## XML Schema

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

## What is a Simple Element?

A simple element is an XML element that can contain only text. It cannot contain any other elements or attributes.

However, 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.

You can also add restrictions (facets) to a data type in order to limit its content, or you can require the data to match a specific pattern.

## 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

Here are some XML elements:

```
<lastname>Refsnes</lastname>
<age>36</age>
<dateborn>1970-03-27</dateborn>
```

## 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"/>
```

## What is an Attribute?

Simple elements cannot have attributes. If an element has attributes, it is considered to be of a complex type. But the attribute itself is always declared as a simple type.

## How to Define an Attribute?

The syntax for defining an attribute is:

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

where xxx is the name of the attribute and yyy specifies the data type of the attribute.

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.

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

```
<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"/>
```

## 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="120"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

## What is a Complex Element?

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!

## Examples of Complex Elements

A complex XML element, "product", which is empty:

```
<product pid="1345"/>
```

A complex XML element, "employee", which contains only other elements:

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

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>
It happened on <date lang="norwegian">03.03.99</date> ....
</description>
```

## How to Define a Complex Element

Look at this complex XML element, "employee", which contains only other elements:

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

We can define a complex element in an XML Schema two different ways:

1. The "employee" element can be declared directly by naming the element, like this:

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

If you use the method described above, only the "employee" element can use the specified complex type. Note that the child elements, "firstname" and "lastname", are surrounded by the <sequence> indicator. This means that the child elements must appear in the same order as they are declared. You will learn more about indicators later.

2. The "employee" element can have a type attribute that refers to the name of the complex type to use:

```
<xs:element name="employee" type="personinfo"/>

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

If you use the method described above, several elements can refer to the same complex type, like this:

```
<xs:element name="employee" type="personinfo"/>
<xs:element name="student" type="personinfo"/>
<xs:element name="member" type="personinfo"/>

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

You can also base a complex element on an existing complex element and add some elements, like this:

```
<xs:element name="employee" type="fullpersoninfo"/>

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

<xs:complexType name="fullpersoninfo">
  <xs:complexContent>
    <xs:extension base="personinfo">
      <xs:sequence>
        <xs:element name="address" type="xs:string"/>
        <xs:element name="city" type="xs:string"/>
        <xs:element name="country" type="xs:string"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

## Complex Empty Elements

An empty complex element cannot have contents, only attributes.

An empty XML element:

```
<product prodid="1345" />
```

The "product" element above has no content at all. To define a type with no content, we must define a type that allows elements in its content, but we do not actually declare any elements, like this:

```
<xs:element name="product">
  <xs:complexType>
    <xs:complexContent>
      <xs:restriction base="xs:integer">
        <xs:attribute name="prodid" type="xs:positiveInteger"/>
      </xs:restriction>
    </xs:complexContent>
  </xs:complexType>
</xs:element>
```

In the example above, we define a complex type with a complex content. The complexContent element signals that we intend to restrict or extend the content model of a complex type, and the restriction of integer declares one attribute but does not introduce any element content.

However, it is possible to declare the "product" element more compactly, like this:

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

Or you can give the complexType element a name, and let the "product" element have a type attribute that refers to the name of the complexType (if you use this method, several elements can refer to the same complex type):

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

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

## Complex Types Containing Elements Only

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

An XML element, "person", that contains only other elements:

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

You can define the "person" element in a schema, like this:

```
<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>
```

Notice the <xs:sequence> tag. It means that the elements defined ("firstname" and "lastname") must appear in that order inside a "person" element.

Or you can give the complexType element a name, and let the "person" element have a type attribute that refers to the name of the complexType (if you use this method, several elements can refer to the same complex type):

```
<xs:element name="person" type="persontype"/>
```

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

## Complex Text-Only Elements

A complex text-only element can contain text and attributes.

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, like this:

```
<xs:element name="somename">
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="basetype">
        ....
        ....
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>

OR

<xs:element name="somename">
  <xs:complexType>
    <xs:simpleContent>
      <xs:restriction base="basetype">
        ....
        ....
      </xs:restriction>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>
```

**Tip:** Use the extension/restriction element to expand or to limit the base simple type for the element.

Here is an example of an XML element, "shoesize", that contains text-only:

```
<shoesize country="france">35</shoesize>
```

The following example declares a complexType, "shoesize". The content is defined as an integer value, and the "shoesize" element also contains an attribute named "country":

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

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

We could also give the complexType element a name, and let the "shoesize" element have a type attribute that refers to the name of the complexType (if you use this method, several elements can refer to the same complex type):

```
<xs:element name="shoesize" type="shoetype"/>

<xs:complexType name="shoetype">
  <xs:simpleContent>
    <xs:extension base="xs:integer">
      <xs:attribute name="country" type="xs:string" />
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>
```

## Complex Types with Mixed Content

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

An XML element, "letter", that contains both text and other elements:

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

The following schema declares the "letter" element:

```
<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>
```

**Note:** To enable character data to appear between the child-elements of "letter", the mixed attribute must be set to "true". The <xs:sequence> tag means that the elements defined (name, orderid and shipdate) must appear in that order inside a "letter" element.

We could also give the complexType element a name, and let the "letter" element have a type attribute that refers to the name of the complexType (if you use this method, several elements can refer to the same complex type):

```
<xs:element name="letter" type="lettertype"/>

<xs:complexType name="lettertype" 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>
```

## Indicators

We can control HOW elements are to be used in documents with indicators.

There are seven indicators:

Order indicators:

- All
- Choice
- Sequence

Occurrence indicators:

- maxOccurs
- minOccurs

Group indicators:

- Group name
- attributeGroup name

## Order Indicators

Order indicators are used to define the order of the elements.

### All Indicator

The <all> indicator specifies that the child elements can appear in any order, and that each child element must occur only once:

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

**Note:** When using the <all> indicator you can set the <minOccurs> indicator to 0 or 1 and

### Sequence Indicator

The <sequence> indicator specifies that the child elements must appear in a specific order:

```
<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>
```

## Occurrence Indicators

Occurrence indicators are used to define how often an element can occur.

**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.

### maxOccurs Indicator

The <maxOccurs> indicator specifies the maximum number of times an element can occur:

```
<xs:element name="person">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="full_name" type="xs:string"/>
      <xs:element name="child_name" type="xs:string" maxOccurs="10"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

The example above indicates that the "child_name" element can occur a minimum of one time (the default value for minOccurs is 1) and a maximum of ten times in the "person" element.

### minOccurs Indicator

The <minOccurs> indicator specifies the minimum number of times an element can occur:

```
<xs:element name="person">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="full_name" type="xs:string"/>
      <xs:element name="child_name" type="xs:string"
      maxOccurs="10" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

The example above indicates that the "child_name" element can occur a minimum of zero times and a maximum of ten times in the "person" element.

**Tip:** To allow an element to appear an unlimited number of times, use the maxOccurs="unbounded" statement

## An XSD Example

The following example will demonstrate how to write an XML Schema. You will also learn that a schema can be written in different ways.

Let's have a look at this XML document called "shiporder.xml":

```
<?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>
```

The XML document above consists of a root element, "shiporder", that contains a required attribute called "orderid". The "shiporder" element contains three different child elements: "orderperson", "shipto" and "item". The "item" element appears twice, and it contains a "title", an optional "note" element, a "quantity", and a "price" element.

The line above: xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" tells the XML parser that this document should be validated against a schema. The line: xsi:noNamespaceSchemaLocation="shiporder.xsd" specifies WHERE the schema resides (here it is in the same folder as "shiporder.xml").

## Create an XML Schema

Now we want to create a schema for the XML document above.

We start by opening a new file that we will call "shiporder.xsd". To create the schema we could simply follow the structure in the XML document and define each element as we find it. We will start with the standard XML declaration followed by the xs:schema element that defines a schema:

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

```
...
</xs:schema>
```

In the schema above we use the standard namespace (xs), and the URI associated with this namespace is the Schema language definition, which has the standard value of http://www.w3.org/2001/XMLSchema.

## Using Named Types

The effective design method defines classes or types, that enables us to reuse element definitions. This is done by naming the simpleTypes and complexTypes elements, and then point to them through the type attribute of the element.

Here is the third design of the schema file ("shiporder.xsd"):

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

<xs:simpleType name="stringtype">
  <xs:restriction base="xs:string"/>
</xs:simpleType>

<xs:simpleType name="inttype">
  <xs:restriction base="xs:positiveInteger"/>
</xs:simpleType>

<xs:simpleType name="dectype">
  <xs:restriction base="xs:decimal"/>
</xs:simpleType>

<xs:simpleType name="orderidtype">
  <xs:restriction base="xs:string">
    <xs:pattern value="[0-9]{6}"/>
  </xs:restriction>
</xs:simpleType>

<xs:complexType name="shiptotype">
  <xs:sequence>
    <xs:element name="name" type="stringtype"/>
    <xs:element name="address" type="stringtype"/>
    <xs:element name="city" type="stringtype"/>
    <xs:element name="country" type="stringtype"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="itemtype">
  <xs:sequence>
    <xs:element name="title" type="stringtype"/>
    <xs:element name="note" type="stringtype" minOccurs="0"/>
    <xs:element name="quantity" type="inttype"/>
    <xs:element name="price" type="dectype"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="shipordertype">
  <xs:sequence>
    <xs:element name="orderperson" type="stringtype"/>
    <xs:element name="shipto" type="shiptotype"/>
    <xs:element name="item" maxOccurs="unbounded" type="itemtype"/>
  </xs:sequence>
  <xs:attribute name="orderid" type="orderidtype" use="required"/>
</xs:complexType>

<xs:element name="shiporder" type="shipordertype"/>

</xs:schema>
```

The restriction element indicates that the datatype is derived from a W3C XML Schema namespace datatype. So, the following fragment means that the value of the element or attribute must be a string value:

```
<xs:restriction base="xs:string">
```

The restriction element is more often used to apply restrictions to elements. Look at the following lines from the schema above:

```
<xs:simpleType name="orderidtype">
  <xs:restriction base="xs:string">
    <xs:pattern value="[0-9]{6}"/>
  </xs:restriction>
</xs:simpleType>
```

This indicates that the value of the element or attribute must be a string, it must be exactly six characters in a row, and those characters must be a number from 0 to 9.

**Another way to create  the schema**

We start by opening a new file that we will call "shiporder.xsd". To create the schema we could simply follow the structure in the XML document and define each element as we find it. We will start with the standard XML declaration followed by the xs:schema element that defines a schema:

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

In the schema above we use the standard namespace (xs), and the URI associated with this namespace is the Schema language definition, which has the standard value of http://www.w3.org/2001/XMLSchema.

Next, we have to define the "shiporder" element. This element has an attribute and it contains other elements, therefore we consider it as a complex type. The child elements of the "shiporder" element is surrounded by a xs:sequence element that defines an ordered sequence of sub elements:

```
<xs:element name="shiporder">
  <xs:complexType>
    <xs:sequence>
      ...
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Then we have to define the "orderperson" element as a simple type (because it does not contain any attributes or other elements). The type (xs:string) is prefixed with the namespace prefix associated with XML Schema that indicates a predefined schema data type:

```
<xs:element name="orderperson" type="xs:string"/>
```

Next, we have to define two elements that are of the complex type: "shipto" and "item". We start by defining the "shipto" element:

```
<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>
```

With schemas we can define the number of possible occurrences for an element with the maxOccurs and minOccurs attributes. maxOccurs specifies the maximum number of occurrences for an element and minOccurs specifies the minimum number of occurrences for an element. The default value for both maxOccurs and minOccurs is 1!

Now we can define the "item" element. This element can appear multiple times inside a "shiporder" element. This is specified by setting the maxOccurs attribute of the "item" element to "unbounded" which means that there can be as many occurrences of the "item" element as the author wishes. Notice that the "note" element is optional. We have specified this by setting the minOccurs attribute to zero:

```
<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>
```

We can now declare the attribute of the "shiporder" element. Since this is a required attribute we specify use="required".

**Note:** The attribute declarations must always come last:

```
<xs:attribute name="orderid" type="xs:string" use="required"/>
```

Here is the complete listing of the schema file called "shiporder.xsd":

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

<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>
```

The previous design method is very simple, but can be difficult to read and maintain when documents are complex.