XML Namespaces & Validation with XML Schema

Lecture 3

University of Technology, Mauritius
Service Oriented Architecture
WAT 5101C

G.Suddul

Outline

- Introduction
- Naming Conflicts
  - Namespace Definitions
- Schema
  - Definition and Purpose
  - Syntax Overview
  - Complex and Simple Type Definitions

Introduction

- We are going to cover XML Namespaces and XML Schema
  - XML Namespaces provide a method to avoid element name conflicts.
  - XML Schema is an XML-based alternative to DTDs.
  - The XML Schema language is also referred to as XML Schema Definition (XSD), and describes the structure of an XML document.
  - This lecture will show you how to read and create XML Schemas, why XML Schemas are more powerful than DTDs, and how to use the XML Schema language in your application.

3

Naming Conflicts

- The extensible markup language:
  - Since it is extensible, each developer can extend the language and create his/her own elements.
  - Naming conflicts may occur when trying to mix XML documents from different XML applications.
  - **Example:**
    - An element table that describes an HTML table and another element table that describes the furniture table will surely have the same name.
    - An element name that describes a book and an element name that describes and author in two different xml documents will also create a conflict.

4

Example: Name Elements

This XML carries the name of a book:

```
<Book>
    <Name> A book Name </Name>
    <Publisher> Wiley and Sons </Publisher>
    <Price> 600 </Price>
</Book>
```

This XML carries the name of an author:

```
<Author>
        <Name>
                <First> John </First>
                <Last> Doe </Last>
        </Name>
        <Address> xyz </Address>
        <Phone> 1234567 </Phone>
</Author>
```

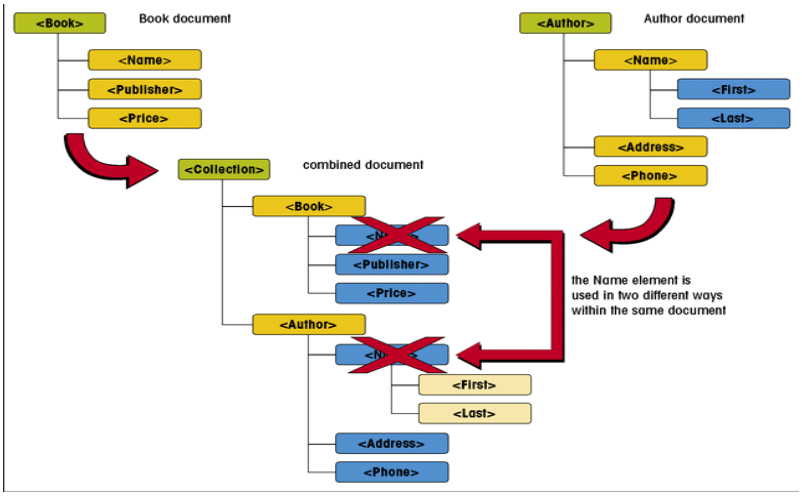
5

The Conflict

- If the previous XML fragments were added together, there would be a name conflict.
- Both contain a <name> element, but the elements have different content and meaning.
- An XML parser will not know how to handle these differences.
- We need a solution to this problem, since XML is open and is used by several organisations and developers.

6

Merge Book and Author Document



Solution to Name Conflict: Use a Prefix

- **Name conflicts in XML can easily be avoided using a name prefix.**

```
<Collection>
    <b:Book>
        <b:Name> A book Name </b:Name>
        <b:Publisher> Wiley and Sons </b:Publisher>
        <b:Price> 600 </b:Price>
    </b:Book>

    <a:Author>
            <a:Name>
                    <a:First> John </a:First>
                    <a:Last> Doe </a:Last>
            </a:Name>
            <a:Address> xyz </a:Address>
            <a:Phone> 1234567 </a:Phone>
    </a:Author>
</Collection>
```

XML Namespaces

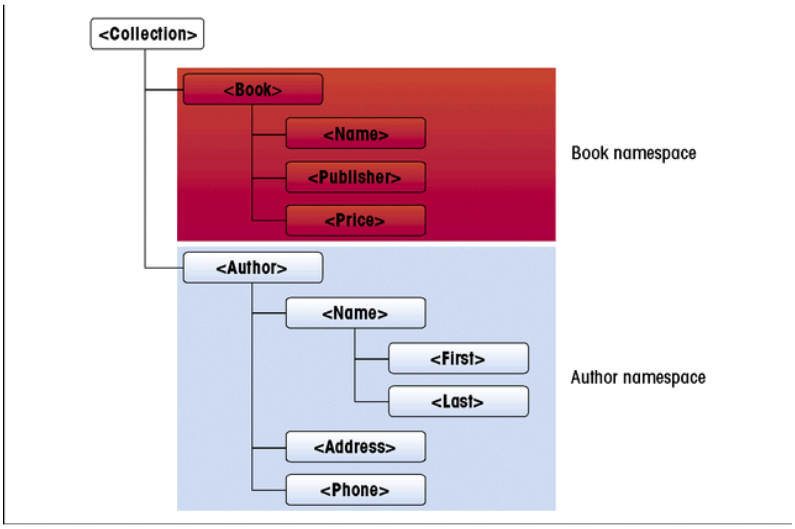
- When using prefixes in XML, a so-called **namespace** for the prefix must be defined.
- The namespace is defined by the **xmlns attribute** in the start tag of an element.
- The namespace declaration has the following syntax:
  - xmlns:*prefix*="*URI*".
- Example:
  - <**h:**table ***xmlns:h="http://www.utm.ac.mu/nmCollection/"***>

xmlns attribute in the <Name> tag

```
<Collection>
    <b:Book  xmlns:b="http://www.utm.ac.mu/book/">
        <b:Name> A book Name </b:Name>
        <b:Publisher> Wiley and Sons </b:Publisher>
        <b:Price> 600 </b:Price>
    </b:Book>

    <a:Author  xmlns:a="http://www.utm.ac.mu/author/">
          <a:Name>
                  <a:First> John </a:First>
                  <a:Last> Doe </a:Last>
           </a:Name>
          <a:Address> xyz </a:Address>
          <a:Phone> 1234567 </a:Phone>
    </a:Author>
</Collection>
```

Namespaces



11

Namespace

- In the previous examples, the xmlns attribute in the <table> and <Name> tags give the prefixes (h: and f:, b: and a:) a qualified namespace.
- When a namespace is defined for an element, all child elements with the same prefix are associated with the same namespace.
- Namespaces can be declared in the elements where they are used or in the XML root element.

12

Notes on Declaration of Namespaces

- A **namespace** is a defined collection of element and attribute names.
- Names that belong to the same namespace must be unique.
- Elements can share the same name if they reside in different namespaces.
- Namespaces must be declared before they can be used.

The Uniform Resource Identifier (URI)

- The **Uniform Resource Identifier** (URI) is a string of characters which identifies an Internet Resource.
- The most common URI is the **Uniform Resource Locator** (URL) which identifies an Internet domain address.
- Another, not so common type of URI is the **Universal Resource Name** (URN).

The Uniform Resource Identifier (URI)

- **NOTE**: The namespace URI is not used by the parser to look up information.
- The purpose is to give the namespace a unique name.
- However, often companies use the namespace as a pointer to a web page containing namespace information.

15

Default Namespaces

- Defining a default namespace for an element saves us from using prefixes in all the child elements.
- It has the following syntax:
  - xmlns="*namespaceURI*"
- Example:

```
<Author xmlns="http://www.utm.ac.mu/author/">
   <Name>
         <First>John</First>
        <Last>Doe</Last>
   </Name>
</Author>
```

16

Default Namespaces: Example

```
<Collection>
    <Book xmlns="http://www.utm.ac.mu/book/">
        <Name> A book Name <Name>
        <Publisher> Wiley and Sons <Publisher>
        <Price> 600 </Price>
    </Book>

    <Author xmlns="http://www.utm.ac.mu/author/">
          <Name>
                  <First> John </First>
                  <Last> Doe </Last>
           <Name>
          <Address> xyz </Address>
          <Phone> 1234567 </Phone>
    </Author>
</Collection>
```

17

XML Schema

- XML Schema is an XML-based alternative to DTD.
- It is an XML document that defines the content and structure of one or more XML documents.
- The XML Schema language is also referred to as XML Schema Definition (XSD).
- A schema is always placed in a separate XML document that is referenced by the instance document.

18

What is an XML Schema?

- The purpose of an XML Schema is to define the legal building blocks of an XML document (like a DTD).
- An XML Schema:
  - defines elements that can appear in a document .
  - defines attributes that can appear in a document .
  - defines which elements are child elements.
  - defines the order of child elements.
  - defines the number of child elements.
  - defines whether an element is empty or can include text.
  - defines data types for elements and attributes.
  - defines default and fixed values for elements and attributes.

19

Successors for DTDs

- XML Schemas is used in most Web applications as a replacement for DTDs. Here are some reasons:
  - XML Schemas are extensible to future additions.
  - XML Schemas are richer and more powerful than DTDs.
  - XML Schemas are written in XML.
  - XML Schemas support data types.
  - XML Schemas support namespaces.

20

A Simple XML file (contact.xml)

```
<?xml version="1.0"?>
   <contact>
        <firstname>Geerish</firstname>
        <lastname>Suddul</lastname>
        <email>g.suddul@utm.intnet.mu</email>
        <phone> 2075250</phone>
   </contact>
```

21

Below a DTD (contact.dtd)

- DTD for the contact xml document:

```
<!ELEMENT contact (firstname, lastname, email, phone)>
<!ELEMENT firstname (#PCDATA)>
<!ELEMENT lastname (#PCDATA)>
<!ELEMENT email (#PCDATA)>
<!ELEMENT phone (#PCDATA)>
```

22

An XML Schema (contactXSD.xsd)

```
<?xml version="1.0" encoding="UTF-8"?>

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   targetNamespace="http://xml.netbeans.org/schema/contactXSD"
   xmlns:tns="http://xml.netbeans.org/schema/contactXSD"
   elementFormDefault="qualified">

   <xsd:element name="contact">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element name="firstname"   type="xsd:string"/>
            <xsd:element name="lastname"   type="xsd:string"/>
            <xsd:element name="email"   type="xsd:string"/>
            <xsd:element name="phone"   type="xsd:integer"/>
         </xsd:sequence>
      </xsd:complexType>
   </xsd:element>
</xsd:schema>
```

23

A Brief Syntax Overview

- In the previous DTD:
  - The first line defines the contact element to have four child elements: “firstname, lastname, email, phone".
  - Other lines defines firstname, lastname, email, phone elements to be of type "#PCDATA".
- In the actual XSD:
  - The ***contact*** element is a **complex type** because it contains other elements.
  - The other elements (***firstname, lastname, email, phone***) are **simple types** because they do not contain other elements.

24

XSD Root Element, <Schema>

- The **<schema>** element is the root element of every XML Schema

```
<?xml version="1.0"?>
  <xsd:schema>
    ... ...
  </xsd:schema>
```

- The **<schema>** element may contain some attributes.

Attributes of Schema

```
<?xml version="1.0"?>
<xsd:schema
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://xml.netbeans.org/schema/contactXSD"
  xmlns:tns="http://xml.netbeans.org/schema/contactXSD"
  elementFormDefault="qualified">
  ... ...

</xsd:schema>
```

Overview of the Attributes

- **xmlns:xsd="http://www.w3.org/2001/XMLSchema"**
  - Indicates that the elements and data types used in the schema come from the "http://www.w3.org/2001/XMLSchema" namespace.
  - All the elements are prefixed by xsd.
- **targetNamespace="**http://xml.netbeans.org/schema/contactXSD**"**
  - Indicates that elements defined by this a schema (e.g. firstname, lastname, email, phone) come from the "http://xml.netbeans.org/schema/contactXSD" namespace.

27

Overview of the Attributes

- **xmlns="**http://xml.netbeans.org/schema/contactXSD**"**
  - Indicates the default namespace to be at http://xml.netbeans.org/schema/contactXSD
- **elementFormDefault="qualified"**
  - Indicates that any elements used by the XML instance document which were declared in this schema must be namespace qualified.

28

Referencing a DTD

```
<?xml version="1.0"?>
<!DOCTYPE contact SYSTEM “contact.dtd”>
<contact>
    <firstname>Geerish</firstname>
     <lastname>Suddul</lastname>
    <email>g.suddul@utm.intnet.mu</email>
    <phone>2075250</phone>
</contact>
```

29

Referencing a Schema

```
<?xml version="1.0" encoding="UTF-8"?>

<contact xmlns=http://xml.netbeans.org/schema/contactXSD
xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
xsi:schemaLocation="http://xml.netbeans.org/schema/contactXSD
 http://localhost/tests/contactXSD.xsd” >

    <firstname>Geerish</firstname>
    <lastname>Suddul</lastname>
    <email>g.suddul@utm.intnet.mu</email>
    <phone> 2075250</phone>
</contact>
```

30

Overview of the Attributes

- **xmlns=http://xml.netbeans.org/schema/contactXSD**
  - specifies the default namespace declaration
- The XML Schema Instance namespace has a SchemaLocation attribute which takes two values.
  - **xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance**
    - The xml schema instance namespace (xsi)
- **xsi:schemaLocation="http://xml.netbeans.org/schema/contactXSD http:///localhost/tests/contactXSD.xsd"**
  - First Value : namespace to use
  - Second Value : location of the xml schema to use

Defining a Simple Element

- A simple element is an XML element which corresponds to a simple data type.
  - It can contain only text, and cannot contain any other elements or attributes.
- Syntax:
  - **<xsd:element name="xxx" type="yyy"/>**
  - Name is represented by xxx
  - Type or Data type is represented by yyy

Data Types

- XML Schema has a lot of built-in data types.
- The most common types are:
  - xsd:string
  - xsd:decimal
  - xsd:integer
  - xsd:boolean
  - xsd:date
  - xsd:time

33

Examples

<lastname>Bond</lastname>
<age>30</age>
<date_of_birth>1981-25-05</date_of_birth>

- **Definitions of the above elements**:

<xsd:element name="lastname" type="xsd:string"/>

<xsd:element name="age" type="xsd:integer"/>

<xsd:element name="date_of_birth" type="xsd:date"/>

34

Default or Fixed Values

- A **default value** is automatically assigned to the element when no other value is specified.
- <xsd:element name="company" type="xsd:string" **default=**“UTM”/>
- A **fixed value** is also automatically assigned to the element, and you cannot specify another value.
- <xsd:element name=" company " type="xsd:string" **fixed=**“UTM”/>

35

Defining Attributes

- Simple elements cannot have attributes (by nature of their definitions). If they have attributes, then they are considered as complex elements.
- But attributes themselves are always declared as simple elements.

1. <xsd:attribute name="xxx" type="yyy"/>
2. <xsd:attribute name="lang" type="xsd:string" default="EN"/>
3. <xsd:attribute name="lang" type="xsd:string" fixed="EN"/>
4. <xsd:attribute name="lang" type="xsd:string" use="required"/>

- **NOTE:** attributes are optional by default, the “use” attribute like in example 4, makes it a required attribute.

36

XSD Restrictions (Facets)

- Restrictions are used to define acceptable values for XML elements or attributes.

- Example: Restriction on age (lets add it to contactXSD.xsd)

```
<xsd:element name="age">
    <xsd:simpleType>
        <xsd:restriction base="xsd:integer">
            <xsd:minInclusive value=“18"/>
            <xsd:maxInclusive value=“65"/>
        </xsd:restriction>
    </xsd:simpleType>
</xsd:element>
```

37

XSD Restrictions on Series of Values

- The **pattern** constraint
  - It helps to define a series of numbers or letters that can be used for an element content. (remember regular expressions)

- **<xsd:pattern value="[a-z]"/>**
  - Only acceptable value is ONE of the LOWERCASE letters from a to z.
- **<xsd:pattern value="[A-Z][A-Z][A-Z]"/>**
  - Only acceptable value is THREE of the UPPERCASE letters from a to z.
- **<xsd:pattern value="[a-zA-Z][a-zA-Z][a-zA-Z]"/>**
  - Only acceptable value is THREE of the LOWERCASE OR UPPERCASE letters from a to z
- **<xsd:pattern value="([a-z])*"/>**
  - Acceptable value is zero or more occurrences of lowercase letters from a to z.
- **<xsd:pattern value="[0-9][0-9][0-9][0-9][0-9]"/>**
  - only acceptable value is FIVE digits in a sequence, and each digit must be in a range from 0 to 9. (applies to xsd:integer)

38

XSD Restrictions on Series of Values

```
<xsd:element name="firstname">
   <xsd:simpleType>
        <xsd:restriction base="xsd:string">
           <xs:pattern value="([a-zA-Z])+"/>
        </xsd:restriction>
   </xsd:simpleType>
</xsd:element>
```

- One or more occurrences of lower/upper-case letter.
- We add same restriction on lastname in contact.xsd.

39

Restriction on Length

```
<xs:element name="password">
     <xs:simpleType>
        <xs:restriction base="xs:string">
                <xs:minLength value="5"/>
                <xs:maxLength value="8"/>
        </xs:restriction>
     </xs:simpleType>
</xs:element>
```

- The example defines an element called "password" with a restriction: The value must be minimum five characters and maximum eight characters.

40

Restrictions on DataTypes

Constraint	Description
enumeration	Defines a list of acceptable values
fractionDigits	Specifies the maximum number of decimal places allowed. Must be equal to or greater than zero
length	Specifies the exact number of characters or list items allowed. Must be equal to or greater than zero
maxExclusive	Specifies the upper bounds for numeric values (the value must be less than this value)
maxInclusive	Specifies the upper bounds for numeric values (the value must be less than or equal to this value)
maxLength	Specifies the maximum number of characters or list items allowed. Must be equal to or greater than zero
minExclusive	Specifies the lower bounds for numeric values (the value must be greater than this value)
minInclusive	Specifies the lower bounds for numeric values (the value must be greater than or equal to this value)
minLength	Specifies the minimum number of characters or list items allowed. Must be equal to or greater than zero
pattern	Defines the exact sequence of characters that are acceptable
totalDigits	Specifies the exact number of digits allowed. Must be greater than zero
whiteSpace	Specifies how white space (line feeds, tabs, spaces, and carriage returns) is handled

41

Complex Elements

- A complex element is an XML element that contains other elements and/or attributes.
- There are four kinds of complex elements:
  - empty elements.
  - elements that contain only other elements.
  - elements that contain only text.
  - elements that contain both other elements and text.
- **Note:** Each of these elements may contain attributes as well.

42

Examples of Complex Elements

- complex XML element, “order", which is empty:
  <order oid=“9876”/>

- A complex XML element, “author", which contains only other elements:

  <author>
  <firstname>John</firstname>
  <lastname>Doe</lastname>
  </author>

Examples of Complex Elements

- A complex XML element, "food", which contains only text:
  <food type="dessert">Ice cream</food>

- A complex XML element, "description", which contains both elements and text:
  <description>
  This dish serves <serving>4</serving> …
  </description>

Defining a Complex Element

```
<xsd:element name="contact">
   <xsd:complexType>
        <xsd:sequence>
          <xsd:element name="firstname" type="xsd:string"/>
          <xsd:element name="lastname" type="xsd:string"/>
          <xsd:element name="email" type="xsd:string"/>
          <xsd:element name="phone" type="xsd:integer"/>
          <xsd:element name="company" type="xsd:string"/>
          <xsd:element name="age" type="xsd:integer"/>
        </xsd:sequence>
   </xsd:complexType>
</xsd:element>
```


Defining a Complex Element

- The method used previously specifies the "contact" element as a complex type.
- All the child elements (firstname, lastname, email, phone, company, age) are surrounded by the <sequence> indicator.
  - This means that the child elements must appear in the same order as they are declared.

Defining Empty Elements

- <contact_number id="1975" />
- The “order” element above has no content at all.

```
<xsd:element name=“contact_number”>
   <xsd:complexType>
        <xsd:attribute name=“id” type="xsd:positiveInteger"/>
   </xsd:complexType>
</xsd:element>
```

47

Defining Element-Only

- An "elements-only" complex type contains an element that contains only other elements.

```
<xsd:element name=“contact”>
   <xsd:complexType>
        <xsd:sequence>
          <xsd:element name=“firstname” type=“xsd:string”/>
           <xsd:element name=“lastname” type=“xsd:string”/>
           <xsd:element name=“email” type=“xsd:string”/>
           <xsd:element name=“phone” type=“xsd:integer”/>
        </xsd:sequence>
   </xsd:complexType>
</xsd:element>
```

48

Defining Complex Text-Only

- This type contains only simple content (text and attributes), therefore we wrap the content around **simpleContent** elements.
- <email email_type="work">g.suddul@utm.intnet.mu</email>
- When using the simple content, an **extension** OR a **restriction** must be defined within the **simpleContent** element.
  - extension/restriction element expands or limits the base simple type for the element.

49

Defining Complex Text-Only

```
<xsd:element name="email">
    <xsd:complexType>
        <xsd:simpleContent>
            <xsd:extension base="xsd:string">
                <xsd:attribute name="email_type" type="xsd:string" />
            </xsd:extension>
        </xsd:simpleContent>
    </xsd:complexType>
</xsd:element>
```

50

Complex Types With Mixed Content

- A mixed complex type element can contain attributes, elements, and text.

<letter>
Dear Mr.<name>John Smith</name>. Your order <orderid>1032</orderid> will be shipped on <shipdate>2001-07-13</shipdate>.
</letter>

Schema for letter element

```
<xsd:element name="letter">
  <xsd:complexType mixed="true">
    <xsd:sequence>
      <xsd:element name="name" type="xsd:string"/>
      <xsd:element name="orderid" type="xsd:positiveInteger"/>
      <xsd:element name="shipdate" type="xsd:date"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

- To enable character data to appear between the child-elements of "letter", the mixed attribute must be set to "true".

Complex Type Indicators

- We can control how elements are to be used in XML documents with indicators.
- There are seven indicators:
  - **Order indicators:**
    - All
    - Choice
    - Sequence
  - **Occurrence indicators:**
    - maxOccurs
    - minOccurs
  - **Group indicators:**
    - Group name
    - attributeGroup name

53

Order Indicators

```
<xsd:element name="contact">
    <xsd:complexType>
        <xsd:all>
          <xsd:element name="firstname"  type="xsd:string"/>
          <xsd:element name="lastname"  type="xsd:string"/>
        </xsd:all>
    </xsd:complexType>
</xsd:element>
```

- **All**: child elements can appear in any order, and that each child element must occur only once.
- **Choice**: specifies that either one child element or another can occur
- **Sequence**: indicator specifies that the child elements must appear in a specific order

54

Occurrence Indicators

```
<xsd:element name="contact">
    <xsd:complexType>
        <xsd:sequence>
          <xsd:element name="firstname"  type="xsd:string"/>
          <xsd:element name="lastname"  type="xsd:string"/>
          <xsd:element name="email" type="xsd:string" maxoccurs="5"
            minoccurs="0">
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
```

- **maxOccurs**: maximum number of times an element can occur
- **minOccurs**: minimum number of times an element can occur

55

Occurrence Indicators

- The previous example indicates that the "email" element can occur a minimum of zero times and a maximum of five times in the "contact" element.
- **Tip:**
  - To allow an element to appear an unlimited number of times, use the maxOccurs="unbounded" statement.
- **Note:**
  - For all "Order" and "Group" indicators (any, all, choice, sequence, group name, and group reference) the default value for maxOccurs and minOccurs is 1.

56

Example: family.xml

```
<?xml version="1.0"?>
<persons xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            xsi:noNamespaceSchemaLocation="family.xsd">

    <person>
        <full_name>David Gilmour</full_name>
        <child_name>Cecilie</child_name>
    </person>
    <person>
        <full_name>Steve Barret</full_name>
         <child_name>Hege</child_name>
         <child_name>Stale</child_name>
         <child_name>Jim</child_name>
        <child_name>Borge</child_name>
    </person>
    <person>
        <full_name>Roger Waters</full_name>
     </person>
</persons>
```

57

Family.xsd

```
<?xml version="1.0“ ?>
<xsd:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    elementFormDefault="qualified">
    <xsd:element name=“persons”>
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name=“person” maxOccurs=“unbounded”>
                    <xsd:complexType>
                        <xsd:sequence>
                            <xsd:element name=“full_name” type=“xsd:string”/>
                            <xsd:element name=“child_name” type=“xsd:string”
                                                minOccurs=“0” maxOccurs=“5”/>
                        </xsd:sequence>
                    </xsd:complexType>
                </xsd:element>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
</xsd:schema>
```

58

References/Further Reading

- The w3c tutorials on XSD
  - www.w3schools.com
- Learn XML Tips
  - George M.Doss
  - Wordware Publising, 2000.
- Dictionary of XML Technologies and the Semantic Web
  - By Vladimir Geroimenko, 2004