

XML Schema

Code: xml-xsd

Author and version

- Daniel K. Schneider
- Email: Daniel.Schneider@unige.ch
- Version: 0.6 (modified 5/12/10 by DKS)

Prerequisites

- Editing XML (being able to use a simple DTD)
- Namespaces

Availability

url: <http://tecfa.unige.ch/guides/te/files/xml-xsd.pdf>



Objectives

- Being able to cope with XSD editing
- Translating DTDs to XSD with a conversion tool
- Modifying data types of a given XSD
- Writing very simple XSD grammars

Disclaimer

- There may be typos (sorry) and mistakes (sorry again)
- Please also consult a textbook !

Acknowledgement

These slides have been prepared with the help of

- The W3C XML Schema primer: <http://www.w3.org/TR/xmlschema-0/>
- Roger Costello's extensive XML Schema tutorial: <http://www.xfront.com/>

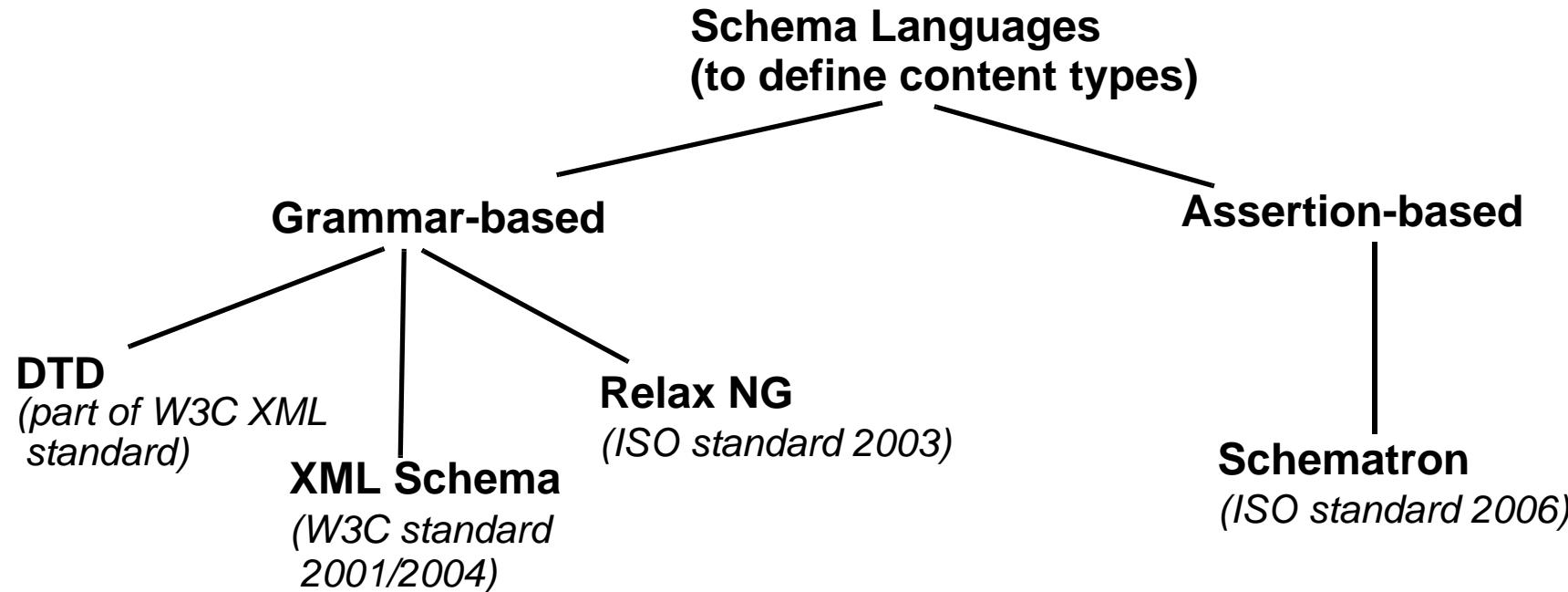
Contents

1. Introduction	5
1.1 Kinds of XML grammars	5
1.2 Feature comparison between grammar-based schemas	6
1.3 Resources	7
2. XSD bare bones	8
2.1 The structure and namespace of an XSD file	8
A.Solution 1: Give a namespace to XSD code 9	
Example 2-1:XSD definition for a simple recipe 9	
B.Solution 2: Give a namespace to target code 10	
Example 2-2:XSD definition for a simple recipe 10	
2.2 Validation	11
A.Association of XSD with XML, Solution 1 11	
B.Association of XSD with XML, Solution 2 12	
Example 2-3:XML for a simple recipe with an associated XSD (file recipe.xml) 12	
Exemple 2-4:IMS Content Packaging 1.1.4 and IMS/LOM Metadata 14	
2.3 Element definitions	15
2.4 Data types	17
2.5 Simple user-defined types	19
Exemple 2-5:Exemple "list": 19	
Exemple 2-6:restricted list of words to choose from 19	
Exemple 2-7:Restrictions on numbers 20	
2.6 Organization of elements	21
A.References vs. direct insertion (recall) 21	
B.Sequences 22	
Example 2-8:A list of ordered child elements 22	
Example 2-9:A list with one more recipe child elements 22	
Example 2-10:A list of ordered child elements 23	
Example 2-11:A list with an optional email element - repeatable 23	
C.Choice 24	
Example 2-12:Optional repeatable child elements 24	
Example 2-13:Either - or child elements 24	
D.Mixed contents (tags and text) 25	

E.Empty elements	25
2.7 Attributes	26
Example 2-14:Attribute groups	27
2.8 Value constraints	29
Example 2-15:Restrict values for an age element	29
3. From DTDs to XSDs	30
3.1 Encoding elements	30
3.2 Attribute definitions	32

1. Introduction

1.1 Kinds of XML grammars



- A grammar-based schema specifies:
 - what elements may be used in an XML document, the order of the elements, the number of occurrences of each element, etc.
 - the content and datatype of each element and attribute.
- An assertion-based schema:
 - makes assertions about the relationships that must hold between the elements and attributes in an XML instance document.

1.2 Feature comparison between grammar-based schemas

Features	DTD	XML Schema	Relax NG
Adoption	wide spread	Data-centric applications like web services	R&D mostly
Complexity of structure	Medium	Powerful (e.g. sets, element occurrence constraints)	Powerful
Data types	Little (10, mostly attribute values)	Powerful (44 + your own derived data types)	Powerful (same as XSD)
Overall complexity	low	high	medium
XML-based formalism	no	yes	yes (also a short notation)
Association with XML document	DOCTYPE declaration	Namespace declaration	No standard solution
Browser support	IE (not Firefox)	no	no
File suffix	*.dtd	*.xsd	*.rng / *.rnc
Entities	yes	no (use xinclude instead)	no

- XML Schemas were created to define more precise grammars than with DTDs, in particular one can define Data Types and more sophisticated element structures
 - DTD supports 10 datatypes; XML Schemas supports 44+ datatypes
- Relax NG was a reaction by people who didn't like this new format. It is about as powerful as XSD but not as complicated

1.3 Resources

- XML Schema (also called XSD or simply Schema) is difficult
- A good way to learn XSD is to translate your own DTDs with a tool and then study the code
- See also chapter 3. “From DTDs to XSDs” [30]

W3C websites:

url: <http://www.w3.org/XML/Schema> (W3C Overview Page)

url: <http://www.w3.org/TR/xmlschema-0/> The W3C XML Schema primer

Specifications:

url: <http://www.w3.org/TR/xmlschema-1/> XML Schema Part 1: Structures Second Edition 2004

url: <http://www.w3.org/TR/xmlschema-2/> XML Schema Part 2: Datatypes Second Edition 2004

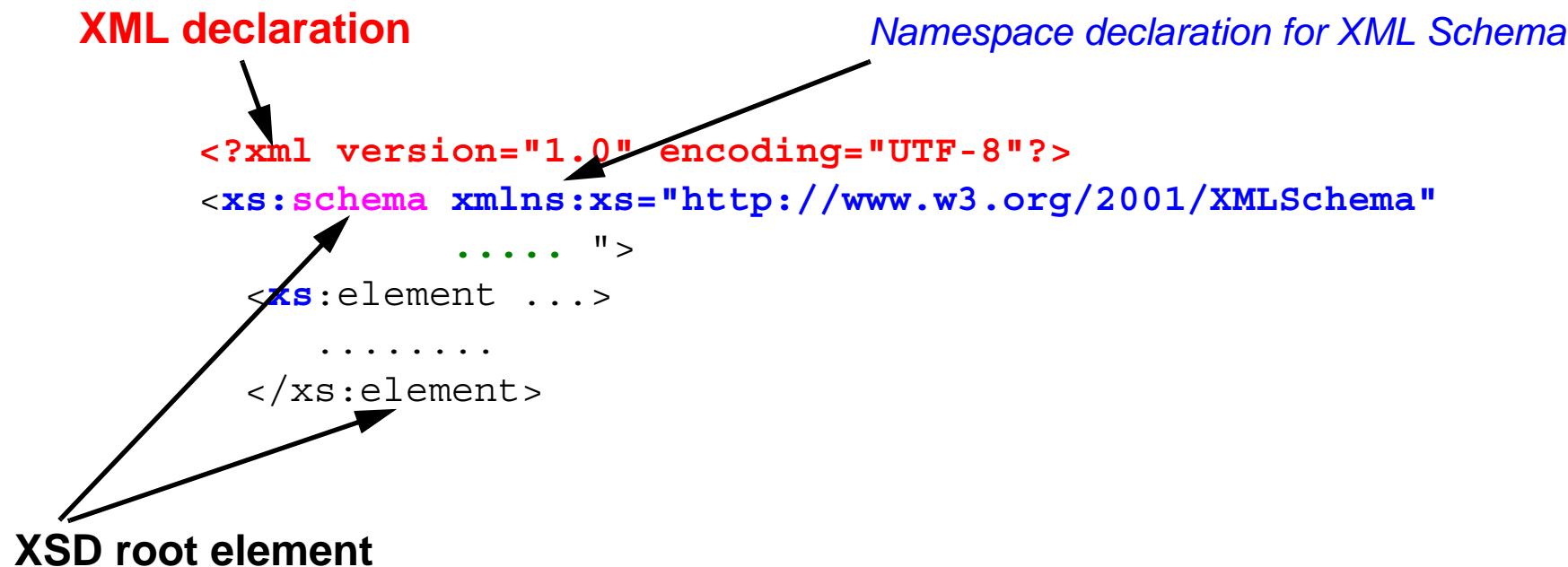
Tools:

- Exchanger XML Editor can handle XML Schema
 - Support for XSD editing
 - Validation of XSD file
 - Validation of XML against XSD
 - DTD/XSD/Relax NG translation

2. XSD bare bones

2.1 The structure and namespace of an XSD file

- As **any** XML file, an XSD file must start with an XML declaration
- Root of an XSD is `<schema> ... </schema>`
- Attributes of `schema` are used to declare certain things (see later)
- XSD makes use of namespaces since we have to make a distinction between code that belongs to XSD and code that refers to the defined elements and attributes (same principle as in XSLT).
- Complex XSD files refer to more than one "Schema" namespace (see later)



Namespaces and prefixes

- You can **either** define a prefix for the XSD elements **or** one for your own XML elements
 - See solution 1 and 2 below
 - You then can decide whether your XML elements are namespaced

A. Solution 1: Give a namespace to XSD code

- We define the **xs:** prefix for the XSD namespace
 - Doesn't matter what prefix we use (usually **xs:** but often **xsd:**)
 - **elementFormDefault="qualified"** means that your target XML files will not have namespaces

Example 2-1: XSD definition for a simple recipe

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Simple recipe Schema -->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           elementFormDefault="qualified">
  <xs:element name="list">
    <xs:complexType>
      <xs:sequence>
        <xs:element maxOccurs="unbounded" ref="recipe" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>
```

B. Solution 2: Give a namespace to target code

- We use a prefixed namespace for **our** XML elements
- Declare the XMLSchema namespace as default namespace, i.e. XSD elements will not be prefixed as in the next example

Example 2-2: XSD definition for a simple recipe

```
<schema  
    xmlns='http://www.w3.org/2000/10/XMLSchema'  
    targetNamespace='http://yourdomain.org/namespace/'  
    xmlns:t='http://yourdomain.org/namespace/'>  
  
<element name='list'>  
    <complexType>  
        <sequence>  
            <element ref='t:recipe' maxOccurs='unbounded' />  
        </sequence>  
    </complexType>  
</element>
```

2.2 Validation

- An XML document described by a XSD is called an **instance document**.
 - As with DTDs one can validate an XML against an XSD and most XML editors will allow you to do so.
 - In XML Exchanger, simple click the validate icon, then select the XSD file when asked....

A. Association of XSD with XML, Solution 1

- You must declare the **xsi:XMLSchema-instance namespace**
- The **xsi:noNamespaceSchemaLocation** attribute defines the URL of your XSD
- Warning: Make sure you get spelling and case right !!!

XML file (<http://tecfa.unige.ch/guides/xml/examples/xsd-examples/recipe-no-ns.xml>)

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<list
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="recipe-no-ns.xsd">
  <recipe> ....
</list>
```

XSD file (<http://tecfa.unige.ch/guides/xml/examples/xsd-examples/recipe-no-ns.xsd>)

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

B. Association of XSD with XML, Solution 2

- This solution is more popular since many XML standards require a namespace
1. Both XML and XSD files must contain a **namespace declaration for your domain**
The XML file must contain in addition:
 2. a declaration for the **XMLSchemainstance namespace**
 3. a **xsi:schemaLocation attribute** that tells for your namespaces where to find the XSDs
 - This attribute can have as many "namespace-URL" pairs as you like

Example 2-3: XML for a simple recipe with an associated XSD (file recipe.xml)

XML file (<http://tecfa.unige.ch/guides/xml/examples/xsd-examples/recipe.xml>)

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<list
    xmlns="http://myrecipes.org/"
    xmlns:xsi="http://www.w3.org/2001/XMLSchemainstance"
    xsi:schemaLocation="http://myrecipes.org/ recipe.xsd" >
    <recipe>
        <meta> .....</meta>
        .....
    </recipe>
</list>
```

In practical terms: You must provide something for the pink and red above

XSD file (<http://tecfa.unige.ch/guides/xml/examples/xsd-examples/recipe.xsd>)

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Simple recipe Schema -->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
             targetNamespace="http://myrecipes.org/"
             xmlns="http://myrecipes.org/"
             elementFormDefault="qualified">
    ...
</xs:schema>
```

- This XSD defines a default namespace (no prefixes) for your tags
- You should substitute <http://myrecipes.org/> by an URL of your own, preferably an URL over which you have control, e.g. a blog or a home page.

Exemple 2-4: IMS Content Packaging 1.1.4 and IMS/LOM Metadata

This XML file will use two vocabularies

```

<manifest
    xmlns="http://www.imsglobal.org/xsd/imscp_v1p1"
    xmlns:imsmd="http://www.imsglobal.org/xsd/imsmd_v1p2"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    identifier="MANIFEST-1"
    xsi:schemaLocation=
        "http://www.imsglobal.org/xsd/imscp_vlp imscp_vlp1.xsd
         http://www.imsglobal.org/xsd/imsmd_vlp2 imsmd_vlp2p2.xsd">
<metadata>
    <imsmd:lom> . . . . . </imsmd:lom>
</metadata>
<organizations default="learning_sequence_1">
    . . .

```

- imscp_v1p1 is the default namespace (no prefix)
- imsmd_v1p1 is the namespace for metadata.

Extract of ims_v1p1.xsd

```

<xsd:schema
    xmlns = "http://www.imsglobal.org/xsd/imscp_v1p1"
    targetNamespace = "http://www.imsglobal.org/xsd/imscp_v1p1"
    xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xsd = "http://www.w3.org/2001/XMLSchema"
    version = "IMS CP 1.1.4" elementFormDefault = "qualified">

```

2.3 Element definitions

- Recall that XML structure is about nested elements

<xs:element>

- Elements are defined with xs:element,

Example of a simple element without children and attributes:

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

Definition of children

- Element children can be defined in two ways:(1) with a complexType child element or (2) with a type attribute

<xs:complexType> (1)

- complexType** as a child element of **xs:element**: means "we got children or attributes to declare"

```
<xs:element name="recipe">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="meta"/>
      <xs:element minOccurs="0" ref="recipe_author"/>
      <xs:element ref="recipe_name"/>
      <xs:element ref="ingredients"/>
      <xs:element ref="directions"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

<xs:complexType> (2)

- Alternatively, one can declare a complex type by itself and then "use it" in an element declaration.

url: <http://tecfa.unige.ch/guides/xml/examples/xsd-examples/recipe2.xsd>

- Referring to a type:

```
<xs:element name="recipe" type="recipe_contents" />
```

- Defining the type:

```
<xs:complexType name="recipe_contents">
  <xs:sequence>
    <xs:element ref="meta"/>
    <xs:element minOccurs="0" ref="recipe_author"/>
    <xs:element ref="recipe_name"/>
    <xs:element ref="meal"/>
    <xs:element ref="ingredients"/>
    <xs:element ref="directions"/>
  </xs:sequence>
</xs:complexType>
```

2.4 Data types

Simple data types allow to define what kind of data elements and attributes can contain

Examples:

Simple Type	Examples (delimited by commas)	Explanation
string	Confirm this is electric	A text string
base64Binary	GpM7	Base64 encoded binary data
hexBinary	0FB7	HEX encoded binary data
integer	...-1, 0, 1, ...	
positiveInteger	1, 2, ...	
negativeInteger	... -2, -1	
nonNegativeInteger	0, 1, 2, ...	
long	-9223372036854775808, ... -1, 0, 1, ... 9223372036854775807	
decimal	-1.23, 0, 123.4, 1000.00	
float	-INF, -1E4, -0, 0, 12.78E-2, 12, INF, NaN	
boolean	true, false, 1, 0	
duration	P1Y2M3DT10H30M12.3S	1 year, 2 months, 3 days, 10 hours, 30 minutes, and 12.3 seconds

Simple Type	Examples (delimited by commas)	Explanation
dateTime	1999-05-31T13:20:00.000-05:00	May 31st 1999 at 1.20pm Eastern Standard Time
date	1999-05-31	
time	13:20:00.000, 13:20:00.000-05:00	
gYear	1999	
Name	shipTo	XML 1.0 Name type
QName	po:USAddress	XML Namespace QName
anyURI	http://www.example.com/	
language	en-GB, en-US, fr	valid values for xml:lang as defined in XML 1.0

In addition one can define list types, union types and complex types

2.5 Simple user-defined types

Exemple 2-5: Exemple "list":

XSD:

```
<xsd:element name="listOfMyInt" type="listOfMyIntType"/>
<xsd:simpleType name="listOfMyIntType">
    <xsd:list itemType="xsd:integer"/>
</xsd:simpleType>
```

XML:

```
<listOfMyInt>20003 15037 95977 95945</listOfMyInt>
```

Exemple 2-6: restricted list of words to choose from

XSD:

```
<xsd:element name="theory" type="list_theories"/>

<xsd:simpleType name="list_theories">
    <xsd:restriction base="xsd:string">
        <xsd:enumeration value="constructivism"/>
        <xsd:enumeration value="behaviorism"/>
        <xsd:enumeration value="cognitivism"/>
    </xsd:restriction>
</xsd:simpleType>
```

XML:

```
<theory>constructivism</theory>
```

Exemple 2-7: Restrictions on numbers

- This time (to change a bit) we define the type as child.

XSD:

```
<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>
```

XML:

```
<age>100</age>
```

2.6 Organization of elements

- XSD allows for quite sophisticated occurrence constraints, i.e. how child elements can be used within an element. Here we only cover a few basic design patterns

A. References vs. direct insertion (recall)

- It is best to define all elements in a flat list and then refer to these when you define how child elements are to be inserted

Defining elements within elements (not so good)

```
<xs:element name="meta">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="author" type="xs:string"/>
      <xs:element name="version" type="xs:string"/>
      <xs:element name="date" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Defining child elements with a reference (generally a better solution)

- See next Example 2-8: “A list of ordered child elements” [22]

```
<xs:sequence>
  <xs:element ref="author"/>
  ....
</xs:sequence>
```

B. Sequences

- Number of times a child element can occur is defined with minOccurs and maxOccurs attributes.

Example 2-8: A list of ordered child elements

```
<xs:element name="meta">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="author"/>
      <xs:element ref="date"/>
      <xs:element ref="version"/>
    </xs:sequence>
  </xs:complexType>

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

Example 2-9: A list with one more recipe child elements

```
<xs:element name="list">
  <xs:complexType>
    <xs:sequence>
      <xs:element maxOccurs="unbounded" ref="recipe"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Example 2-10: A list of ordered child elements

```
<xs:element name="meta">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="author"/>
      <xs:element ref="date"/>
      <xs:element ref="version"/>
    </xs:sequence>
  </xs:complexType>
```

Example 2-11: A list with an optional email element - repeatable

```
<xs:element name="person">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="name"/>
      <xs:element minOccurs="0" maxOccurs="unbounded" ref="email"/>
      <xs:element ref="link"/>
    </xs:sequence>
    <xs:attributeGroup ref="attlist.person"/>
  </xs:complexType>
</xs:element>
```

C. Choice

Example 2-12: Optional repeatable child elements

```
<xs:element name="INFOS">
  <xs:complexType>
    <xs:choice minOccurs="0" maxOccurs="unbounded">
      <xs:element ref="date"/>
      <xs:element ref="author"/>
      <xs:element ref="a"/>
    </xs:choice>
  </xs:complexType>
</xs:element>
```

Example 2-13: Either - or child elements

```
<xs:element name="ATTEMPT">
  <xs:complexType>
    <xs:choice>
      <xs:element ref="action"/>
      <xs:element ref="EPISODE"/>
    </xs:choice>
  </xs:complexType>
</xs:element>
```

D. Mixed contents (tags and text)

```
<xs:element name="para">
  <xs:complexType mixed="true">
    <xs:sequence>
      <xs:element minOccurs="0" maxOccurs="unbounded" ref="strong"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="strong" type="xs:string"/>

XML
<para> XML is <strong>so</strong> cool ! </para>
```

E. Empty elements

- Simply define an element and do not define any child elements

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

- Of course this also applies to complex elements:

- See Example 2-14: “Attribute groups” [27]

2.7 Attributes

- To declare attributes, define complexTypes.
- The **use** parameter: can be either optional, prohibited or required
 - default is "optional"
- We will not cover all possibilities here, but just demonstrate with examples

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

The above code is actually a short hand notation for:

```
<xs:element name="Name">
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="xs:string">
        <xs:attribute name="lang" type="xs:string" use="required"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>
```

XML example

```
<Name lang="English" />
```

Attribute groups

- More complex attributes are better declared with attribute groups
- Attribute groups are reusable, i.e. the equivalent of DTD's parameter entities.

Example 2-14: Attribute groups

url: <http://tecfa.unige.ch/guides/xml/examples/xsd-examples/family.xsd>

```
<xs:element name="person">
    <xs:complexType>
        <xs:attributeGroup ref="attlist.person"/>
    </xs:complexType>
</xs:element>
```

The element definition above refers to a named attribute group (defined below)

```
<xs:attributeGroup name="attlist.person">
    <xs:attribute name="name" use="required"/>

    <xs:attribute name="gender">
        <xs:simpleType>
            <xs:restriction base="xs:token">
                <xs:enumeration value="male"/>
                <xs:enumeration value="female"/>
            </xs:restriction>
        </xs:simpleType>
    </xs:attribute>
</xs:attributeGroup>
```

<!-- cont. on next slide . . . -->

```
<xs:attribute name="type" default="mother">
  <xs:simpleType>
    <xs:restriction base="xs:token">
      <xs:enumeration value="mother"/>
      <xs:enumeration value="father"/>
      <xs:enumeration value="boy"/>
      <xs:enumeration value="girl"/>
    </xs:restriction>
  </xs:simpleType>
</xs:attribute>

<xs:attribute name="id" use="required" type="xs:ID"/>
</xs:attributeGroup>
```

Valid XML fragment:

url: <http://tecfa.unige.ch/guides/xml/examples/xsd-examples/family.xml>

```
<family>
  <person name="Joe Miller" gender="male" type="father" id="I123456789"/>
  <person name="Josette Miller" type="girl" id="I123456987"/>
</family>
```

2.8 Value constraints

- One can put restraints on values that a user can enter in several ways

Example 2-15: Restrict values for an age element

```
<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>
```

3. From DTDs to XSDs

- Below we present a few typical translation patterns
- Most decent XML editors have a built-in translator that will do most of the work
 - however, generated XSD code is not necessarily the most pretty ...
 - e.g. in Exchanger XML Editor: Menu Schema -> Convert Schema

3.1 Encoding elements

Examples taken from http://www.w3.org/2000/04/schema_hack/

DTD	XML Schema
<!ELEMENT ROOT (A, B) >	<pre><element name="ROOT"> <complexType content="elementOnly"> <element ref="t:A"> <element ref="t:B"> </complexType> <element></pre>
<!ELEMENT ROOT (A B) >	<pre><element name="ROOT"> <complexType content="elementOnly"> <choice> <element ref="t:A"> <element ref="t:B"> </choice> </complexType> <element></pre>

DTD	XML Schema
<pre><!ELEMENT ROOT (A (B, C)) ></pre>	<pre><element name="ROOT"> <complexType content="elementOnly"> <choice> <element ref="t:A"> <sequence> <element ref="t:B"> <element ref="t:C"> </sequence> </choice> </complexType> <element></pre>
<pre><!ELEMENT ROOT (A?, B+, C*) ></pre>	<pre><element name="ROOT"> <complexType content="elementOnly"> <element ref="t:A" minOccurs="0"> <element ref="t:B" maxOccurs="unbounded"> <element ref="t:C" minOccurs="0" maxOccurs="unbounded"> </complexType> <element></pre>

3.2 Attribute definitions

DTD	XML Schema
<!ATTLIST ROOT a CDATA #REQUIRED>	<pre><element name="ROOT"> <complexType content="elementOnly"> <attribute name="a" type="string" use="required"/> </complexType> </element></pre>
<!ATTLIST ROOT a CDATA #IMPLIED>	<pre><element name="ROOT"> <complexType content="elementOnly"> <attribute name="a" type="string" use="optional"/> </complexType> </element></pre>
<!ATTLIST ROOT a (x y z) #REQUIRED;>	<pre><element name="ROOT"> <complexType content="elementOnly"> <attribute name="a"> <simpleType base="string"> <enumeration value="x"/> <enumeration value="y"/> <enumeration value="z"/> </simpleType> </attribute> </complexType> </element></pre>

DTD	XML Schema
<pre><!ATTLIST ROOT a CDATA #FIXED "x"></pre>	<pre><element name="ROOT"> <complexType content="elementOnly"> <attribute name="a" type="string" use="fixed" value="x"/> </complexType> </element></pre>

