XML Schema

- An XML Schema describes the structure of an XML document.
- The XML Schema language is also referred to as XML Schema Definition (XSD).
- XML Schema is an XML-based (and more powerful) alternative to DTD.

The purpose of an XML Schema is to define the legal building blocks of an XML document:

- the elements and attributes that can appear in a document
- the number of (and order of) child elements
- data types for elements and attributes
- default and fixed values for elements and attributes

XML Schemas Support Data Types

One of the greatest strength of XML Schemas is the support for data types.

- It is easier to describe allowable document content
- It is easier to validate the correctness of data
- It is easier to define data facets (restrictions on data)
- It is easier to define data patterns (data formats)
- It is easier to convert data between different data types

XML Schemas are extensible, because they are written in XML.

With an extensible Schema definition you can:

- Reuse your Schema in other Schemas
- Create your own data types derived from the standard types
- Reference multiple schemas in the same document

Two major schema models exist: W3C XML Schema and Microsoft XML Schema.

- An XML document that conforms to a schema document is schema valid; a document that does not conform is *invalid*.
- XML schemas are an alternative for validating XML documents. DTDs and schemas need validating parsers. Schemas are an emerging technology that is expected to eventually replace DTDs as the primary means of describing XML document structure.

- Microsoft XML Schema documents commonly use the **.xml** extension and W3C XML Schema documents commonly use the***.xsd*** extension.
- W3C XML Schema use the URI **http://www.w3.org/2000/10/XMLSchema** and namespace prefix **xsd**. Root element **schema** contains the document definitions.

XSD Example

```
<?xml version="1.0"?>
<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="heading" type="xs:string"/>
      <xs:element name="body" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

</xs:schema>
```

XML Schemas Secure Data Communication

- When sending data from a sender to a receiver, it is essential that both parts have the same "expectations" about the content.
- With XML Schemas, the sender can describe the data in a way that the receiver will understand.
- A date like: "03-11-2004" will, in some countries, be interpreted as 3.November and in other countries as 11.March.
- However, an XML element with a data type like this:
- <date type="date">2004-03-11</date>
- ensures a mutual understanding of the content, because the XML data type "date" requires the format "YYYY-MM-DD".

Well-Formed is Not Enough

A well-formed XML document is a document that conforms to the XML syntax rules, like:

- it must begin with the XML declaration
- it must have one unique root element
- start-tags must have matching end-tags
- elements are case sensitive
- all elements must be closed
- all elements must be properly nested
- all attribute values must be quoted
- entities must be used for special characters

Even if documents are well-formed they can still contain errors, and those errors can have serious consequences.

Think of the following situation: you order 5 gross of laser printers, instead of 5 laser printers. With XML Schemas, most of these errors can be caught by your validating software.

XSD Simple Elements

- simple element is an XML element that can contain only text. It cannot contain any other elements or attributes.
- the "only text" restriction is quite misleading. The text can be of many different types. It can be one of the types included in the XML Schema definition (boolean, string, date, etc.), or it can be a custom type that you can define yourself.

Defining a Simple Element

The syntax for defining a simple element is:

```
<xs:element name="xxx" type="yyy"/>
```

where xxx is the name of the element and yyy is the data type of the element.

XML Schema has a lot of built-in data types. The most common types are:

- xs:string
- xs:decimal
- xs:integer
- xs:boolean
- xs:date
- xs:time

The corresponding simple element definitions:

Example

- Here are some XML elements:
- <lastname>Refsnes</lastname>
  <age>36</age>
  <dateborn>1970-03-27</dateborn>

Xsd content:

<xs:element name="lastname" type="xs:string"/>
<xs:element name="age" type="xs:integer"/>
<xs:element name="dateborn" type="xs:date"/>

Default and Fixed Values for Simple Elements

Simple elements may have a default value OR a fixed value specified.

A default value is automatically assigned to the element when no other value is specified.

In the following example the default value is "red":

<xs:element name="color" type="xs:string" default="red"/>

A fixed value is also automatically assigned to the element, and you cannot specify another value.

In the following example the fixed value is "red":

- <xs:element name="color" type="xs:string" fixed="red"/>

XSD Attributes

- All attributes are declared as simple types
- <xs:attribute name="xxx" type="yyy"/>
- XML Schema has a lot of built-in data types. The most common types are:
- xs:string
- xs:decimal
- xs:integer
- xs:boolean
- xs:date
- xs:time

Example

- Here is an XML element with an attribute:
- <lastname lang="EN">Smith</lastname>
- And here is the corresponding attribute definition:
- <xs:attribute name="lang" type="xs:string"/>

Default and Fixed Values for Attributes

- Attributes may have a default value OR a fixed value specified.
- A default value is automatically assigned to the attribute when no other value is specified.
- In the following example the default value is "EN":
- <xs:attribute name="lang" type="xs:string" default="EN"/>
- A fixed value is also automatically assigned to the attribute, and you cannot specify another value.
- Ex: <xs:attribute name="lang" type="xs:string" fixed="EN"/>

Optional and Required Attributes

- Attributes are optional by default. To specify that the attribute is required, use the "use" attribute:
- <xs:attribute name="lang" type="xs:string" use="required"/>

XSD Restrictions/Facets

- Restrictions are used to define acceptable values for XML elements or attributes. Restrictions on XML elements are called facets.
- **Restrictions on Values**
- The following example defines an element called "age" with a restriction. The value of age cannot be lower than 0 or greater than 120:

```
<xs:element name="age">
  <xs:simpleType>
    <xs:restriction base="xs:integer">
      <xs:minInclusive value="0"/>
      <xs:maxInclusive value="12"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

Restrictions on a Set of Values

- To limit the content of an XML element to a set of acceptable values, we would use the enumeration constraint.
- The example below defines an element called "car" with a restriction. The only acceptable values are: Audi, Golf, BMW:

```
<xs:element name="car">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:enumeration value="Audi"/>
      <xs:enumeration value="Golf"/>
      <xs:enumeration value="BMW"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

The example above could also have been written like this:

```
<xs:element name="car" type="carType"/>

<xs:simpleType name="carType">
  <xs:restriction base="xs:string">
    <xs:enumeration value="Audi"/>
    <xs:enumeration value="Golf"/>
    <xs:enumeration value="BMW"/>
  </xs:restriction>
</xs:simpleType>
```

Note: In this case the type "carType" can be used by other elements because it is not a part of the "car" element.

Restrictions on a Series of Values

- To limit the content of an XML element to define a series of numbers or letters that can be used, we would use the pattern constraint.
- The example below defines an element called "letter" with a restriction. The only acceptable value is ONE of the LOWERCASE letters from a to z:
-

```
<xs:element name="letter">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:pattern value="[a-z]"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

The next example defines an element called "initials" with a restriction. The only acceptable value is THREE of the UPPERCASE letters from a to z:

- <xs:element name="initials">
  <xs:simpleType>
  <xs:restriction base="xs:string">
  <xs:pattern value="[A-Z][A-Z][A-Z]"/>
  </xs:restriction>
  </xs:simpleType>
  </xs:element>

- The next example also defines an element called "initials" with a restriction. The only acceptable value is THREE of the LOWERCASE OR UPPERCASE letters from a to z:
- <xs:element name="initials">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:pattern value="[a-zA-Z][a-zA-Z][a-zA-Z]"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>

The next example defines an element called "choice" with a restriction. The only acceptable value is ONE of the following letters: x, y, OR z:

- <xs:element name="choice">
  <xs:simpleType>
  <xs:restriction base="xs:string">
  <xs:pattern value="[xyz]"/>
  </xs:restriction>
  </xs:simpleType>
  </xs:element>

The next example defines an element called "prodid" with a restriction. The only acceptable value is FIVE digits in a sequence, and each digit must be in a range from 0 to 9:

- <xs:element name="prodid">
  <xs:simpleType>
  <xs:restriction base="xs:integer">
  <xs:pattern value="[0-9][0-9][0-9][0-9][0-9]"/>
  </xs:restriction>
  </xs:simpleType>
  </xs:element>

- **Other Restrictions on a Series of Values**
- The example below defines an element called "letter" with a restriction. The acceptable value is zero or more occurrences of lowercase letters from a to z:
- <xs:element name="letter">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:pattern value="([a-z])*"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>

- The next example also defines an element called "letter" with a restriction. The acceptable value is one or more pairs of letters, each pair consisting of a lower case letter followed by an upper case letter. For example, "sToP" will be validated by this pattern, but not "Stop" or "STOP" or "stop":
- <xs:element name="letter">
  <xs:simpleType>
  <xs:restriction base="xs:string">
  <xs:pattern value="([a-z][A-Z])+"/>
  </xs:restriction>
  </xs:simpleType>
  </xs:element>

- The next example defines an element called "gender" with a restriction. The only acceptable value is male OR female:
- <xs:element name="gender">
  <xs:simpleType>
  <xs:restriction base="xs:string">
  <xs:pattern value="male|female"/>
  </xs:restriction>
  </xs:simpleType>
  </xs:element>

- The next example defines an element called "password" with a restriction. There must be exactly eight characters in a row and those characters must be lowercase or uppercase letters from a to z, or a number from 0 to 9:
- <xs:element name="password">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:pattern value="[a-zA-Z0-9]{8}"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>

Restrictions on Whitespace Characters

- To specify how whitespace characters should be handled, we would use the whiteSpace constraint.
- This example defines an element called "address" with a restriction. The whiteSpace constraint is set to "preserve", which means that the XML processor WILL NOT remove any white space characters:
- <xs:element name="address">
  <xs:simpleType>
  <xs:restriction base="xs:string">
  <xs:whiteSpace value="preserve"/>
  </xs:restriction>
  </xs:simpleType>
  </xs:element>

This example also defines an element called "address" with a restriction. The whiteSpace constraint is set to "replace", which means that the XML processor WILL **REPLACE** all white space characters (line feeds, tabs, spaces, and carriage returns) with spaces:

- <xs:element name="address">
  <xs:simpleType>
  <xs:restriction base="xs:string">
  <xs:whiteSpace value="replace"/>
  </xs:restriction>
  </xs:simpleType>
  </xs:element>

This example also defines an element called "address" with a restriction. The whiteSpace constraint is set to "**collapse**", which means that the XML processor **WILL REMOVE** all white space characters (line feeds, tabs, spaces, carriage returns are replaced with spaces, leading and trailing spaces are removed, and multiple spaces are reduced to a single space):

- <xs:element name="address">
  <xs:simpleType>
  <xs:restriction base="xs:string">
  <xs:whiteSpace value="collapse"/>
  </xs:restriction>
  </xs:simpleType>
  </xs:element>

preserve
No normalization is done, the value is the ·normalized value·
replace
All occurrences of #x9 (tab), #xA (line feed) and #xD (carriage return) are replaced with #x20 (space).
collapse
Subsequent to the replacements specified above under replace, contiguous sequences of #x20s are collapsed to a single #x20, and initial and/or final #x20s are deleted.

- **Restrictions on Length**
- To limit the length of a value in an element, we would use the length, maxLength, and minLength constraints.
- This example defines an element called "password" with a restriction. The value must be exactly eight characters:
- <xs:element name="password">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:length value="8"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>

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

Attribute - Restriction

Restriction	Description
xs:enumeration	Defines a list of values for an element
xs:length	Defines the exact number of characters or list elements that are allowed. The value of this length must equal to or greater than zero.
maxExclusive	Defines the upper limit for numeric values
maxInclusive	defines the upper limit for numeric values
maxLength	Defines the maximum number of characters or list items that is allowed. Must be equal to or greater than zero
minExclusive	Defines the lower limit for numeric values
minInclusive	defines the lower limit for numeric values
minLength	Defines 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
whiteSpace	Defines how white space (line feeds, tabs, spaces, and carriage returns) is handled
totalDigits	Defines the exact number of digits allowed. Must be greater than zero

XML Schema file called "note.xsd"

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.w3schools.com"
xmlns="http://www.w3schools.com"
elementFormDefault="qualified">

<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="heading" type="xs:string"/>
      <xs:element name="body" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

</xs:schema>
```

The note element is a **complex type** because it contains other elements. The other elements (to, from, heading, body) are **simple types** because they do not contain other elements.

Restrictions for 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

Figure 7.20 is a Microsoft XML Schema for the day planner introduced.

```
<?xml version = "1.0"?>
<!-- Fig. 7.20 : planner-schema.xml --> <!-- Microsoft XML Schema for day planner -->
<Schema xmlns = "urn:schemas-microsoft-com:xml-data“ xmlns:dt = "urn:schemas-microsoft-com:datatypes">
<ElementType name = "planner" content = "eltOnly“ model = "closed">
<element type = "year" minOccurs = "0" maxOccurs = "*"/>
</ElementType>
<ElementType name = "year" content = "eltOnly" model = "closed">
<AttributeType name = "value" dt:type = "int"/>
<attribute type = "value"/>
<element type = "date" minOccurs = "0" maxOccurs = "*"/>
</ElementType>
<ElementType name = "date" content = "eltOnly" model = "closed">
<AttributeType name = "month" dt:type = "int"/>
<attribute type = "month"/>
<AttributeType name = "day" dt:type = "int"/>
<attribute type = "day"/>
<element type = "note" minOccurs = "0" maxOccurs = "*"/>
</ElementType>
<ElementType name = "note" content = "textOnly" model = "closed“ dt:type = "string">
<AttributeType name = "time" dt:type = "int"/>
<attribute type = "time"/>
</ElementType>
```

- **Internet and World Wide Web Resources**

msdn.microsoft.com/xml/xmlguide/schema-overview.asp

- The *Microsoft Schema Developer's Guide* provides an extensive coverage of schemas from a basic
- introduction to advanced definitions and uses.

msdn.microsoft.com/xml/reference/schema/start.asp

- The *Microsoft XML Schema Reference* contains an introduction to schema.

XML Schema Data Types

Table : XML Schema Data Types

Simple Type	User can independently define. This type is used when a restriction is placed on an embedded simple type to create and use a new type.
Complex Type	User can independently define. This type is used when the type has a child element or attribute.

Simple Type Example

- <xs:element name="Department" type="xs:string" />
- Here, the section described together with "xs:string" is an embedded simple type according to XML Schema. In this example, we have established the definition that the data type for the element called "Department" is a text string.

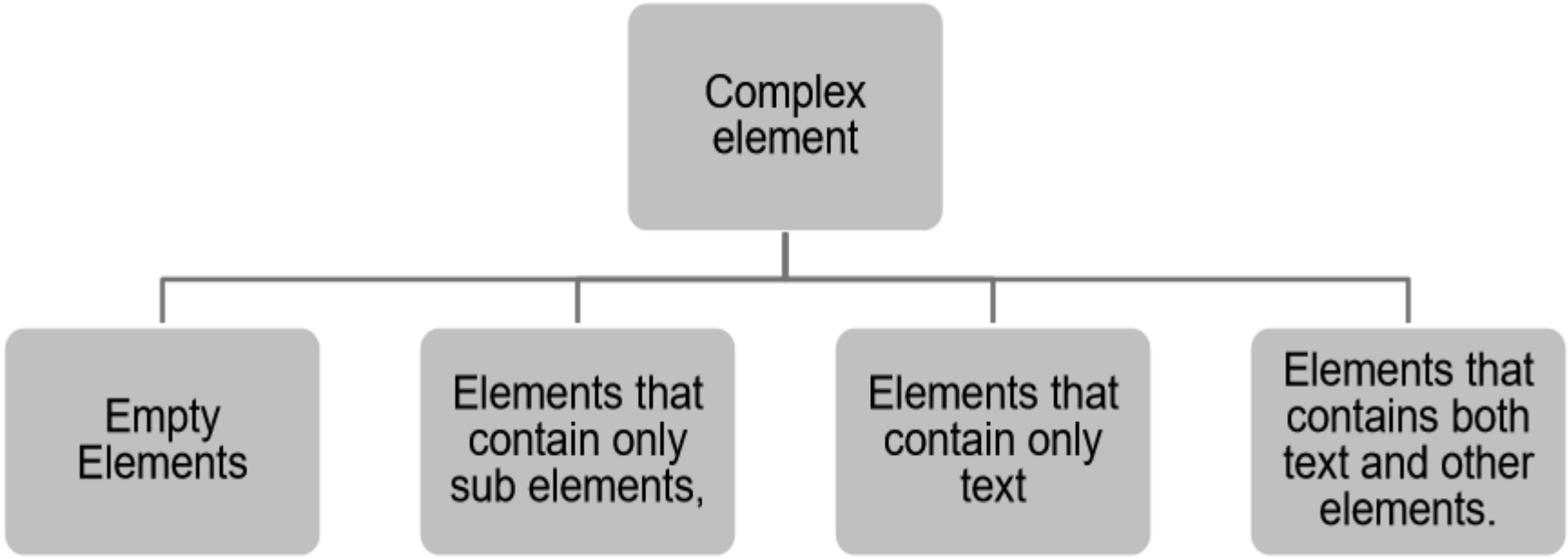
Complex Type Example

```
<xs:complexType name="EmployeeType">
  <xs:sequence maxOccurs="unbounded">
    <xs:element ref="Name" />
    <xs:element ref="Department" />
  </xs:sequence>
< /xs:complexType>
< xs:element name="Name" type="xs:string" />
< xs:element name="Department" type="xs:string" />
```

In this case the type name "EmployeeType" is designated by the name attribute of the complexType element. A model group (what designates the order of occurrence for the child element) is designated in the child element.

New types are created by placing restrictions on or extending simple or complex types. In this volume, we will discuss restrictions and extensions for simple types.

Complex Elements



Complex Empty Elements

```
<xs:element name="product">
  <xs:complexType>
    <xs:attribute name="prodid“
type="xs:positiveInteger"/>
  </xs:complexType>
</xs:element>
```

Or

```
<xs:element name="product" type="prodtype"/>

<xs:complexType name="prodtype">
  <xs:attribute name="prodid"
type="xs:positiveInteger"/>
</xs:complexType>
```

XSD Elements Only

```
<xs:element name="person">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="firstname" type="xs:string"/>
      <xs:element name="lastname" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Xml for this is:

```
<person>
  <firstname>John</firstname>
  <lastname>Smith</lastname>
</person>
```

XSD Text-Only Elements

This type contains only simple content (text and attributes), therefore we add a simpleContent element around the content. When using simple content, you must define an extension OR a restriction within the simpleContent element

```
<xs:element name="somename">
  <xs:complexType>
    <xs:simpleContent>

        ....
        ....

    </xs:simpleContent>
  </xs:complexType>
</xs:element>
```

XSD Mixed Content

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

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

Eg xml:

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

Examples:

```
<?xml version = "1.0"?>
<!-- Figure 7.20 : xml-schema.xsd --> <!-- Example W3C XML Schema -->
<xsd:schema xmlns:xsd = "http://www.w3.org/2000/10/XMLSchema">
<xsd:element name = "myMessage" type = "myMessageType"/>

<xsd:complexType name = "myMessageType">
<xsd:element ref = "greeting" minOccurs = "0“ maxOccurs = "1"/>
<xsd:element ref = "message" minOccurs =’1"  maxOccurs ="unbounded"/>
</xsd:complexType>
 <xsd:element name = "message" type = "xsd:string"/>
<xsd:element name = "greeting" type = "greetingType"/>
 <xsd:complexType name = "greetingType" content = "mixed">
</xsd:complexType>
</xsd:schema>
```
Fig. 7.18 W3C XML Schema document.

This shows an XML document that conforms to above schema.

```
<?xml version = "1.0"?>
<!-- Fig. 7.19 : intro3.xml -->
 <!-- Introduction to W3C XML Schema -->
<myMessage
xmlns:xsd = "http://www.w3.org/2000/10/XMLSchema-instance"
 xsd:noNamespaceSchemaLocation = "xml-schema.xsd">

 <greeting>Welcome to W3C XML Schema!</greeting>
 <message>This is a message.</message>
 <message>This is another message.</message>

</myMessage>
```

```
<!-- definition of attributes -->
<xs:attribute name="orderid" type="xs:string"/>
<!-- definition of complex elements -->
<xs:element name="shipto">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="name"/>
      <xs:element ref="address"/>
      <xs:element ref="city"/>
      <xs:element ref="country"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

```
<xs:element name="item">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="title"/>
      <xs:element ref="note" minOccurs="0"/>
      <xs:element ref="quantity"/>
      <xs:element ref="price"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="shiporder">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="orderperson"/>
      <xs:element ref="shipto"/>
      <xs:element ref="item" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute ref="orderid" use="required"/>
  </xs:complexType>
</xs:element>

</xs:schema>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<shiporder orderid="889923"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="shiporder.xsd">
  <orderperson>John Smith</orderperson>
  <shipto>
    <name>Ola Nordmann</name>
    <address>Langgt 23</address>
    <city>4000 Stavanger</city>
    <country>Norway</country>
  </shipto>
  <item>
    <title>Empire Burlesque</title>
    <note>Special Edition</note>
    <quantity>1</quantity>
    <price>10.90</price>
  </item>
  <item>
    <title>Hide your heart</title>
    <quantity>1</quantity>
    <price>9.90</price>
  </item>
</shiporder>
```

Ques. Same as previous but less organized.

```
<?xml version="1.0" encoding="UTF-8" ?>
<xs:schema
xmlns:xs="http://www.w3.org/2001/XMLSch
ema">

<xs:element name="shiporder">
 <xs:complexType>
  <xs:sequence>
   <xs:element name="orderperson"
type="xs:string"/>
    <xs:element name="shipto">
     <xs:complexType>
      <xs:sequence>
       <xs:element name="name"
type="xs:string"/>
       <xs:element name="address"
type="xs:string"/>
       <xs:element name="city"
type="xs:string"/>
       <xs:element name="country"
type="xs:string"/>
      </xs:sequence>
     </xs:complexType>
    </xs:element>
    <xs:element name="item"
maxOccurs="unbounded">
     <xs:complexType>
      <xs:sequence>
       <xs:element name="title"
type="xs:string"/>
       <xs:element name="note"
type="xs:string" minOccurs="0"/>
       <xs:element name="quantity"
type="xs:positiveInteger"/>
       <xs:element name="price"
type="xs:decimal"/>
      </xs:sequence>
     </xs:complexType>
    </xs:element>
  </xs:sequence>
  <xs:attribute name="orderid"
type="xs:string" use="required"/>
 </xs:complexType>
</xs:element>

</xs:schema>
```