

# XML Schema

Mario Alviano

University of Calabria, Italy

A.Y. 2018/2019

- 1 Introduction
- 2 Elements
- 3 Simple and complex types
- 4 Attributes
- 5 Groups and built-in
- 6 Import of other schemes
- 7 Exercises

# Introduction: Why XML Schema?

- DTDs allow to define the schema of XML applications
- DTDs are constrained to several limitations
  - No types for CDATA and #PCDATA
  - No strict control of mixed content
  - No support for namespaces
- XML Schema fulfils all these lacks
- XML Schema uses XML syntax
  - XML Schema is an XML application!
  - **Convention:** XML Schema files end with **.xsd**

## Example. Schema declaration

```
<person
  xmlns="http://alviano.net/km"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://alviano.net/km
    person.xsd">
```

# Root element

- Elements of XML Schema are defined in the namespace `http://www.w3.org/2001/XMLSchema`

- **Convention:** use the prefix `xs`

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  targetNamespace="URI"
  attributeFormDefault="qualified or unqualified"
  elementFormDefault="qualified or unqualified"
  version="version number">
```

- `targetNamespace` (optional) specifies to which namespace the elements defined by the schema belong
  - **Hint:** set the default namespace equal to `targetNamespace`
- Elements and attributes with namespace are said **qualified**
  - Qualified/unqualified names can be forced with the attributes `attributeFormDefault` and `elementFormDefault`

# Element declarations

- Elements are declared by `<xs:element>`
- Elements can be global or local
  - A global declaration is a child of `<xs:schema>` (the root)
  - A local declaration is a child of some other tag

```
<xs:element name="name of the element"  
  type="global type"  
  ref="global element declaration"  
  form="qualified or unqualified"  
  minOccurs="non-negative number"  
  maxOccurs="non-negative number or 'unbounded' "  
  default="default value"  
  fixed="fixed value">
```

## Warning!

Attributes `minOccurs` and `maxOccurs` cannot be used in global declarations

# Type declarations

- Types can be global or local
- Types can be complex
  - Structured types
- Types can be simple
  - Specializations of other types
- There is another, special type named wildcard (or jolly)

- Complex types are declared by `<xs:complexType>`
- Their content is a list of elements and types, to be interpreted as
  - `<xs:sequence>`: elements must appear in the given order
  - `<xs:choice>`: only one element of the list can appear
  - `<xs:all>`: elements must appear once (unless optional), in any order

## Example

```
<xs:complexType name="NameOrEmail">
  <xs:choice>
    <xs:element name="email" type="xs:string"/>
    <xs:sequence>
      <xs:element name="first" type="xs:string"/>
      <xs:element name="middle" type="xs:string"/>
      <xs:element name="last" type="xs:string"/>
    </xs:sequence>
  </xs:choice>
</xs:complexType>
```

- `<xs:all>` is subject to several restrictions
  - It must be the only element in `<xs:complexType>`
  - It can contain only `<xs:element>`
  - Every `<xs:element>` can appear only once

## Example

```
<xs:element name="person-name">
  <xs:complexType>
    <xs:all>
      <xs:element name="first" type="xs:string"/>
      <xs:element name="middle" type="xs:string"
minOccurs="0"/>
      <xs:element name="last" type="xs:string"/>
    </xs:all>
  </xs:complexType>
</xs:element>
```

## This is OK!

```
<person-name>
  <first>Mario</first>
  <last>Alviano</last>
</person-name>
```

## This is also OK!

```
<person-name>
  <last>Alviano</last>
  <first>Mario</first>
</person-name>
```



- Complex types can have **mixed** content

## Example. Mixed content

```
<xs:element name="description">
  <xs:complexType mixed="true">
    <xs:choice minOccurs="0" maxOccurs="unbounded">
      <xs:element name="em" type="xs:string" />
      <xs:element name="br" type="xs:string" />
    </xs:choice>
  </xs:complexType>
</xs:element>

<description>
  The <em>KM</em> course of the <em>Computer Science</em>
  degree
</description>
```

- Sometimes complex types are not really complex

## Example. Empty element

```
<xs:element name="br">
  <xs:complexType />
</xs:element>
```

- Simple types, also said **derived types**, are declared by

```
<xs:simpleType>
```

- They can be restrictions, lists or unions (of simple types)

```
<xs:simpleType name="name of the simpleType"  
  final="#all or list or union or restriction">
```

- **Note:** attribute `final` forbid further derivations

- Restrictions apply one or more **facets**

- `minExclusive`, `maxExclusive`, `minInclusive`,  
`maxInclusive`, `totalDigits`, `length`, `minLength`,  
`maxLength`, `enumeration`, `whiteSpace`, `pattern`

## Example

```
<xs:simpleType>  
  <xs:restriction base="xs:string">  
    <xs:enumeration value="Home" />  
    <xs:enumeration value="Work" />  
  </xs:restriction>  
</xs:simpleType>
```

- Lists are space-separated, hence the simpleType used for validating items in the list must not allow spaces

```
<xs:list itemType="name of simpleType used for  
validating items in the list" />
```

## Example

```
<xs:simpleType name="ContactType">  
  <xs:restriction base="xs:string">  
    <xs:enumeration value="Home" />  
    <xs:enumeration value="Work" />  
  </xs:restriction>  
  
</xs:simpleType>  
<xs:simpleType name="ContactTypeList">  
  <list itemType="ContactType" />  
</xs:simpleType>
```

- Unions are declared by `<xs:union>`

```
<xs:union memberTypes="whitespace separated list  
of simpleType" />
```

### Example

```
<xs:simpleType name="UnknownString">  
  <xs:restriction base="xs:string">  
    <xs:enumeration value="Unknown" />  
  </xs:restriction>  
</xs:simpleType>  
  
<xs:simpleType name="UnknownOrFloatType">  
  <xs:union memberTypes="xs:float UnknownString" />  
</xs:simpleType>
```

# The wildcard (or jolly)

- Sometimes you just need any content

```
<xs:any minOccurs="non-negative number"  
  maxOccurs="non-negative number or unbounded"  
  namespace="allowable namespaces"  
  processContents="lax or skip or strict" />
```

- As namespace one can use

- `##any`, i.e., anything
- `##other`, i.e., different from `targetNamespace`
- `##targetNamespace`
- `##local`, i.e., unqualified names
- a space-separated list of namespaces

- As `processContent`

- `skip` if the content must not be validated
- `lax` to validate reporting warnings
- `strict` to validate reporting errors

- Attributes of a complexType are declared by `<xs:attribute>` (the syntax is similar to `<xs:element>`; the type must be simple)

```
<xs:attribute name="name of the element"
  type="global type"
  ref="global element declaration"
  form="qualified or unqualified"
  default="default value"
  fixed="fixed value"
  use="optional or prohibited or required">
```

## Example 1

```
<xs:attribute name="title">
  <xs:simpleType>
    <!-- type information -->
  </xs:simpleType>
</xs:attribute>
```

## Example 2

```
<xs:attribute name="title" type="xs:string" />
```

# Text only elements with attributes

- Text only elements are usually simpleTypes (eg. xs:string)
- Elements with attributes have complexTypes
- How to add attributes to text only elements?
- **Use** `<xs:simpleContent>` and `<xs:extension>`

## Example

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

# Groups of elements

- XML Schema allows to define groups by `<xs:group>`
- Groups can be referred more times

## Example

```
<xs:group name="NameGroup">
  <xs:sequence>
    <xs:element name="first" type="xs:string">
      <xs:element name="middle" type="xs:string" minOccurs="0"
    />
    <xs:element name="last" type="xs:string">
  </xs:sequence>
</xs:group>

<xs:complexType name="NameType">
  <xs:group ref="NameGroup" />
</xs:complexType>
```



# Groups of attributes and built-in types

- As for the elements, groups of attributes can be declared
  - The tag `<xs:attributeGroup>` is used
- There are several built-in types, i.e., types defined in the language
  - `xs:string`, `xs:normalizedString`, `xs:byte`,  
`xs:unsignedByte`, `xs:hexBinary`,  
`xs:positiveInteger`, `xs:negativeInteger`,  
`xs:int`, `xs:short`, ...
- In addition to the types defined by XML Schema, one can use types in the XML Recommendation
  - `ID`, `IDREF`, `IDREFS`, `ENTITY`, `ENTITIES`,  
`NOTATION`, `NMTOKEN`, `NMTOKENS`

# Combine more schemes

- A schema can be imported by `<xs:import>`
- `<xs:import>` must be a child of `<xs:schema>`

```
<xs:import namespace="..." schemaLocation="..." />
```

- The namespace specifies which elements to import
- Usually, imported elements are defined in namespaces different from `targetNamespace`
- To import elements in `targetNamespace` use

```
<xs:include>
```

```
<xs:include schemaLocation="..." />
```

# New feature of XML Schema 1.1

- XML Schema 1.1 allows to declare assertions

```
<xs:assert test="XPath expression" />
```

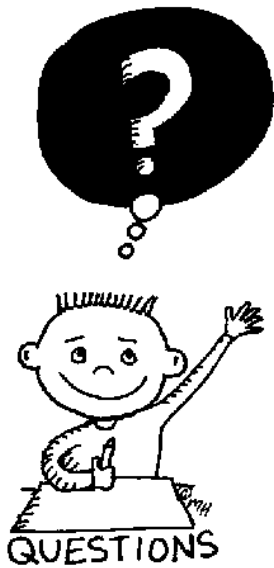
- We will see XPath in the next lecture

## Example

```
<xs:element name="sizeRange">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="minSize" type="xs:decimal" />
      <xs:element name="maxSize" type="xs:decimal" />
    </xs:sequence>
    <xs:assert test="minSize le maxSize" />
  </xs:complexType>
</xs:element>
```

- XML validation with libxml
  - `xmllint -schema XMLSchemaFile XMLfile -noout`
- XML validation with Eclipse EE
  - Right-click on the file(s) to be validated, then **Validate**

- 1 Given the specification in **personName.txt**, write an XML Schema and an XML document
- 2 Given the specification in **catalog.txt** and the documents **catalog1.xml** and **catalog2.xml**, write an XML Schema
- 3 Given the specification in **bank-accounts.txt** and the document **bank-accounts.xml**, write an XML Schema



END OF THE  
LECTURE