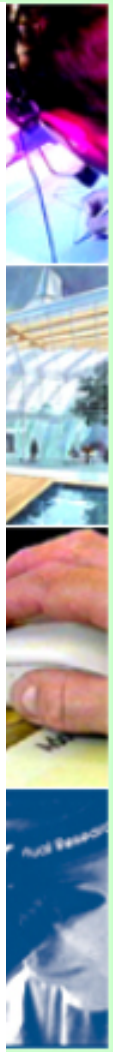


Complex Types contain Nested + attributes only

ISO 9001

- <father job="Engineer″>
  <mother>hoda</mother>
  <daughter>Kareema</daughter>
  <son>mohamed</son>
  </father>

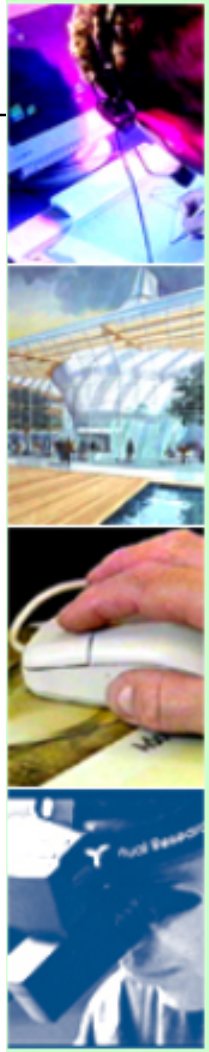
- <xs:element name="father">
  <xs:complexType>
  <xs:sequence>
  <xs:element name="mother" type="xs:string"/>
  <xs:element name="daughter" type="xs:string"/>
  <xs:element name="son" type="xs:string"/>
  </xs:sequence>
  <xs:attribute name="job" type="xs:string" default="Engineer"/>
  </xs:complexType>
  </xs:element>



Empty Content

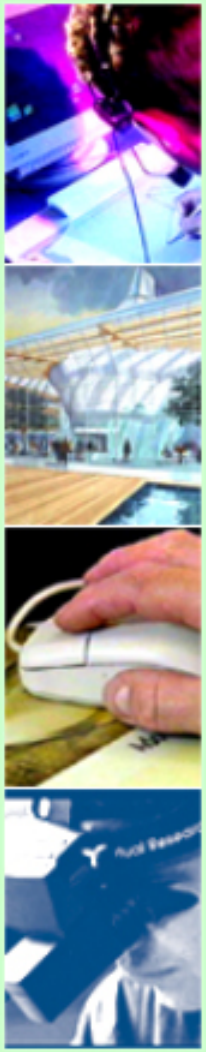
<internationalPrice currency="EUR"/>

```
<xs:element name="internationalPrice">
<xs:complexType>
<xs:complexContent>
<xs:restriction base="xs:anyType">
<xs:attribute name="currency“
        type="xs:string"/>
  </xs:restriction>
</xs:complexContent>
</xs:complexType>
</xs:element>
```



Occurrences of elements

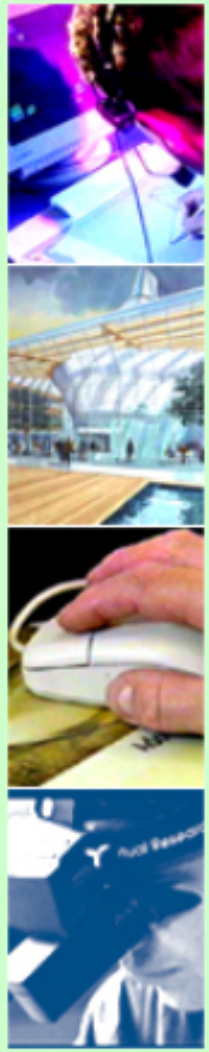
- *minOccurs*
- *maxOccurs*
- *fixed = "30":* the value must be 30
- *default* = "*20*",the value is set to what is specified



Occurrences of attributes

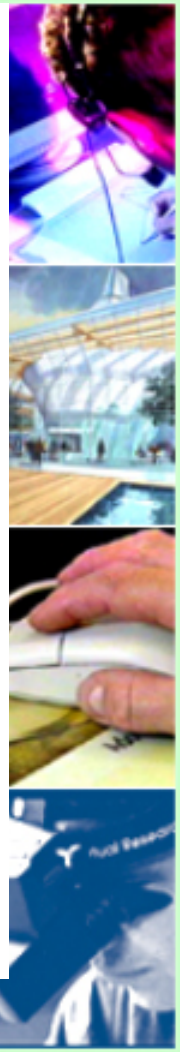
- Attributes can occur once or not at all
- "*use*" attribute
  - required
  - optional
- "default" attribute.
- "fixed " attribute .

```
<xs:attribute name="age" type="xs:integer"
        use="required" fixed="22"/>
```



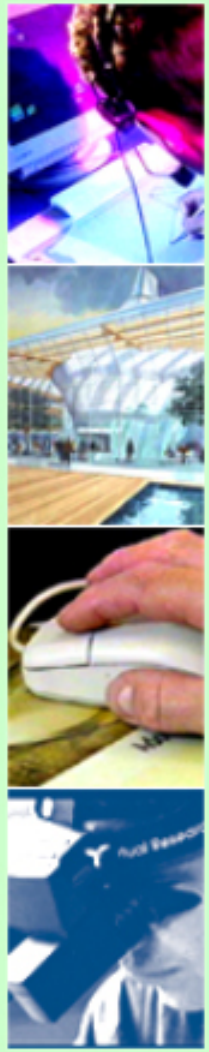
Elements Occurrences Examples

- <xs:element name="son" type="xs:string"
  minOccurs="1" maxOccurs="1"
  minOccurs="2" maxOccurs="unbounded"
  minOccurs="1" maxOccurs="1" fixed="Ali"
  minOccurs="0" maxOccurs="1" default="Ali"
>
<xs:element name="daughter" type="xs:string"
minOccurs="0"
maxOccurs="unbounded"/>



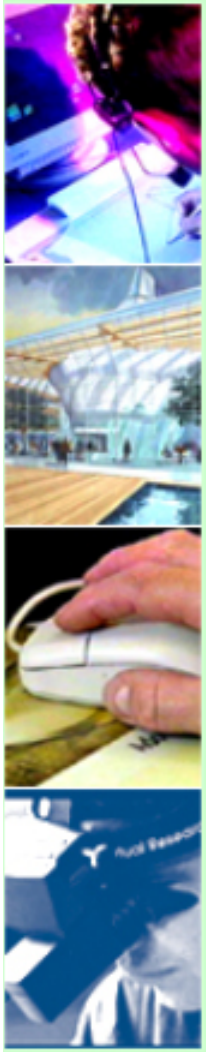
Derived Complex Types

- We can do a form of subclassing complexType definitions. We call this "derived types“:
  - Derive by extension: extend the parent complexType with more element.
  - Derive by restriction: create a type which is a subset of the base type. There are two ways to subset the elements



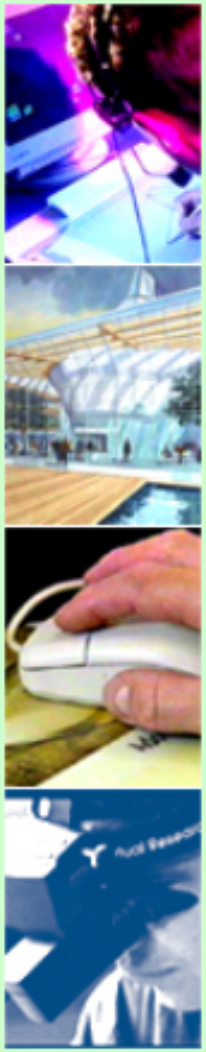
Derived by extension

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

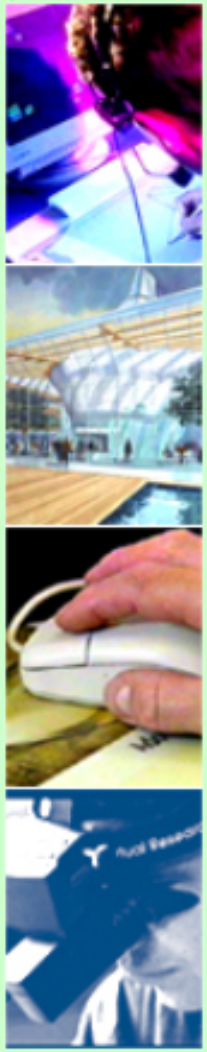


Derived by Restriction

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



Prohibiting Derivations



•Publication cannot be extended nor restricted:

```
<xs:complexType name="Publication"
                final="#all" >
```

•Publication cannot be restricted:

```
<xs:complexType name="Publication"
                final="restriction" >
```

•Publication cannot be extended:

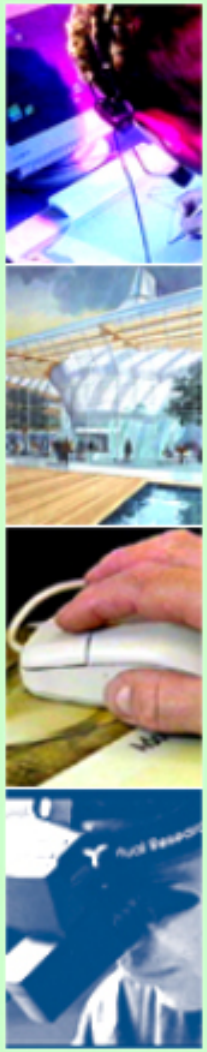
```
<xs:complexType name="Publication"
                final="extension" >
```

General examples

Example 1:

Consider the case where we need to Convert the following ***BookStore.dtd*** file to the XML Schema syntax:

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



```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           targetNamespace="http://www.books.org"
           elementFormDefault="qualified">

  <xs:element name="BookStore">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="Book" minOccurs="1" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:element name="Book">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="Title" minOccurs="1" maxOccurs="1"/>
        <xs:element ref="Author" minOccurs="1" maxOccurs="1"/>
        <xs:element ref="Date" minOccurs="1" maxOccurs="1"/>
        <xs:element ref="ISBN" minOccurs="1" maxOccurs="1"/>
        <xs:element ref="Publisher" minOccurs="1" maxOccurs="1"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:element name="Title" type="xs:string"/>
  <xs:element name="Author" type="xs:string"/>
  <xs:element name="Date" type="xs:string"/>
  <xs:element name="ISBN" type="xs:string"/>
  <xs:element name="Publisher" type="xs:string"/>
</xs:schema>
```

<!ELEMENT BookStore (Book)+>

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

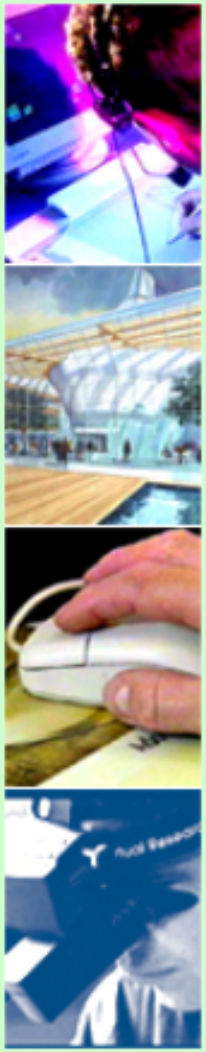
<!ELEMENT Title (#PCDATA)>

<!ELEMENT Author (#PCDATA)>

<!ELEMENT Date (#PCDATA)>

<!ELEMENT ISBN (#PCDATA)>

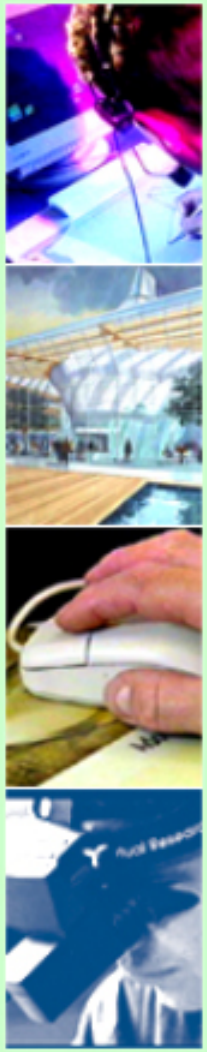
<!ELEMENT Publisher (#PCDATA)>



General examples

Example 2:

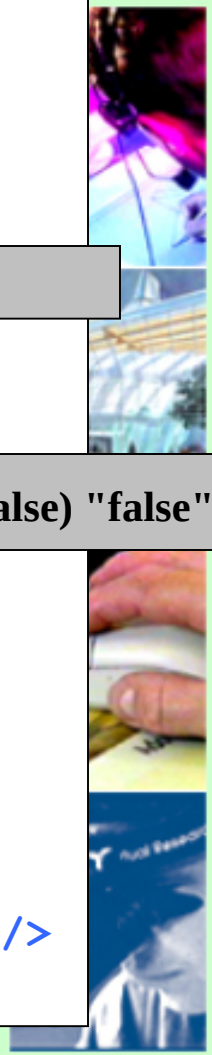
```
<!ELEMENT BookStore (Book+)>
<!ELEMENT Book (Title, Author)>
<!ATTLIST Book
Category (autobiography | fiction) #REQUIRED
        InStock (true | false) "false“    >
<!ELEMENT Title (#PCDATA)>
<!ELEMENT Author (#PCDATA)>
<!ELEMENT Date (#PCDATA)>
<!ELEMENT ISBN (#PCDATA)>
<!ELEMENT Publisher (#PCDATA)>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



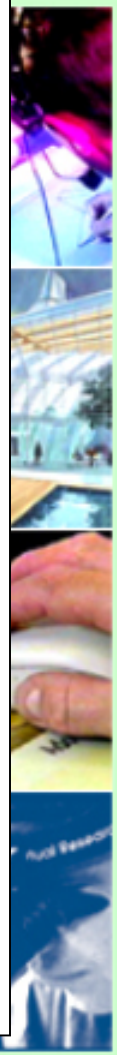
```
<xs:element name="BookStore">
 <xs:complexType>
   <xs:sequence>
   <xs:element name="Book"  maxOccurs="unbounded">
    <xs:complexType>
      <xs:sequence>
      <xs:element name="Title" type="xs:string"/>
      <xs:element name="Author" type="xs:string"/>
      </xs:sequence>
 <xs:attributeGroup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

Category (autobiography | fiction) #REQUIRED

InStock (true | false) "false"

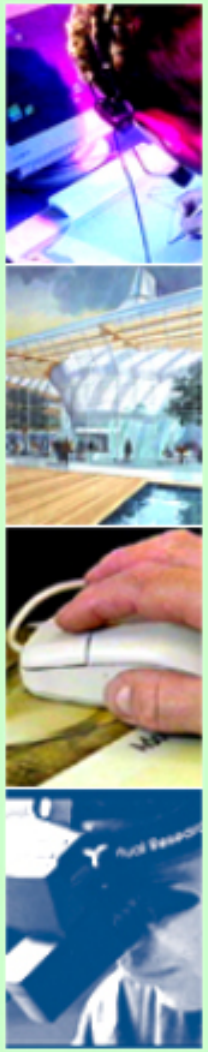


```
<xs:element name="Book"  maxOccurs="unbounded">
<xs:complexType>
  <xs:sequence>
  <xs:element name="Title" type="xs:string"/>
  <xs:element name="Author" type="xs:string"/>
  </xs:sequence>
<xs:attribute ref="Category" use="required"/>
<xs:attribute name="InStock" type="xs:boolean"
              default="false"/>
</xs:complexType>
</xs:element>
<xs:attribute name="Category">
    <xs:simpleType>
       <xs:restriction base="xs:string">
       <xs:enumeration value="autobiography"/>
       <xs:enumeration value="fiction"/>
       </xs:restriction>
     </xs:simpleType>
</xs:attribute>
```

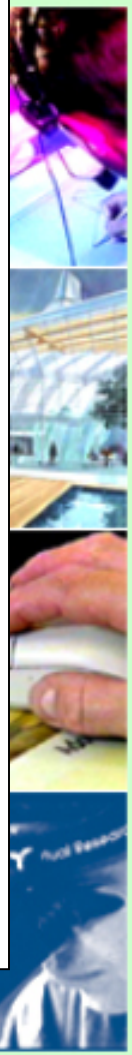


group Element

- The group element enables you to group together element declarations.
- Note: the group element is just for grouping together element declarations, no attribute declarations allowed!




```
<xs:element name="Book" >
    <xs:complexType>
        <xs:sequence>
      <xs:group ref="PublicationElements"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
<xs:group name="PublicationElements">
     <xs:sequence>
    <xs:element name="Title" type="xs:string"/>
    <xs:element name="Author" type="xs:string"
                    maxOccurs="unbounded"/>
    <xs:element name="Date" type="xs:string"/>
    </xs:sequence>
</xs:group>
```



XML doc.&XML schema

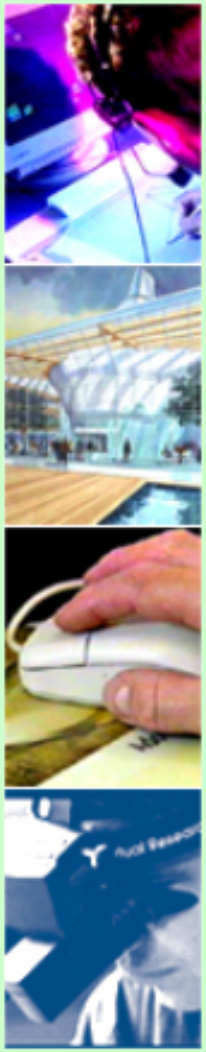
```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"----------(1)
           targetNamespace="http://www.books.org"----------(2)
           elementFormDefault="qualified">----------(3)
</xs:schema>
```

1)The elements and datatypes that are used to construct schemas: ***schema***, ***element***,***complexType***,***Sequence*** and ***string*** come from the http://…/XMLSchema namespace.

2) Says that the elements defined by this schema BookStore: ***Book***,***Title, Author,Date ,ISBN*** and ***Publisher*** are to go in this namespace.

3) This is to imply to any instance document which conforms to this schema: Any elements used by the instance document which were declared in this schema must be namespace qualified by the namespace specified by targetNamespace.

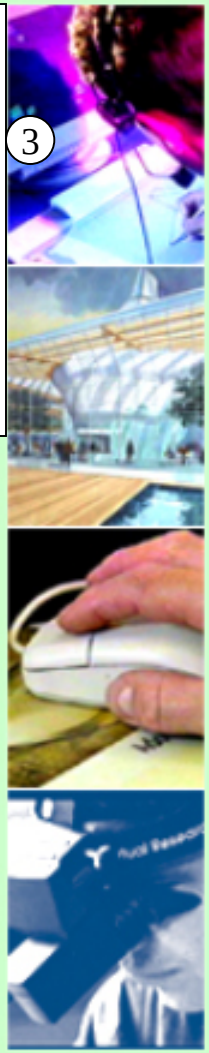
- **Note:** The “targetNamespace” attribute is removed when elements aren’t to go to any namespace.



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"
xsi:schemaLocation="http://www.books.org/BookStore.xsd">
     OR                                                     (2)
xsi:noNamespaceSchemaLocation="fname.xsd">
...
</BookStore>
```

(3)



1. First, using a default namespace declaration, tell the schema-validator that all of the elements used in this instance document come from the Book namespace.

2. Second, with schemaLocation tell the schema-validator that the *http://www.books.org* namespace is defined by BookStore.xsd

   In case Of no namespace associated,use"xsi:nonamespaceSchemaLocation"

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