<?xml?>

XML – Part A

Introduction, Specification

Part 2 : Specifying XML Documents

Specifying XML Documents

In order to be treated correctly, XML documents must be

- **well-formed**, and
- **valid**.

well-formed : Each opening tag has its corresponding end tag,

```
<street>Main Street<number>24</street>
```

… and tags are strictly nested

```
<street>Main Street<number>24</street></number>
```

valid : The tag names, their nesting, the value types etc. are specified by a specification

Specifying XML Documents

XML Documents may be specified using 2 different techniques:

- DTD (Document Type Definition)

  A Document Type Definition defines the legal building blocks of an XML document. It defines the document structure with a list of legal elements.
  A DTD can be declared inline in your XML document, or as an external reference.

- XML Schema Definition

  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

Specifying an XML Document using a DTD

A DTD may be declared inline (in the document) or separately (in a separate file).

– Internal Declaration

Type definition enclosed in brackets

```
<?xml version="1.0"?>
<!DOCTYPE Hello [ <!ELEMENT Hello (#PCDATA)> ] >
<Hello>
  Hello Jupiter!
</Hello>
```

root element: Must be the same in DTD and in document

Start of declaration

What‘s the difference between a DTD and a Schema?

Both describe the structure and the components of XML documents for both a human or a machine reader. BUT, Schemas…

+ are written in XML while DTDs have their own syntax
+ provide more detailed data types (over 25 predefined types, as well as user-definable types)
+ allow for more details in the specification of structures
+ may be annotated (description of the document’s purpose, comments to element or attribute definitions)
+ provide the means to use namespaces.
+ allow extensions.

BUT, Schemas…

– are quite a bit more complicated to write and read than DTDs

Specifying an XML Document using an XML Schema

A Schema is always an external file which must be referenced by the **`schemaLocation`** attribute:

```
<address
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://example.org/blah
                        http://example.org/blah/address.xsd">
  <firstname>John</firstname>
  <name>Smith</name>
  <street>45th Avenue</street>
  <number>16</number>
  <apt-number>3</apt-number>
  <city>New York</city>
  <zip>10002</zip>
  <country>USA</country>
</address >
```

Namespace

Location

XML Schema

Thus, let‘s start …

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

This URL is mandatory.
Though there has been a new version in 2004, the XML world insists on the 2001 version!

XML Schema: Elements

a) Simple Elements

any standard XML type ☞☞☞

```
<xsd:element name="firstname" type="xsd:string"/>
```

or

```
<xsd:element name="firstname" type="xsd:string" default="Joe"/>
<xsd:element name="firstname" type="xsd:string" fixed="Bob"/>
```

default or fixed values

XML Schema: Standard Types

In XML, standard types are defined for

- strings/tokens

  `xsd:string, xsd:token`

- Date, Time, DateTime and Duration types

  `xsd:date, xsd:time, xsd:duration, xsd:gDay, gMonthDay...`

- Numbers (Decimal ≈ double, integer numbers from byte … long)

  `xsd:decimal, xsd:integer, xsd:unsignedLong, xsd:byte...`

- boolean

  `xsd:boolean`

- binary data

  `xsd:hexBinary, xsd:base64Binary`

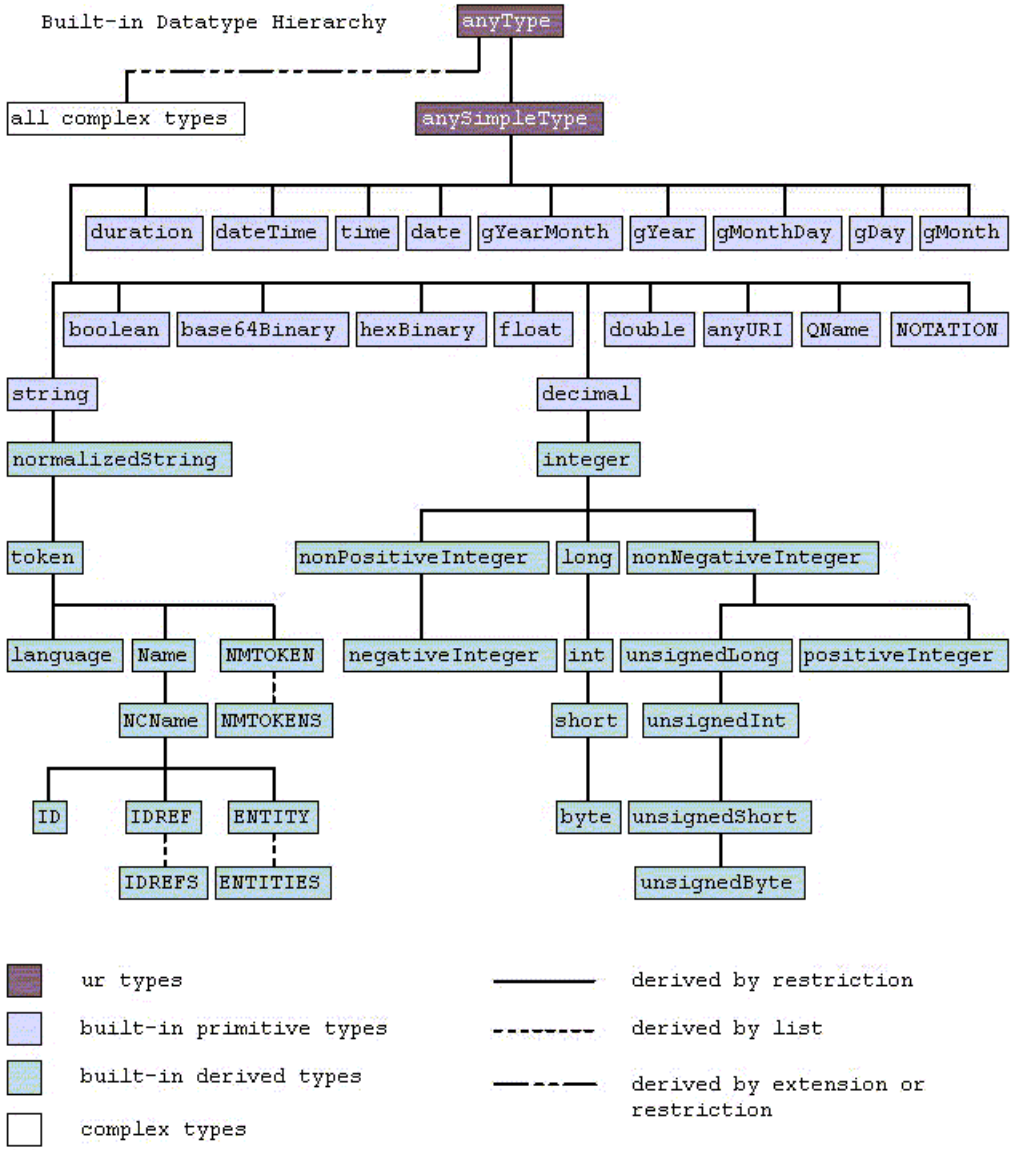
- URI

  `xsd:anyURI`

- `...`

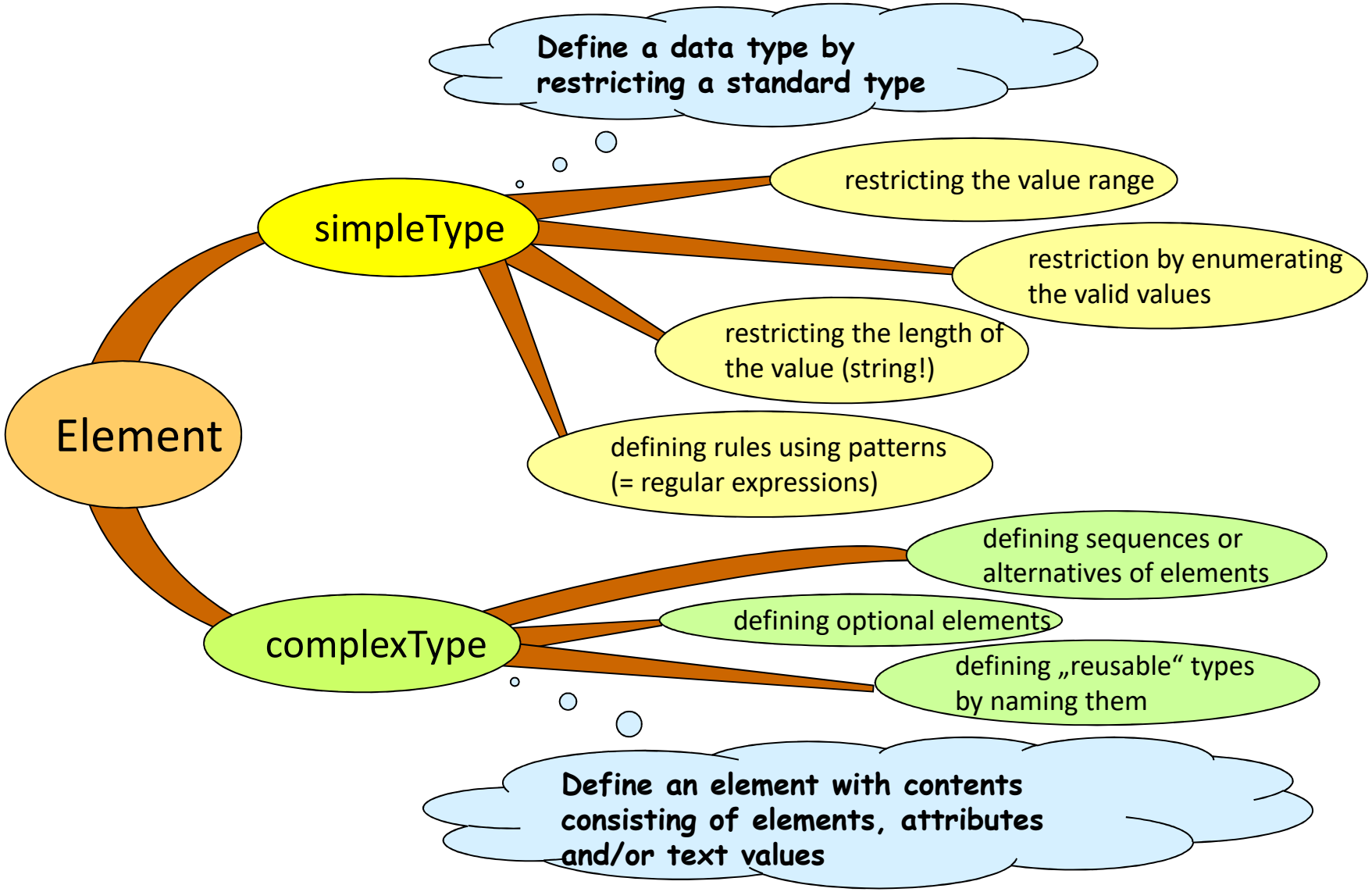
see separate sheet

XML Schema: Standard Types



W3C
XML Schema Part 2
Datatypes

XML Schema: Typed Elements



XML Schema: simpleType Elements – Value Range

- Restrictions on Value Ranges

```
<xsd:element name="age">
  <xsd:simpleType>
    <xsd:restriction base="xsd:integer">
       <xsd:minInclusive value="0"/>
       <xsd:maxInclusive value="150"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:element>
```

Indicates which basic type is used and restricted

Defining a restricted interval, for example

  - Also **`min/maxExclusive`** possible
  - Example xml:

```
<age>75</age>
```

<age> is restricted to values between 0 und 150

XML Schema: simpleType Elements – Set of Values

- Restrictions on a Set of Values

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

Defining a list of admitted values

- Example xml:

```
<car>Audi</car>
```

<car> is restricted to 3 values

XML Schema: simpleType Elements - Patterns

- Restrictions on a Series of Values (Patterns)

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

Indicates which basic type is used and restricted

Only lower case letter allowed

- Example xml:

```
<letter>abc</letter>
```

<letter> is restricted to lower cases

XML Schema: simpleType Elements - Trim

- Remove Extra White Space Characters

```
<xsd:element name="singleSpace">
  <xsd:simpleType>
     <xsd:restriction base="xsd:string">
        <xsd:whiteSpace value="collapse"/>
     </xsd:restriction>
  </xsd:simpleType>
</xsd:element>
```

Each contiguous group of white space characters will be collapsed into one single space

- - Example xml:

```
<singleSpace>
this   is   a    sentence
</singleSpace>
```

White spaces collapsed when processed

XML Schema: complexType Elements 1/4

- Each element containing either other elements or text values, or having attributes is a **complexType** element.
- Contents may be sequences, alternatives and optional elements, all characterized by a cardinality.

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

XML Schema: complexType Elements 2/4

`xsd:sequence`: You may specify the **number of occurrences** of enclosed elements;

```
<xsd:element name="employee">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="prenom" type="xsd:string"
        minOccurs="1" maxOccurs="5"/>
      <xsd:element name="nom" type="xsd:string"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

minOccurs = maxOccurs = 1
if not specified

maxOccurs = unbounded means
unlimited number of repetitions

XML Schema: complexType Elements 3/4

`xsd:choice:` You may choose among **alternative elements**; :

```
<xsd:element name="note">
  <xsd:complexType>
    <xsd:choice>
      <xsd:element name="to" type="xsd:string"/>
      <xsd:element name="from" type="xsd:string"/>
      <xsd:element name="header" type="xsd:string"/>
      <xsd:element name="message" type="xsd:string"/>
    </xsd:choice>
  </xsd:complexType>
</xsd:element>
```

Exactly one of the elements listed within "choice" may appear as a child element of "note"

XML Schema: complexType Elements 4/4

`xsd:all:` You may specify that all elements must appear exactly once but in an arbitrary order:

```
<xsd:element name="note">
  <xsd:complexType>
    <xsd:all>
      <xsd:element name="to" type="xsd:string"/>
      <xsd:element name="from" type="xsd:string"/>
      <xsd:element name="header" type="xsd:string"/>
      <xsd:element name="message" type="xsd:string"/>
    </xsd:all>
  </xsd:complexType>
</xsd:element>
```

Each of the elements listed within "all" may appear just once but in any order

XML Schema: Defining Attributes 1/3

Attributes are defined similarly to simple elements:

```
<xsd:attribute name="xxxx" type="yyyy"/>
```

Example: In the element

```
<lastname lang="DE"> ... </lastname>
```

the **lang** attribute is defined as:

As soon as an element has an attribute, it is considered to have a `complexType`

Attribute definitions AFTER element definitions

```
<xsd:element name="lastname">
  <xsd:complexType>
    ...
    <xsd:attribute name="lang" type="xsd:string"/>
  </xsd:complexType>
</xsd:element>
```

XML Schema: Defining Attributes 2/3

As with simple elements, the values of attributes may be defined as default or fixed values:

```
<xsd:attribute name="xxxx" type="yyyy" default="zzzz"/>
<xsd:attribute name="xxxx" type="yyyy" fixed="ffff"/>
```

`Fixed` means that the user is not allowed to specify a value himself.

Example: In the element

```
<productname lang="FR"> ... </productname>
```

we could preset the **lang** attribute as:

If the document explicitly specifies a value for `lang`, this value is used; otherwise, `EN` will be used.

```
<xsd:element name=«productname">
  <xsd:complexType>
    ...
    <xsd:attribute name="lang" type="xsd:string" default="EN"/>
  </xsd:complexType>
</xsd:element>
```

XML Schema: Defining Attributes 3/3

By default, attributes are assumed to be optional.

If you want an attribute to be mandatory, add the attribute „use“:

```
<xsd:attribute name="xxxx" type="yyyy" use="required"/>
```

The attribute `use` may have the value `required` or `optional` which is the default value.

```
<productname lang="FR"> ... </productname>
```

XML Schema: Combining All Kinds of Nodes 1/3

Text values can be used in several ways:

1. An element contains **only a single text value**:
   → The element is declared as a simpleElement:

```
<xsd:element name="firstname" type="xsd:string"/>
```

Example:

```
<firstname>John</firstname>
```

XML Schema: Combining All Kinds of Nodes 2/3

2. An element contains **text** but also an **attribute**:

Specifying an attribute always requires a `complexType`.

```
<xsd:element name="shoesize">
  <xsd:complexType mixed="true">
    <xsd:attribute name="country" type="xsd:string"/>
  </xsd:complexType>
</xsd:element>
```

Text + anything else requires attribute mixed="true"

This defines the attribute.

Example:

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

XML Schema: Combining All Kinds of Nodes 3/3

3. An element contains intermixed **text values and elements (and attributes)**:

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

allows for mixing text and elements

elements contained in `letter`

any attributes go here (after the sequence!)

Example:

```
<letter format="A4">
Dear Mr. <name>John Smith</name>,
Your order <orderid>1032</orderid>
will be shipped on <shipdate>2007-01-05</shipdate>
</letter>
```

Exercise Hotel

Write an XML Schema for the following XML document:

```
<?xml version="1.0" encoding="UTF-8" ?>
  <!-- Restrict the stars to values between 1 and 7 -->
  <hotel stars="3">
      <!-- each element should occur only once -->
      <!-- only positive numbers are allowed -->
      <single-room> 12 </single-room>
      <double-room> 26 </double-room>
      <suite> 2 </suite>
      <!-- optional empty element -->
      <lounge/>
</hotel>
```

XML Schema: An Overall Example 1/3

We want to define a schema for the following document:

```
<book isbn="123-456-789" year-of-issue="1699">
  <title>King Lear</title>
  <author>William Shakespeare</author>
  <character>
    <name>King Lear</name>
    <age>75</age>
  </character>
  <character>
    <name>XYZ</name>
    <age>25</age>
  </character>
</book>
```

XML Schema: An Overall Example 2/3

```
<?xml version="1.0" encoding="utf-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="book">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="title" type="xsd:string"/>
        <xsd:element name="author" type="xsd:string"/>
        <xsd:element name="character" minOccurs="1"
                                      maxOccurs="unbounded">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name="name" type="xsd:string"/>
              <xsd:element name="age" type="xsd:decimal"/>
            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
      </xsd:sequence>
      <xsd:attribute name="isbn" type="xsd:string"/>
      <xsd:attribute name="year-of-issue" type="xsd:gYear"/>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

XML Schema: An Overall Example 3/3

A „Russian Doll“ example:

- nested structures
- structure similar to the specified XML document
- relations between schema parts described by nesting:
  - no possibility to „re-use“ parts of the schema
  - not easily readable
  - not easily maintainable

XML Schema: Naming and Referencing Types

```
<?xml version="1.0" encoding="utf-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <!- defining simple types -->
  <xsd:simpleType name="nameType">
    <xsd:restriction base="xsd:string">
      <xsd:minLength value="1"/>
      <xsd:maxLength value="32"/>
    </xsd:restriction>
  </xsd:simpleType>
  ...
  <xsd:complexType name="bookType">
    <xsd:sequence>
      <xsd:element name="title" type="nameType"/>
      <xsd:element name="author" type="nameType"/>
      <xsd:element name="character"  type="nameType"
        minOccurs="0" maxOccurs="unbounded"/>
    <xsd:sequence>
    <xsd:attribute name="isbn" type="isbnType"/>
  </xsd:complexType>

  <xsd:element name="book" type="bookType"/>
</xsd:schema>
```

A type definition for a simpleType

More type definitions for other simpleTypes and complexTypes

Finally, the entire document is defined by one single element!

Exercise Hotel 2

Once more:

Write a specific type definition for the stars attribute allowing only 1 .. 5 stars.

Define types and attributes.

```
<?xml version="1.0" encoding="UTF-8" ?>
<hotel stars="***">
  <single-room> 12 </single-room>
  <double-room> 26 </double-room>
  <suite> 2 </suite>
  <!-- optional element -->
  <lounge/>
</hotel>
```

XML Schema: Defining a Namespace for a Schema

Namespaces can be used in two ways:

- Defining our own namespace for the schema

```
<?xml version="1.0" encoding="utf-8"?>
<xsd:schema targetNamespace="http://example.org/myself/myschema"
            xmlns="http://example.org/myself/myschema"
            xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            elementFormDefault="qualified">
...
</xsd:schema>
```

The standard XMLSchema namespace to use for the schema definition itself.

In order to use this namespace here, we declare it as the default namespace within this schema.
This means that we don't have to write a prefix for our own (element/attribute…) definitions

Everybody writing documents according to this schema must use this namespace.
If no **targetNamespace** is specified, **noNamespace** will be used.

All newly defined elements belong to the namespace.

XML Schema: Defining a Namespace for a Schema

- Using one or more namespaces defined elsewhere

```
<?xml version="1.0" encoding="utf-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            xmlns:other="http://somebody.else.org/someschema">
  ...
</xsd:schema>
```

Again:
The standard namespace to use
for the schema definition itself

Using an already existing namespace

Use of Namespace

- Namespaces are a mechanism for breaking up your schemas.
- Up until now, we had only a single schema file containing all element definitions,
- but the XSD standard allows you to structure your XSD schemas by breaking them into multiple files.

Example: In this example, the schema is broken out into 4 files.

- **CommonTypes** - this could contain all your basic types, AddressType, PriceType, PaymentMethodType etc.
- **CustomerTypes** - this could contain all your definitions for your customers.
- **OrderType** - this could contain the definitions for orders.
- **Main** - this would pull all the sub schemas together into a single schema, and define your main element/s.

Example: CommonTypes.xsd 1/2

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema
     xmlns:xsd="http://www.w3.org/2001/XMLSchema"
     targetNamespace=http://NamespaceTest.com/CommonTypes
     elementFormDefault="qualified">

    <xsd:complexType name="AddressType">
        <xsd:sequence>
            <xsd:element name="Line1" type="xsd:string" />
            <xsd:element name="Line2" type="xsd:string" />
        </xsd:sequence>
    </xsd:complexType>
```

Unique Identifier as targetNamespace

Reusable item (type)

Example: CommonTypes.xsd 2/2

```
    <xsd:simpleType name="PriceType">
        <xsd:restriction base="xsd:decimal">
            <xsd:fractionDigits value="2" />
        </xsd:restriction>
    </xsd:simpleType>

    <xsd:simpleType name="PaymentMethodType">
        <xsd:restriction base="xsd:string">
            <xsd:enumeration value="VISA" />
            <xsd:enumeration value="MasterCard" />
            <xsd:enumeration value="Cash" />
            <xsd:enumeration value="Amex" />
        </xsd:restriction>
    </xsd:simpleType>
</xsd:schema>
```

Reusable items (types)

Example: CustomTypes.xsd

Unique Identifier as targetNamespace

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http://NamespaceTest.com/CustomerTypes"
    xmlns:cmn="http://NamespaceTest.com/CommonTypes"
    elementFormDefault="qualified">

    <xsd:import schemaLocation="CommonTypes.xsd"
    namespace="http://NamespaceTest.com/CommonTypes" />

    <xsd:complexType name="CustomerType">
        <xsd:sequence>
            <xsd:element name="Name" type="xsd:string" />
            <xsd:element name="DeliveryAddress" type="cmn:AddressType" />
            <xsd:element name="BillingAddress" type="cmn:AddressType" />
        </xsd:sequence>
    </xsd:complexType>
</xsd:schema>
```

Import of another schema

Use of type from another schema

Example: OrderType.xsd

Unique Identifier as targetNamespace

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http://NamespaceTest.com/OrderType"
    xmlns:cmn="http://NamespaceTest.com/CommonTypes"
    elementFormDefault="qualified">

<xsd:import namespace="http://NamespaceTest.com/CommonTypes"
    schemaLocation="CommonTypes.xsd" />

<xsd:complexType name="OrderType">
    <xsd:sequence>
        <xsd:element maxOccurs="unbounded" name="Item">
            <xsd:complexType>
                <xsd:sequence>
                    <xsd:element name="ProductName" type="xsd:string" />
                    <xsd:element name="Quantity" type="xsd:int" />
                    <xsd:element name="UnitPrice" type="cmn:PriceType" />
                </xsd:sequence>
            </xsd:complexType>
        </xsd:element>
    </xsd:sequence>
</xsd:complexType>
</xsd:schema>
```

Import of another schema

Use of type from another schema

Example: Main.xsd 1/2

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http://NamespaceTest.com/Purchase"
    xmlns:ord="http://NamespaceTest.com/OrderType"
    xmlns:cmn="http://NamespaceTest.com/CommonTypes"
    xmlns:cust="http://NamespaceTest.com/CustomerTypes"
    elementFormDefault="qualified">

<xsd:import schemaLocation="CommonTypes.xsd"
    namespace="http://NamespaceTest.com/CommonTypes" />
<xsd:import schemaLocation="CustomerTypes.xsd"
    namespace="http://NamespaceTest.com/CustomerTypes" />
<xsd:import schemaLocation="OrderType.xsd"
    namespace="http://NamespaceTest.com/OrderType" />
```

Unique Identifier as targetNamespace

Definition of alias

Import of used schemata

Example: Main.xsd 2/2

```
  <xsd:element name="Purchase">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="OrderDetail" type="ord:OrderType" />
        <xsd:element name="PaymentMethod" type="cmn:PaymentMethodType" />
        <xsd:element name="CustomerDetails" type="cust:CustomerType" />
     </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

</xsd:schema>
```

Use of types from other schemata with different alias

Example: Purchase.xml 1/2

```
<?xml version="1.0"?>
<p:Purchase
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://NamespaceTest.com/Purchase Main.xsd"
    xmlns:p="http://NamespaceTest.com/Purchase"
    xmlns:o="http://NamespaceTest.com/OrderType"
    xmlns:c="http://NamespaceTest.com/CustomerTypes"
    xmlns:cmn="http://NamespaceTest.com/CommonTypes">

    <p:OrderDetail>
        <o:Item>
            <o:ProductName>Widget</o:ProductName>
            <o:Quantity>1</o:Quantity>
            <o:UnitPrice>3.42</o:UnitPrice>
        </o:Item>
    </p:OrderDetail>

    <p:PaymentMethod>VISA</p:PaymentMethod>
```

Root element

Schema location

Definition of alias

alias = "p" because defined in Main.xsd

alias = "o" because defined in OrderType.xsd

Example: Purchase.xml 2/2

```
    <p:CustomerDetails>
        <c:Name>James</c:Name>
        <c:DeliveryAddress>
            <cmn:Line1>15 Some Road</cmn:Line1>
            <cmn:Line2>SomeTown</cmn:Line2>
        </c:DeliveryAddress>
        <c:BillingAddress>
            <cmn:Line1>15 Some Road</cmn:Line1>
            <cmn:Line2>SomeTown</cmn:Line2>
        </c:BillingAddress>
    </p:CustomerDetails>
</p:Purchase>
```

alias = "p" because defined in Main.xsd

alias = "c" because defined in CustomerType.xsd

Use of Namespace – general rules

- The alias must be the same as the target namespace in which the **element** is defined.
- It is important to note that this is where the element is defined - not where the *complexType* is defined.
- Example:
  - So the element <OrderDetail> is actually defined in main.xsd so it is part of the namespace "http://NamespaceTest.com/Purchase", even though it uses the complexType "OrderType" which is defined in the OrderTypes.xsd.
  - The contents of <OrderDetail> are defined within the complexType "OrderType", which is in the target namespace "http://NamespaceTest.com/OrderTypes", so the child element <Item> needs qualifying within the namespace "http://NamespaceTest.com/OrderTypes".

Specifying XML Documents

References:

XML Schema Definition

- XML Schema Part 0: Primer Second Edition
  http://www.w3.org/TR/xmlschema-0/

- XML Schema Teil 0: Einführung.
  http://www.edition-w3c.de/TR/2001/REC-xmlschema-0-20010502/

DTD (Document Type Definition)

- Extensible Markup Language (XML) 1.0 (Fifth Edition)
  https://www.w3.org/TR/xml/

- Dokumenttyp-Definitionen (DTDs)
  http://de.selfhtml.org/xml/dtd/index.htm