# XSD

André Restivo

# Index

# Introduction

# XSD

XML Schema Definition (XSD)

- W3C's recommendation to replace DTD.
- An XML based annotation language to **formally** describe the elements in an XML document.

# Schema Location

The `xsi:schemaLocation` and `xsi:noNamespaceSchemaLocation` attributes can be used in a document to provide **hints** as to the **physical** location of schema documents which can be used for **validation**.

An example XSD:

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

An example XML conforming to the XSD:

```
<?xml version="1.0" encoding="UTF-8"?>
<person xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:noNamespaceSchemaLocation="person.xsd">John Doe</person>
```

# Resources

- W3 Specification:
  - Structures: http://www.w3.org/TR/xmlschema11-1/
  - Data Types: http://www.w3.org/TR/xmlschema11-2/

# Elements

# Elements

- The `<element>` declaration contains the definition of an XML element.
- Elements have a **name** and a **type**.
- The **name** defines the **tag** used to represent the element.
- The **type** defines the possible values, children and attributes.
- Types can be either **simple** or **complex**.

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

# Simple Types

# Simple Types

- Simple types describe values **not structured** by XML markup.
- They can be used as **content** or in **attributes**.
- XSD has a library of built-in simple types.

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

```
<name>John Doe</name>
```

# Restrictions

- Simple types can be **derived** by **restriction**.
- The **base** type must be a simple type.
- The **derived** type will be a simple type.
- All simple types form a **tree**, rooted at the `anySimpleType`.

# Facets

- Restrictions are based on **facets**.
- Each restriction can have **zero or more** facets.
- The specification defines 12 different facets: `length`, `minLength`, `maxLength`, `pattern`, `enumeration`, `whiteSpace`, `maxInclusive`, `maxExclusive`, `minExclusive`, `minInclusive`, `totalDigits`, `fractionDigits`.
- Each built-in simple type allows only some facets.

```
<xs:simpleType name="personName">
  <xs:restriction base="xs:string">
    <xs:pattern value="[a-zA-Z ]+"/>
  </xs:restriction>
</xs:simpleType>

<xs:element name="name" type="personName"/>
```

```
<name>John Doe</name>
```

# Union

The `union` element defines a simple type as a **collection** of values from specified simple data types.

```
<xs:simpleType name="size">
  <xs:union memberTypes="sizebynumber sizebyname" />
</xs:simpleType>

<xs:simpleType name="sizebynumber">
  <xs:restriction base="xs:positiveInteger">
    <xs:maxInclusive value="20"/>
  </xs:restriction>
</xs:simpleType>

<xs:simpleType name="sizebyname">
  <xs:restriction base="xs:string">
    <xs:enumeration value="small"/>
    <xs:enumeration value="medium"/>
    <xs:enumeration value="large"/>
  </xs:restriction>
</xs:simpleType>

<xs:element name="size" type="size"/>
```

# List

The `list` element defines a **whitespace-separated list** of values.

```
<xs:simpleType name="sizeList">
  <xs:list itemType="size"/>
</xs:simpleType>

<xs:simpleType name="threeSizes">
  <xs:restriction base="sizeList">
    <xs:length value="3"/>
  </xs:restriction>
</xs:simpleType>

<xs:element name="sizes" type="threeSizes"/>
```

```
<sizes>12 small 15</sizes>
```

# Anonymous Types

The type of an element can be directly defined **inside** the element declaration, making it an anonymous type.

```
<xs:element name="size">
  <xs:simpleType>
    <xs:union memberTypes="sizebynumber sizebyname" />
  </xs:simpleType>
</xs:element>
```

```
<size>15</size>
```

```
<size>small</size>
```

# Complex Types

# Complex Types

- The definition of a complex type starts with the element `<complexType>`.
- Complex types can **contain** other **elements** and **attributes**.
- To define the way those child elements are allowed to appear we use the `<sequence>`, `<all>` and `<choice>` group elements.

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

```
<person>
  <name>John Doe</name>
  <email>john.doe@gmail.com</email>
</person>
```

# Sequence

The `<sequence>` element specifies that the child elements **must** appear in a specific **sequence**.

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

Not valid:

```
<person>
  <email>john.doe@gmail.com</email>
  <name>John Doe</name>
</person>
```

# All

The `<all>` element specifies that the child elements **can** appear in any order.

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

Valid:

```
<person>
  <email>john.doe@gmail.com</email>
  <name>John Doe</name>
</person>
```

# Choice

The `<choice>` element specifies that only **one** child element can occur:

```
<xs:element name="person">
    <xs:complexType>
    <xs:choice>
      <xs:element name="name" type="xs:string"/>
      <xs:element name="email" type="xs:string"/>
    </xs:choice>
    </xs:complexType>
</xs:element>
```

Valid:

```
<person>
  <name>John Doe</name>
</person>
```

```
<person>
  <email>john.doe@gmail.com</email>
</person>
```

# Occurrence Indicators

- The **minOccurs** and **maxOccurs** attributes specify the number of times each element can appear.
- The **default** value for both attributes is **1**.
- The **maxOccurs** attribute can have a value of **unbounded** (unlimited).
- The **minOccurs** attribute can have a value of **0** (optional).
- These attributes can be applied to `<element>`, `<sequence>`, `<all>` and `<choice>` elements.

? In XSD 1.0 the maxOccurs attribute of elements inside an all group was always 1. This restriction was lifted in XSD 1.1

# Example

```
<xs:element name="group">
    <xs:complexType>
    <xs:sequence maxOccurs="unbounded">
      <xs:element name="name" type="xs:string"/>
      <xs:element name="email" type="xs:string" maxOccurs="unbounded"/>
    </xs:sequence>
    </xs:complexType>
</xs:element>
```

```
<group>
  <name>John Doe</name>
  <email>john.doe@gmail.com</email>
  <email>john.doe@example.com</email>
  <name>Jane Doe</name>
  <email>jane.doe@gmail.com</email>
</group>
```

# Group Element Nesting

Group elements can be nested to create more complex groupings.

```
<xs:element name="group">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="person" maxOccurs="unbounded">
          <xs:complexType>
              <xs:sequence>
                <xs:element name="name" type="xs:string"/>
                <xs:choice>
                  <xs:element name="phone" type="xs:string"/>
                  <xs:element name="email" type="xs:string"/>
                </xs:choice>
              </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
</xs:element>
```

# Named Types

Complex types can be reused by giving them a **name** and **referencing** them from elements.

```
<xs:element name="group" type="groupType"/>

<xs:complexType name="groupType">
  <xs:sequence>
    <xs:element name="person" type="personType" maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="personType">
  <xs:sequence>
    <xs:element name="name" type="xs:string"/>
    <xs:choice>
      <xs:element name="phone" type="xs:string"/>
      <xs:element name="email" type="xs:string"/>
    </xs:choice>
  </xs:sequence>
</xs:complexType>
```

# Mixed

By **default**, complex elements **cannot** contain **text** (only other elements).

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

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

<xs:complexType name="personType" mixed="true">
  <xs:sequence>
    <xs:element name="name" type="xs:string"/>
    <xs:choice>
      <xs:element name="phone" type="xs:string"/>
      <xs:element name="email" type="xs:string"/>
    </xs:choice>
  </xs:sequence>
</xs:complexType>
```

```
<person>
  My name is <name>John Doe</name> and my email
  is <email>john.doe@gmail.com</email>.
</person>
```

# Attributes

Complex Types can have **attributes**. The type of an attribute is always a **simple type**.

```
<xs:element name="person">
  <xs:complexType>
    <xs:attribute name="id" type="xs:integer"/>
    <xs:attribute name="name" type="personName"/>
  </xs:complexType>
</xs:element>
```

Attributes can have **default** or **fixed** values.

```
<xs:attribute name="lang" type="xs:string" default="en"/>
```

By default they are **optional** but can be made **mandatory**.

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

# Any

We can use the `<any>` and `<anyAttribute>` elements to allow elements and attributes in complex types not defined in the XSD.

```
<xs:element name="person">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="name" type="xs:string"/>
      <xs:choice>
        <xs:element name="phone" type="xs:string"/>
        <xs:element name="email" type="xs:string"/>
      </xs:choice>
      <xs:any minOccurs="0"/>
    </xs:sequence>
    <xs:attribute name="id" type="xs:integer"/>
    <xs:anyAttribute/>
  </xs:complexType>
</xs:element>
```

# Extensions

- Allow the extension of `complexTypes`.
- Have to be used inside `simpleContent` and `complexContent` elements
- This allows a more **object-oriented** approach to the type system.

# Simple Content

The `simpleContent` element enables you to specify an element as containing a `simpleType` with no elements but enables you to ***restrict*** the value of the element's content or ***extend*** the element with attributes.

```
<xs:element name="name">
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="xs:string">
        <xs:attribute name="type">
         <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:enumeration value="given"/>
            <xs:enumeration value="family"/>
          </xs:restriction>
         </xs:simpleType>
        </xs:attribute>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>
```

```
<name type="given">John</name>
```

# Complex Content

Contains **extensions** or **restrictions** on a **complex type** that contains mixed content or elements only.

```
<xs:complexType name="personType">
  <xs:sequence>
    <xs:element name="name" type="xs:string"/>
    <xs:element name="email" type="xs:string"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="studentType">
  <xs:complexContent>
    <xs:extension base="personType">
      <xs:sequence>
        <xs:element name="number" type="xs:integer"/>
        <xs:element name="course" type="xs:string"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:element name="student" type="studentType" />
```

# Complex Content

```
<student>
  <name>John Doe</name>
  <email>john.doe@gmail.com</email>
  <number>1234</number>
  <course>MIEIC</course>
</student>
```

# References

# Ref

References another element that is declared **elsewhere**.

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

<xs:element name="student">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="name" type="xs:string"/>
        <xs:element ref="email"/>
      </xs:sequence>
    </xs:complexType>
</xs:element>
```

```
<student>
  <name>John Doe</name>
  <email>dasdas</email>
</student>
```

# Keys

# Key

The **key** element specifies one, or more, **attributes** or **element values** as a **key** (**unique**, **non-nullable**, and always **present**) within the containing **element** in an instance document.

- a required **name**.
- one **selector** element (**XPath** expression that specifies the set of elements across which the values specified by field must be unique)
- one or more **field** elements (**XPath** expression that specifies the values that must be unique for the set of elements specified by the selector element)

# Key [Example 1 XSD]

```
<xs:complexType name="referenceType">
  <xs:sequence>
    <xs:element name="make" type="xs:string"/>
    <xs:element name="model" type="xs:string"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="productType">
  <xs:sequence>
    <xs:element name="reference" type="referenceType"/>
    <xs:element name="price" type="xs:decimal"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="productList">
  <xs:sequence>
    <xs:element name="product" type="productType" maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>

<xs:element name="products" type="productList">
  <xs:key name="productKey">
    <xs:selector xpath="product"/>
    <xs:field xpath="reference/make">
    <xs:field xpath="reference/model">
  </xs:key>
</xs:element>
```

# Key [Example 1 XML]

```
<products>
  <product>
    <reference>
      <make>LG</make>
      <model>Nexus 5</model>
    </reference>
    <price>415.00</price>
  </product>

  <product>
    <reference>
      <make>Motorola</make>
      <model>Nexus 6</model>
    </reference>
    <price>620.00</price>
  </product>
</products>
```

# Key [Example 2 XSD]

```
<xs:complexType name="studentType">
  <xs:sequence>
    <xs:element name="name" type="xs:string"/>
  </xs:sequence>
  <xs:attribute name="number" type="xs:integer"/>
</xs:complexType>

<xs:complexType name="classType">
  <xs:sequence>
    <xs:element name="student" type="studentType" maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="schoolType">
  <xs:sequence>
    <xs:element name="class" type="classType" maxOccurs="unbounded">
      <xs:key name="studentKey">
        <xs:selector xpath="student"/>
        <xs:field xpath="@number"/>
      </xs:key>
    </xs:element>
  </xs:sequence>
</xs:complexType>
<xs:element name="school" type="schoolType"/>
```

# Key [Example 2 XML]

We can have 2 students with the same number as long as they are in different classes.

```
<school>

  <class>
    <student number="1">
      <name>John Doe</name>
    </student>

    <student number="2">
      <name>Jane Doe</name>
    </student>
  </class>

  <class>
    <student number="1">
      <name>Mary Doe</name>
    </student>

    <student number="2">
      <name>Carl Doe</name>
    </student>
  </class>
</school>
```

# Unique

A **weaker** form of **key**. Specifies that an **attribute** or **element value** (or a combination of attribute or element values) must be **unique** or **null** within the specified scope.

```
<xs:complexType name="studentType">
  <xs:sequence>
    <xs:element name="name" type="xs:string"/>
    <xs:element name="email" type="xs:string" minOccurs="0"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="schoolType">
  <xs:sequence>
    <xs:element name="student" type="studentType" maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>

<xs:element name="school" type="schoolType">
  <xs:unique name="uniqueEmail">
    <xs:selector xpath="student"/>
    <xs:field xpath="email"/>
  </xs:unique>
</xs:element>
```

# Unique

Students can't have the same email but can have no email.

```
<school>
  <student>
    <name>John Doe</name>
    <email>john.doe@gmail.com</email>
  </student>

  <student>
    <name>Jane Doe</name>
    <email>jane.doe@gmail.com</email>
  </student>

  <student>
    <name>Mary Doe</name>
  </student>
</school>
```

# KeyRef

Works like a **foreign key**. Specifies that an **attribute** or **element value** (or set of values) correspond to those of the specified **key** or **unique** element.

```
<xs:complexType name="studentType">
  <xs:sequence>
    <xs:element name="name" type="xs:string"/>
  </xs:sequence>
  <xs:attribute name="number" type="xs:integer"/>
</xs:complexType>

<xs:complexType name="classType">
  <xs:sequence>
    <xs:element name="student">
      <xs:complexType>
        <xs:attribute name="number" type="xs:integer"/>
      </xs:complexType>
    </xs:element>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="schoolType">
  <xs:sequence>
    <xs:element name="student" type="studentType" maxOccurs="unbounded"/>
    <xs:element name="class" type="classType" maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>
```

# KeyRef

## Continued

```
<xs:element name="school" type="schoolType">
  <xs:key name="studentKey">
    <xs:selector xpath="student"/>
    <xs:field xpath="@number"/>
  </xs:key>
  <xs:keyref name="studentRef" refer="studentKey">
    <xs:selector xpath="class/student"/>
    <xs:field xpath="@number"/>
  </xs:keyref>
</xs:element>
```

# KeyRef

Student numbers inside classes must exist in the school's student list.

```
<school>
  <student number="1">
    <name>John Doe</name>
  </student>
  <student number="2">
    <name>John Doe</name>
  </student>
  <class>
    <student number="1"/>
    <student number="2"/>
  </class>
</school>
```

# Namespaces

# Target Namespace

When writing XSD schemas, you **can** use the XSD `targetNamespace` attribute to specify a target namespace.

By defining a target namespace, all **elements** and **attributes** defined in the XSD belong to that namespace.

```
<xs:schema
    targetNamespace="http://www.example.com/students"
    xmlns="http://www.example.com/students"
    xmlns:xs="http://www.w3.org/2001/XMLSchema" >
  <xs:element name="student">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="name" type="xs:string"/>
        <xs:element name="email" type="xs:string"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

```
<s:student xmlns:s="http://www.example.com/students">
  <name>John Doe</name>
  <email>john.doe@gmail.com</email>
</s:student>
```

# Schema Location

To associate a XML file to its corresponding XSD we can use a `schemaLocation` attribute, if the `targetNamespace` has been defined in the XSD, or a `noNamespaceSchemaLocation`, if it hasn't.

```
<s:student
  schemaLocation="students.xsd"
  xmlns:s="http://www.example.com/students">
  <name>John Doe</name>
  <email>john.doe@gmail.com</email>
</s:student>
```

# Schema Location

When using `schemaLocation`, a list of URIs can be used.

```
<c:class
  schemaLocation="students.xsd classes.xsd"
  xmlns:s="http://www.example.com/students"
  xmlns:c="http://www.example.com/classes">
  <s:student>
      <name>John Doe</name>
      <email>john.doe@gmail.com</email>
  </s:student>
</c:class>
```

# Qualification

By default, only elements defined as root elements in the XSD must be qualified (with the namespace prefix). To change this behavior we can set the `elementFormDefault` and `attributeFormDefault` attributes to `qualified`.

```
<xs:schema
    targetNamespace="http://www.example.com/students"
    xmlns="http://www.example.com/students"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    elementFormDefault="qualified" attributeFormDefault="qualified">
  <xs:element name="student">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="name" type="xs:string"/>
        <xs:element name="email" type="xs:string"/>
      </xs:sequence>
      <xs:attribute name="id" type="xs:integer"/>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

```
<s:student xmlns:s="http://www.example.com/students" s:id="1">
  <s:name>John Doe</s:name>
  <s:email>john.doe@gmail.com</s:email>
</s:student>
```

# Include

The `include` element, imports external XSDs that share the **same namespace**.

types.xsd:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema
    targetNamespace="http://www.example.com/school"
    xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:complexType name="studentType">
        <attribute name="name" type="xs:string"/>
    </xs:complexType>
</xs:schema>
```

school.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema
    targetNamespace="http://www.example.com/school"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:s="http://www.example.com/school">
    <include schemaLocation="types.xsd"/>
    <element name="student" type="s:studentType"/>
</xs:schema>
```

# Import

The `import` element, imports external XSDs that have **different namespaces.**

types.xsd:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema
    targetNamespace="http://www.example.com/students"
    xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:complexType name="studentType">
        <attribute name="name" type="xs:string"/>
    </xs:complexType>
</xs:schema>
```

school.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema
    targetNamespace="http://www.example.com/school"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:s="http://www.example.com/students">
    <import schemaLocation="types.xsd"/>
    <element name="student" type="s:studentType"/>
</xs:schema>
```

# Validator

http://www.freeformatter.com/xml-validator-xsd.html