



181135 VU Semistrukturierte Daten 1

XML Schema

18.10.2005

Reinhard Pichler

The title 'XML Schema' is centered within a grey, jagged-edged shape that resembles a lightning bolt or a starburst, pointing to the right.

XML Schema

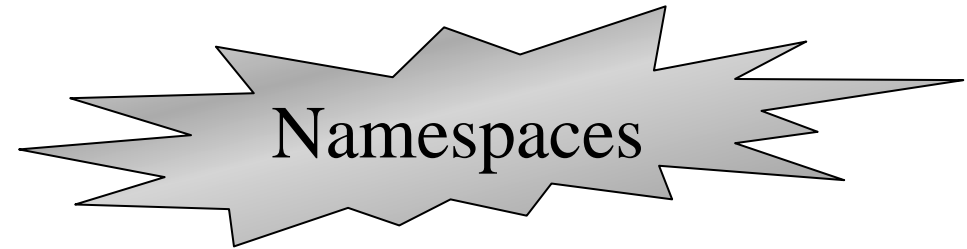
- XML Schema Description (XSD)
 - ist selbst als XML Dokument dargestellt
 - es gibt auch DTD für XSD
- W3C Recommendation 2001
 - <http://www.w3.org/TR/xmlschema-0/> (bzw. xmlschema-1, xmlschema-2)
 - XML Schema Teil 0: „Primer“ (gute Einführung!)
 - XML Schema Teil 1: Strukturen
 - XML Schema Teil 2: Datentypen
- Highlights:
 - explizite Behandlung von Namespaces
 - viele built-in Datentypen und Möglichkeit der Definition neuer Typen
 - Typisierung auch für Elementinhalt möglich
 - Wiederverwendung / abgeleitete Definitionen

XSD-Datei(en)

- XML Schema ist immer separate Entität
- Verknüpfung Instanzdokument / Schema:
 - XML Processor „kennt“ zugehöriges XML Schema
 - über Attribut in Dokumentelement:
 - schemaLocation Attribut
 - noNamespaceSchemaLocation Attribut
- Schemata können ineinander eingebettet werden:
 - <include>: mehrere Schemata mit demselben „Target NS“
 - <redefine>: einzelne Elemente können neu definiert werden
 - <import>: mehrere Schemata mit unterschiedlichem „Target NS“

Inhalt

- XML Schema:
 - Namespaces
 - Element-Deklarationen
 - Attribut-Deklarationen
 - Typendefinitionen (simple/complex types)
 - Vererbung
 - weitere XML-Schema Komponenten
- XML-Schema Validierung
 - mit IE6
 - mit Xerces



Namespaces

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="contact" type="ContactType"/>

  <xsd:complexType name="ContactType">
    <xsd:sequence>
      <xsd:element name="customer" type="USAddress"/>
      <xsd:element name="organization" type="USAddress"/>
    </xsd:sequence>
  </xsd:complexType>

  <xsd:complexType name="USAddress">
    <xsd:sequence>
      <xsd:element name="name" type="xsd:string"/>
      <xsd:element name="street" type="xsd:string"/>
      <xsd:element name="city" type="xsd:string"/>
      <xsd:element name="state" type="xsd:string"/>
      <xsd:element name="zip" type="xsd:decimal"/>
    </xsd:sequence>
    <xsd:attribute name="country" type="xsd:NMTOKEN"
      fixed="US"/>
  </xsd:complexType>
</xsd:schema>
```

Referenzierung eines Schemas ohne Target Namespace

mit noNamespaceSchemaLocation

```
<?xml version="1.0"?>
<contact
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="contact.xsd">
  <customer country="US">
    <name>B.Gates</name>
    <street>123 Main Street</street>
    <city>Redmond</city>
    <state>WA</state>
    <zip>98052</zip>
  </customer>
  <organization country="US">
    <name>Microsoft Corporation</name>
    . . . .
  </organization>
</contact>
```

XML Schema Namespaces

- Im Root Element <schema> des XML Schemas:
 - Referenz des Schema NS als Default oder qualifiziert (entsprechend werden die Komponenten von XSD mit oder ohne Präfix geschrieben):

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
```

```
oder: <schema xmlns="http://www.w3.org/2001/XMLSchema">
```

- Elemente eines Dokumentes, das mit XSD beschrieben wird, können einem oder mehreren NS angehören (aber nur 1 Target NS pro XSD-Datei!):

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.example.com/contact"
  xmlns:co="http://www.example.com/contact">
  <element name="contact" type="co:ContactType"/>
  <complexType name="ContactType">
    <sequence>
      <element name="customer" type="co:USAddress"/>
      <element name="organization" type="co:USAddress"/>
    </sequence>
  </complexType>
  <complexType name="USAddress"> ... </complexType>
</schema>
```

- Üblicherweise Präfix "xsd" (oder "xs") für den XML Schema NS:

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.example.com/contact"
  xmlns:co="http://www.example.com/contact">

  <xsd:element name="contact" type="co:ContactType"/>
  <xsd:complexType name="ContactType">
    <xsd:sequence>
      <xsd:element name="customer" type="co:USAddress"/>
      <xsd:element name="organization" type="co:USAddress"/>
    </xsd:sequence>
  </xsd:complexType>

  <xsd:complexType name="USAddress"> ... </xsd:complexType>
</xsd:schema>
```

- In diesem Fall könnte man den Target NS als Default NS definieren:

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.example.com/contact"
  xmlns="http://www.example.com/contact">

  <xsd:element name="contact" type="ContactType"/>
  <xsd:complexType name="ContactType">
    <xsd:sequence>
      <xsd:element name="customer" type="USAddress"/>
      <xsd:element name="organization" type="USAddress"/>
    </xsd:sequence>
    ....
  </xsd:complexType>
</xsd:schema>
```


XML Schema Namespaces

- Global definierte Elemente und Attribute:
 - (d.h. Subelemente von `<schema>`)
 - sind automatisch im Target NS
 - müssen daher „qualifiziert“ (d.h.: mit Präfix) referenziert werden.
- Lokale Elemente und Attribute:
 - Sind default-mäßig im leeren NS
 - Müssen daher „unqualifiziert“ (d.h.: ohne Präfix) verwendet werden.
- Änderung des Defaults:
 - Global: mit den Attributen `elementFormDefault` bzw. `attributeFormDefault` des `<schema>` Elements
 - Lokal: mit dem Attribut `form` für Elemente/Attribute
 - Wertebereich dieser Attribute: `qualified` oder `unqualified` (default)

Referenzierung eines Schemas mit Target Namespace

mit schemaLocation

```
<?xml version="1.0"?>
<co:contact
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:co="http://www.example.com/contact"
  xsi:schemaLocation="http://www.example.com/contact
    contact.xsd">
  <customer country="US">
    <name>B.Gates</name>
    <street>123 Main Street</street>
    <city>Redmond</city>
    <state>WA</state>
    <zip>98052</zip>
  </customer>
  ...
</co:contact>
```

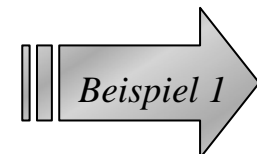
Beispiel

- Änderung des FormDefaults:

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.example.com/contact"
  xmlns:co="http://www.example.com/contact"
  elementFormDefault="qualified"
  attributeFormDefault="unqualified">
  <xsd:element name="contact" type="co:ContactType"/>
  <xsd:complexType name="ContactType">
  .....
```

- Referenzierung im Instanz-Dokument:

```
<co:contact
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:co="http://www.example.com/contact"
  xsi:schemaLocation="http://www.example.com/contact
    contact.xsd">
  <co:customer country="US">
    <co:name>B.Gates</co:name>
    <co:street>123 Main Street</co:street>
  .....
```



Einbetten von weiteren XSD-Dateien

- `<include>` für Definitionen mit demselben Target NS

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.example.com/contact"
  xmlns:co="http://www.example.com/contact">
  <include schemaLocation="http://www.example.com/customer.xsd"/>
  <include schemaLocation="product.xsd"/> .....
```

- `<redefine>`
 - um importierte simple/complex types bzw. Elementgruppen zu modifizieren
 - Ansonsten wie `<include>`

- `<import>` für Definitionen mit unterschiedlichem Target NS
(pro XSD-Datei nur 1 Target NS erlaubt!)

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.example.com/contact"
  xmlns:co="http://www.example.com/contact">
  <import
    namespace= "http://www.example.com/customer"
    schemaLocation="http://www.example.com/customer.xsd"/>
```

A grey starburst graphic with a black outline, containing the text 'Element-Deklaration' in a serif font.

Element- Deklaration

- Global definierte Elemente
 - direkt als Kind des <schema> Dokumentelementes
- Lokal definierte Elemente
 - im Kontext anderer Elemente definiert
(bzw. in complexType oder Element-Gruppe)
- Festlegen des Typs:
 - Typangabe (Attribut „type“)
 - Referenz auf global definiertes Element (Attribut „ref“)
 - Anonyme Typdefinition

Beispiele

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="veranstaltung">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="titel" type="xsd:string" maxOccurs="1"/>
        <xsd:element ref="schlagwort" maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

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

global definiertes
Element

anonymer komplexer Typ

lokal definiertes
Element

Elementreferenz

benannter simpler
Typ

Inhaltsmodelle

- beliebiges Inhaltsmodell
- leeres Inhaltsmodell
- Simpler Typ:
 - nur Zeichendaten
- Komplexer Typ:
 - enthält Elemente und/oder Attribute
 - „mixed“: Zeichendaten und Subelemente

Beliebiges Inhaltsmodell

- ohne Attribute:

```
<xsd:element name="veranstaltung" type="xsd:anyType"/>
```

- Kurzschreibweise:

```
<xsd:element name="veranstaltung"/>
```

(anyType ist der default, wenn kein Typ angegeben wird.)

- mit Attributen:

```
<xsd:element name="veranstaltung">
```

```
  <xsd:complexType>
```

```
    <xsd:complexContent>
```

```
      <xsd:extension base="xsd:anyType">
```

```
        <xsd:attribute name="jahr" type="xsd:gYear"/>
```

```
        <xsd:attribute name="vorbesprechung"  
          type="xsd:dateTime"/>
```

```
      </xsd:extension>
```

```
    <xsd:complexContent>
```

```
  </xsd:complexType>
```

```
</xsd:element>
```


Leeres Inhaltsmodell

- ohne Attribute:

```
<xsd:element name="veranstaltung">  
  <xsd:complexType/>  
</xsd:element>
```

- mit Attributen:

```
<xsd:element name="veranstaltung">  
  <xsd:complexType>  
    <xsd:attribute name="jahr" type="xsd:gYear"/>  
    <xsd:attribute name="vorbesprechung"  
      type="xsd:dateTime"/>  
  </xsd:complexType>  
</xsd:element>
```

Mögliche Attribute zur Elementdeklaration

- **type**: Ein vordefinierter oder anwenderdefinierter Elementtyp
- **ref**: Referenz auf globale Elementdeklaration zur Übernahme der dort spezifizierten Definitionen
- **name**: der unqualifizierte (lokale) Name
- **minOccurs**: Minimale Anzahl des Vorkommens. Default = 1.
- **maxOccurs**: Maximale Anzahl des Vorkommens. Default = 1.
- **id**: eindeutiger Identifier
- **fixed**: Element bekommt immer diesen Wert zugewiesen
- **default**: Element bekommt diesen Wert wenn kein anderer vorhanden
- **nillable**: Element kann leeren Inhalt haben ("Nullwerte")
- **form**: (qualified, unqualified): gibt bei lokalem Element an, ob es im Target-NS ist oder nicht (überschreibt elementFormDefault)
- **weitere Attribute** (steuern Aspekte der Vererbung):
abstract, block, final, substitutionGroup

Erläuterungen

- **fixed/default-Wert bei Elementen:**
 - Wird verwendet, wenn im Instanzdokument das Element mit leerem Inhalt vorkommt.
 - im Gegensatz zu Attributen: fixed/default-Wert wird verwendet, wenn Attribut fehlt!
- **nillable:**
 - Element darf leeren Inhalt haben (auch bei anders lautendem Inhaltsmodell)
 - Element muss in diesem Fall das Attribut `xsi:nil="true"` haben mit namespace `xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"`
 - Beispiel-Schema:

```
<xsd:element name="Bestand" type="BestandTyp" nillable="true"/>
```
 - Beispiel-Instanz:

```
<Bestand xsi:nil="true"/>
```



Komplexe Datentypen

`<xsd:complexType>`

- Nur Elemente können komplexen Datentyp haben
- kann Elemente und/oder Attribute enthalten
- Simple vs. complex content
- Angabe von Subelementen:
 - sequence
 - choice
 - all

Simple vs. Complex Content

- Simple Content
 - nur Text Inhalt
 - keine Subelemente möglich
 - Attribute möglich bei simpleContent, aber nicht in simpleType!
 - ist default bei simpleType (wenn z.B. neuer Typ auf Basis von simpleType definiert wird, wird implizit simpleContent verwendet)
- Complex Content
 - Elemente oder gemischter Inhalt
 - Attribute möglich
 - ist default bei complexType (d.h. es kann auf Element complexContent verzichtet werden)
- Complex Type mit Simple Content?
 - ja, macht Sinn – Paradebeispiel: Ein Blattelement mit einem Attribut
 - Grund: in simple type keine Attribute spezifizierbar.

Blattelemente mit Attributen

```
<xsd:complexType name="titel">
  <xsd:simpleContent>
    <xsd:extension base="xsd:string">
      <xsd:attribute name="sprache" type="xsd:NMTOKEN"/>
    </xsd:extension>
  </xsd:simpleContent>
</xsd:complexType>
```

- Blattelemente haben simplen Elementtyp (d.h.: keine Subelemente)
- Attribute können aber nur für „complexType“ definiert werden!!
- Lösung: complexType, aber simpleContent
- Benötigt Vererbung, um Elementinhalt als simpleType zu definieren.

Feste Reihenfolge der Subelemente

- **<xsd:sequence>**
 - ist der Default wenn nichts angegeben
 - Reihenfolge des Auftretens muss beachtet werden
 - wie in DTDs: (...,...)
 - in DTD kein min/maxOccurs, nur Verschachtelungen aus *,+ ,?
 - min/maxOccurs auch auf sequence selbst anwendbar
- Beispiel:

```
<xsd:complexType name="Bestand">  
  <xsd:sequence>  
    <xsd:element name="Firma" type="xsd:string"/>  
    <xsd:element name="Stichtag" minOccurs="0" type="xsd:date"/>  
    <xsd:element name="Artikel" maxOccurs="unbounded" type="artTyp"/>  
  </xsd:sequence>  
</xsd:complexType>
```

Auswahl eines Subelementes

- **<xsd:choice>**
 - wie in DTD: (..|..|..)
 - mächtiger als in DTDs, da mit min/maxOccurs verknüpfbar
 - min/maxOccurs auch auf choice selbst anwendbar
 - Verschachtelungen von choice, sequence möglich
- Beispiel:

```
<xsd:choice minOccurs="0" maxOccurs="2">  
  <xsd:element name="titel" type="xsd:string"  
    maxOccurs="2"/>  
  <xsd:element name="nummer" type="xsd:long" />  
</xsd:choice>
```

So etwas ist nicht leicht in DTD ausdrückbar!

Beliebige Reihenfolge der Subelemente

- **<xsd:all>**
 - Alle Elemente in beliebiger Ordnung
 - all kann nicht verschachtelt werden.
 - Auch Schachtelung mit Sequence und Choice ist verboten.
 - für jedes Element darin muss minOccurs 1 sein und minOccurs 0 oder 1.

- Beispiel:

```
<xsd:complexType name="Haustiere">
  <xsd:all>
    <xsd:element ref="Hunde"/>
    <xsd:element ref="Katzen"/>
    <xsd:element ref="Hasen"/>
  </xsd:all>
</xsd:complexType>
```

Gemischter Inhalt

- Gemischter Inhalt: Attribute **mixed**
 - kann beliebig mit sequence, choice, minOccurs,... verknüpft werden

- Beispiel-Schema:

```
<xsd:element name="veranstaltung" type="neuerTyp">

<xsd:complexType name="neuerTyp" mixed="true">
  <xsd:sequence>
    <xsd:element name="titel" type="xsd:string" maxOccurs="1"/>
    <xsd:element name="nummer" type="xsd:long"
      maxOccurs="unbounded"/>
  </xsd:sequence>
</xsd:complexType>
```

- Beispiel-Instanzdokument:

```
<veranstaltung>Dies ist <titel>Semistrukturierte Daten 1</titel>
mit Nummer <nummer>181135</nummer> oder <nummer>181136</nummer>,
das weiß ich nicht genau</veranstaltung>
```

A grey starburst graphic with a black outline, containing the text 'Attribut-Deklaration' in a serif font.

Attribut- Deklaration

- Global definierte Attribute
 - direkt als Kind des <schema> Dokumentelementes
- Lokal definierte Attribute
 - im Kontext eines complexType bzw. einer Attribut-Gruppe
- Festlegen des Typs:
 - Typangabe (Attribut „type“)
 - Referenz auf global definiertes Attribut (Attribut „ref“)
 - Anonyme Typdefinition

Beispiel

- auf globaler Ebene definieren
 - referenzierbar in complex types bzw. attribute group

- oder: lokal in complex type bzw. attribute group definieren

```
<xsd:attribute name="sine_tempore" default="yes" type="xsd:NMTOKEN">
```

beliebiger simple type

```
<xsd:attribute name="ort">  
    ....simple type definition....  
</xsd:attribute>
```

optionaler Default

use kann required, optional (default), prohibited sein

```
<xsd:attribute ref="sine_tempore" use="required">
```

Attributdeklaration

- immer *nach* Elementdefinitionen definieren
- haben beliebigen simplen Typ
- use: optional, required, prohibited (default ist „optional“)
- minOccurs/maxOccurs nicht sinnvoll (Attribut kommt immer 0/1-mal vor).

```
<xsd:element name="veranstaltung">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="titel" maxOccurs="1" type="titel">
      <xsd:element name="schlagwort" maxOccurs="unbounded"
        type="xsd:string" />
      <xsd:element ref="vorbesprechung" maxOccurs="1"/>
    </xsd:sequence>
    <xsd:attribute name="jahr" use="required" type="xsd:gYear"/>
  </xsd:complexType>
</xsd:element>
```

Mögliche Attribute zur Attributdeklaration

- **type**: ein vordefinierter oder anwenderdefinierter Datentyp
- **ref**: Referenz auf andere Attributdeklaration zur Übernahme der dort spezifizierten Definitionen
- **name**: der unqualifizierte (lokale) Name
- **id**: eindeutiger Identifizier
- **fixed**: Attribut bekommt immer diesen Wert zugewiesen
- **default**: Attribut bekommt diesen Wert, wenn kein anderer vorhanden ist
- **use**: (optional, required, prohibited): default ist „optional“
- **form**: (qualified, unqualified): gibt bei lokalem Attribut an, ob es im Target-NS ist oder nicht (überschreibt attributeFormDefault)

A grey starburst graphic with a black outline, containing the text 'Simple Datentypen' in a serif font.

Simple Datentypen

- Simple Datentypen:
 - nur (Zeichen-)Daten
 - keine Elemente oder Attribute enthalten
- Mögliche Klassifikationen für simple Datentypen
 - Atomar vs. aggregiert
 - atomare bestehen aus unteilbaren Werten
 - aggregierte: listen-artige und Vereinigungstypen
 - Primitiv vs. abgeleitet
 - primitiv: unabhängig von anderen Datentypen
 - abgeleitet: auf der Basis eines anderen Typs definiert
 - Vorgegeben vs. anwenderdefiniert
 - vorgegeben: 44 built-in types in XML Schema
 - Der Anwender kann abgeleitete Typen definieren

Built-In Datentypen

- primitive Datentypen:
 - string
 - decimal, float, double, boolean
 - duration, time, dateTime, date
 - gMonth, gYear, gYearMonth, gDay, gMonthDay
 - QName, anyURI, NOTATION
- abgeleitete Typen
 - (von decimal abgeleitet): integer, byte, unsignedByte, base64Binary, hexBinary, positiveInteger, negativeInteger, nonNegativeInteger, nonPositiveInteger, int, unsignedInt, long, unsignedLong, short, unsignedShort
 - (von string abgeleitet): normalizedString, token, Name, NCName, language
 - (für DTD Kompatibilität, von token abgeleitet): ID, IDREF, IDREFS, ENTITY, ENTITIES, NMTOKEN, NMTOKENS

Beispiele für zulässige Werte

boolean: true, false, 1, 0

float (single-precision, 32-bit) : -INF, -1E4, 12.78E-2, NaN (not a number)

double (double-precision 64-bit): -INF, -1E4, 12.78E-2, 12, INF, NaN

language (gültige Werte für xml:lang lt. XML 1.0): en-GB, en-US, fr

date: 1999-05-31

gMonth: --05--

gYear: 1999

gYearMonth: 1999-02

gDay: ---31

gMonthDay: --05-31

time: 13:20:00.000, 13:20:00.000-05:00 (d.h. -5h Zeitverschiebung bzgl. UCT)

duration: P1Y2M3DT10H30M12.3S (P = period, year, month, day, time, ...)



Ableitung neuer Typen ausgehend von Basistyp(en):

- **derived by restriction:**
 - auf simple/komplexe Typen anwendbar
 - Instanz des eingeschränkten Typs B ist auch Instanz des Basistyps A.
- **derived by extension:**
 - neue Elemente/Attribute hinzufügen
 - Simpler Typ wird dadurch zu komplexem Typ
- **derived by list:**
 - Liste von Elementen eines simplen Typs
- **derived by union:**
 - Vereinigung von 2 oder mehr simplen Typen

Restriktionsmöglichkeiten bei simplem Typen

- heißen auch Eigenschaften bzw. Facetten
- length, minLength, maxLength
 - Beschränkung der Stringlänge
- minInclusive, maxInclusive, minExclusive, maxExclusive
 - Beschränkung des Wertebereichs bei Zahlen
- Pattern
 - Reguläre Ausdrücke wie in Perl
- enumeration
 - Auflistung aller erlaubten Werte
- whitespace
 - preserve, replace, collapse
- totalDigits, fractionDigits
 - Dezimalstellen insgesamt bzw. nach dem Komma

Aufzählungstypen

- Erlaubt: ein Wert aus einer vordefinierten Menge

```
<xsd:simpleType name="Termine">  
  <xsd:restriction base="xsd:string">  
    <xsd:enumeration value="gestern">  
    <xsd:enumeration value="heute">  
    <xsd:enumeration value="morgen">  
  </xsd:restriction>  
</xsd:simpleType>
```

- Patterns und Enumerations werden immer mit „oder“ interpretiert wenn sie öfter vorkommen
- alle anderen Einschränkungen mit „und“

Patterns

- Typeneinschränkung basierend auf regulären Ausdrücken (*unicode regular expressions*)

```
<xsd:simpleType name="test">  
  <xsd:restriction base="xsd:string">  
    <xsd:pattern value="\p{Lu}{1,3}">  
  ...
```

Im Prinzip wie in Perl regexps: ?,+, {n,m}, \w für Wort, etc.

Unicode Buchstaben: \p{L} alle, \p{Lu} Großbuchstaben, ...

Unicode Zahlen: \p{N}alle, \p{Nd} dezimal, ...

Listentypen

- `<xsd:list itemType="...">`
- Listen können nur simple Typen enthalten
 - insbes.: Listen von Listen nicht unterstützt
- Listenelemente durch Whitespaces getrennt
- Restriktion eines Listentyps:
 - im Prinzip „wie üblich“ (enumeration, pattern, ...)
 - insbesondere: Länge der Liste
- Beispiel:

```
<xsd:simpleType name="Mitarbeiterliste">  
    <xsd:list itemType="xsd:string"/>  
</xsd:simpleType>
```
- Instanz: "Baumgartner Gottlob Herzog Koch"

Vereinigungstypen

- Kombination von mehreren simplen Typen

- entweder via `memberTypes` :

```
<xsd:simpleType name="Bsp-Union">  
    <xsd:union memberTypes="Typ1 Typ2"/>  
</xsd:simpleType>
```

- oder Typen direkt innerhalb der union definieren:

```
<xsd:simpleType name="Size">  
    <xsd:union>  
        <xsd:simpleType>  
            <xsd:restriction base='xsd:integer'>...  
        </xsd:simpleType>  
        <xsd:simpleType>  
            <xsd:restriction base='xsd:string'>...  
        </xsd:simpleType>  
    </xsd:union>  
</xsd:simpleType>
```


Beispiele für derived built-in Types

Restriktion mittels whiteSpace Facette:

```
<xsd:simpleType name="normalizedString">
  <xsd:restriction base="xsd:string">
    <xsd:whiteSpace value="replace"/>
  </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="token" id="token">
  <xsd:restriction base="xsd:normalizedString">
    <xsd:whiteSpace value="collapse"/>
  </xsd:restriction>
</xsd:simpleType>
```

Derived by List (+ minLength Facette):

```
<xsd:simpleType name="IDREFS">
  <xsd:restriction base="xsd:token">
    <xsd:simpleType>
      <xsd:list itemType="xsd:IDREF"/>
    </xsd:simpleType>
    <xsd:minLength value="1"/>
  </xsd:restriction>
</xsd:simpleType>
```

Beispiele für derived built-in Types

Restriktion mittels min/maxInclusive Facette:

```
<xsd:simpleType name="short">  
  <xsd:restriction base="xsd:int">  
    <xsd:minInclusive value="-32768"/>  
    <xsd:maxInclusive value="32767"/>  
  </xsd:restriction>  
</xsd:simpleType>
```

```
<xsd:simpleType name="positiveInteger">  
  <xsd:restriction base="xsd:nonNegativeInteger">  
    <xsd:minInclusive value="1"/>  
  </xsd:restriction>  
</xsd:simpleType>
```

Abgeleitete komplexe Typen

- Restriction:
 - Einschränkungen des Wertebereichs, z.B.:
Komponententyp einschränken,
optionale Komponenten weglassen
min/max-Occurs einschränken
 - eingeschränkter Typ ist Teilmenge des Basistyps
 - Alle Komponenten (Subelemente, Attribute), die enthalten sein sollen, müssen noch einmal explizit angegeben werden.
- Extension:
 - Hinzufügen von Elementen/Attributen
 - Basistyp kann simpler oder komplexer Typ sein.
(simpler Typ wird dadurch zu komplexem Typ)
 - Bei Erweiterung eines komplexen Typs: Man muss nur jene Komponenten angeben, die neu dazukommen.

Beispiele

- Restriction:

```
<xsd:complexType name="Bestand">
  <xsd:sequence>
    <xsd:element name="Firma" type="xsd:string"/>
    <xsd:element name="Stichtag" minOccurs="0" type="xsd:date"/>
    <xsd:element name="Artikel" maxOccurs="unbounded" type="artTyp"/>
  </xsd:sequence>
</xsd:complexType>
```

```
<xsd:complexType name="BestandNeu">
  <xsd:restriction base="Bestand">
    <xsd:sequence>
      <xsd:element name="Firma" type="xsd:string"/>
      <xsd:element name="Artikel" maxOccurs="100" type="artTyp"/>
    </xsd:sequence>
  </xsd:restriction>
</xsd:complexType>
```

- Extension (bei complex type):

```
<xsd:complexType name="Address">
  <xsd:sequence>
    <xsd:element name="name" type="xsd:string"/>
    <xsd:element name="street" type="xsd:string"/>
    <xsd:element name="city" type="xsd:string"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="US-Address">
  <xsd:extension base="Address">
    <xsd:sequence>
      <xsd:element name="state" type="US-State"/>
      <xsd:element name="zip" type="xsd:positiveInteger"/>
    </xsd:sequence>
  </xsd:extension>
</xsd:complexType>
```

- Extension (bei simple type):

```
<xsd:complexType name="length">
  <xsd:simpleContent>
    <xsd:extension base="xsd:nonNegativeInteger">
      <xsd:attribute name="unit" type="xsd:NMTOKEN"/>
    </xsd:extension>
  </xsd:simpleContent>
</xsd:complexType>
```

Substitutionsgruppen

- Definition von Elementen, die austauschbar verwendet werden können
 - Substitutionselemente müssen global definiert werden
 - Substitutionsgruppen sind nicht notwendigerweise symmetrisch (Beispiel: „place“ und „ortDetailliert“ können statt „ort“ verwendet werden aber nicht umgekehrt)

```
<xsd:element name="ort" type="xsd:string">
<xsd:element name="place" substitutionGroup="ort">
<xsd:element name="ortDetailliert" type="ortTyp"
  substitutionGroup="ort">

<xsd:complexType name="ortTyp">
  <xsd:simpleContent>
    <xsd:extension base="xsd:string">
      <xsd:attribute name="Staat" type="xsd:NMTOKEN"/>
    ...
```

Abgeleitete Typen im Instanzdokument

- Im Instanzdokument darf anstelle eines Elements A auch ein beliebiges Substitutionselement B verwendet werden.
- Außerdem darf für ein Element vom Typ A auch der Typ B verwendet werden, wenn B von A abgeleitet wurde.
- Diese Ersetzungen können durch ein block-Attribut verboten werden.
- Bei Ersetzung ohne Substitutionsgruppe muss der tatsächliche Typ des Elements im Attribut `xsi:type` angegeben werden (mit namespace `xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"`)
- Beispiel:

Schema: `<xsd:element name="shipto" type="Address">`

Instanz: `<shipto xsi:type="US-Address">
 <name>Robert Smith</name>
 <street>8 Oak Avenue</street>
 <city>San Diego</city>
 <state>CA</state>
 <zip>95819</zip>
</shipto>`

Attribute zur Steuerung der Vererbung/Substitution

- Abstrakte Elemente

- wie abstrakte Klassen: dürfen nicht im XML Dokument verwendet werden
- aber: Substitutionselemente bzw abgeleitete Elemente dürfen

```
<xsd:element name="scheitnrichtauf" type="xsd:string" abstract="true">
```

- block-Attribut

- Liste aus (extension, restriction, substitution) oder #all
- Damit können Substitutionen im Instanzdokument verboten werden.
- Außerdem gibt es ein blockDefault Attribut des schema-Elements.

```
<xsd:element name="address" type="AddressType"  
  block="extension restriction">
```

=> address-element darf im Instanzdokument nicht durch ein Element eines von "AddressType" (durch extension/restriction) abgeleiteten Typs ersetzt werden.

- **final-Attribut**

- Liste aus (*extension, restriction*) oder *#all*
- zusätzlich erlaubte Listen-Elemente bei simplen Typen: (*list, union*)
- Damit können Definitionen mittels Ableitungen verboten werden
- Außerdem gibt es ein finalDefault Attribut des schema-Elements

```
<xsd:complexType name="addressType" final="restriction">  
  <xsd:sequence>  
    ...
```

=> Von addressType darf kein Typ durch Einschränkung abgeleitet werden.

- **fixed-Attribut bei Facetten in simplen Typen:**

- Damit kann bei Facetten festgelegt werden, dass sie in abgeleiteten Typen nicht verändert werden dürfen.

```
<xsd:simpleType name='productCode'>  
  <xsd:restriction base='string'>  
    <xsd:length value='8' fixed='true' />  
  </xsd:restriction>  
</xsd:simpleType>
```



Weitere XSD-Komponenten

- Element/Attribut-Gruppen
- Any-Elemente/Attribute
- Eindeutigkeit/Schlüssel
- Dokumentation

Elementgruppen

- Elementgruppen
 - müssen global deklariert werden (xsd:group mit Attribut „name“)
 - keine Attributdeklarationen erlaubt

```
<xsd:group name="namensgruppe">  
  <xsd:sequence>  
    <xsd:element name="Vorname" type="xsd:string"/>  
    <xsd:element name="Nachname" type="xsd:string"/>  
  </xsd:sequence>  
</xsd:group>
```

- Wiederverwendbarkeit durch Referenzierung
 - in complexType oder anderer Elementgruppe
 - Referenzierung mittels xsd:group mit Attribut „ref“

```
<xsd:complexType name="Kunde">  
  <xsd:sequence>  
    <xsd:element name="Firma" type="xsd:string"/>  
    <xsd:group ref="namensgruppe">  
  </xsd:sequence>  
</xsd:complexType>
```

Attributgruppen

- Attributgruppen
 - müssen global deklariert werden (xsd:attributeGroup mit Attribut „name“)

```
<xsd:attributeGroup name="namensgruppe">  
  <xsd:attribute name="Vorname" type="xsd:string"/>  
  <xsd:attribute name="Nachname" type="xsd:string"/>  
</xsd:attributeGroup>
```

- Wiederverwendbarkeit durch Referenzierung
 - in complexType, Elementdeklaration, Attributgruppe
 - Referenzierung mittels xsd:attributeGroup mit Attribut „ref“

```
<xsd:complexType name="Kunde">  
  <xsd:sequence>  
    <xsd:element name="KundenNummer" type="xsd:integer"/>  
    <xsd:element name="Firma" type="xsd:string"/>  
  </xsd:sequence>  
  <xsd:attributeGroup ref="namensgruppe">  
</xsd:complexType>
```

Any Elemente/Attribute

- Idee:
 - beliebiges Element/Attribut aus einem bestimmten Namespace erlaubt
 - spezifiziere mit „processContents“ das Verhalten des Parsers:
 - lax: keine Prüfung des Inhalts
 - skip: Parser prüft jene Teile, für die die Deklaration verfügbar ist
 - strict: strenge Prüfung des Inhalts (ist der default).
(insbes.: Deklaration der Subelemente muss existieren)

- Beispiel:

```
<xsd:element name="htmlExample">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:any namespace="http://www.w3.org/1999/xhtml"
        minOccurs="1" maxOccurs="unbounded" processContents="skip"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Integritätsbedingungen: Unique

- Unique: Elementinhalt bzw. Attributwert muss eindeutig sein
 - Definition innerhalb eines global definierten Elements
 - XPath Selektor (4.Termin): definiert eine Menge von untergeordneten Elementen
 - XPath Feld: gibt für jedes Element der selektierten Menge ein oder mehrere Felder (= Attribute oder Elemente) an, für die die Eindeutigkeit verlangt wird. (Bei mehreren Feldern: Kombination muss eindeutig sein).

```
<Lehre>  
  <veranstaltung><name>...</name><nummer>13</nummer></veranstaltung>  
  <veranstaltung><name>...</name><nummer>17</nummer></veranstaltung>  
</Lehre>
```

```
<xsd:element name="Lehre" type="LehreTyp">  
  <xsd:unique name="einschränkung">  
    <xsd:selector xpath="veranstaltung"/>  
    <xsd:field xpath="nummer"/>  
  </xsd:unique>  
</xsd:element>
```

Integritätsbedingungen: Key/Keyref

- wie Primär- und Fremdschlüssel in rel. DB
- erweitern ID/IDREF-Konzept in DTDs
- Definition von **<key>** im Prinzip wie **<unique>**
 - d.h.: mit XPath Selektor wird eine Menge von Elementen bestimmt; mit XPath Feld(ern) wird für jedes selektierte Element ein Elementinhalt, Attributwert oder Kombination ausgewählt.
 - aber: jedes selektierte Element muss tatsächlich einen key besitzen.
- Referenzierung durch **<keyref>**
- Die Anzahl der Felder im key muss mit der Anzahl der Felder von keyref übereinstimmen.
- key/keyref müssen in einem gemeinsamen übergeordneten Element vorkommen.

Beispiel: Key/Keyref

```
<Lehre>
```

```
  <veranstaltung><name>...</name><nummer>13</nummer></veranstaltung>
```

```
  <veranstaltung><name>...</name><nummer>17</nummer></veranstaltung>
```

```
  <termine>
```

```
    <termin><datum>...<datum><lvanummer>17</lvanummer></termin>
```

```
    <termin><datum>...<datum><lvanummer>13</lvanummer></termin>
```

```
    <termin><datum>...<datum><lvanummer>17</lvanummer></termin>
```

```
  </termine>
```

```
</Lehre>
```

```
<xsd:element name="Lehre" type="LehreTyp">
```

```
  <xsd:key name="veranstaltungsnummer">
```

```
    <xsd:selector xpath="veranstaltung"/>
```

```
    <xsd:field xpath="nummer"/>
```

```
  </xsd:key>
```

```
  <xsd:keyref name="referenzname" refer="veranstaltungsnummer">
```

```
    <xsd:selector xpath="termine/termin"/>
```

```
    <xsd:field xpath="lvanummer"/>
```

```
  </xsd:keyref>
```

```
</xsd:element>
```


Dokumentation im Schema

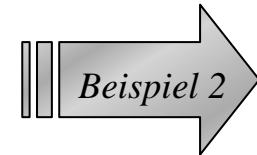
- Es gibt dafür 3 spezielle Elemente:
 - annotation: top-level element (direkt unter `<schema>`) oder auch in Element-, Attribut-, ...Definitionen
 - Subelemente von annotation: (reine Text-Elemente)
 - documentation
 - appInfo: zusätzliche Information für Schema-Processor

- Beispiel:

```
<xsd:annotation>  
  <xsd:documentation xml:lang="en">  
    contact Schema for Example.com.  
    Copyright 2000 Example.com. All rights reserved.  
  </xsd:documentation>  
</xsd:annotation>
```

Validierung

- **IE6:**
 - Einfache Datenbindung (wie bei DTD) funktioniert nicht
 - Eigenes Skript (dem XML-Dokument + Schema mitgeteilt werden) erforderlich
 - DOM-Parser
- **Xerces:**
 - Achtung: man muss für lokale Elemente `form="qualified"` einstellen, auch wenn `elementFormDefault="qualified"`
- **XML-Tester:**
 - Einfache GUI rund um Xerces
 - von Mordinyi/Mor (SSD1-Studenten im SS03)



Übungsbeispiel A

- Erstellen Sie ein XML Schema für die LVA-Homepage (laut Übungsbeispiel A des ersten Termins).
- Verwenden Sie dabei mindestens folgende XSD Sprachkonstrukte:
 - Attributgruppe
 - Benannter bzw. anonymer Typ
 - Inhaltsmodelle: beliebig, leer, simpel, komplex
 - fixed oder default Attribut
 - Simpler Typ mittels Liste abgeleitet.
- Passen Sie das XML Dokument von Übungsbeispiel A des ersten Termins an dieses Schema an und validieren Sie das XML Dokument gegen dieses XML Schema.

Übungsbeispiel B

- Erstellen Sie ein XML Schema für das XML Dokument laut Übungsbeispiel B des ersten Termins (d.h.: relationale Uni-DB).
 - Definieren Sie einen Target Namespace
 - Verwenden Sie dabei key/keyref Bedingungen für die Primärschlüssel bzw. Fremdschlüssel.
 - Verwenden Sie geeignet eingeschränkte Datentypen für die „Spalten“ der Tabellen der DB.
 - Verwenden Sie annotation/documentation Elemente zur Dokumentation (z.B. top-level und Typdeklarationen).
- Passen Sie das XML Dokument von Übungsbeispiel B des ersten Termins an dieses Schema an und validieren Sie das XML Dokument gegen dieses XML Schema.