Definition von Datentypen mit XML-Schemata

© Klaus Schild, 2003

Lernziele

- Wie wird die Element-Struktur mit einem XML-Schemata deklariert?
- Wie werden Attribute mit einem XML-Schema Deklariert?
- Was sind einfache und komplexe Datentypen?
- Wie können diese definiert werden?
- Was sind die Beschränkungen von XML-Datentypen

© Klaus Schild, 2003

Datentypen

- Datentypen beschreiben gültige Inhalte für Elemente oder Attribute.
- Formal repräsentiert ein Datentyp eine Menge von gültigen Werten, den so genannten Wertebereich.
- Es gibt zwei verschiedene Arten von Datentypen:
  - benannte Datentypen
  - anonyme Datentypen

© Klaus Schild, 2003

Anonyme vs. Benannte Datentypen

```
<xsd:element name="BookStore">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="Book" type="BookType"
                     minOccurs="1" maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:complexType>
</xsd:element>
```

- anonymer Datentyp
- lokale Definition

```
<xsd:complexType name="BookType">
    <xsd:sequence>
      …
    </xsd:sequence>
</xsd:complexType>
```

- benannter Datentyp
- globale Definition
- ist wieder verwendbar

© Klaus Schild, 2003

Element-Deklaration: 1. Möglichkeit

- Element-Inhalt kann mit einem benannten Datentyp deklariert werden, der global definiert ist:

```
<xsd:element name="BookStore">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="Book" type="BookType"
                     minOccurs="1" maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:complexType>
</xsd:element>
```

Instanz

```
<BookStore>
  <Book>
      Inhalt vom Typ BookType
  </Book>
      …
</BookStore>
```

© Klaus Schild, 2003

Element-Deklaration: 1. Möglichkeit

```
<xsd:element name="name" type="type" minOccurs="int" maxOccurs="int"/>
```

- Type: benannter Datentyp, der woanders definiert ist
- minOccurs: so oft erscheint das Element *mindestens* (nicht-negative Zahl)
- maxOccurs: so oft darf das Element *höchstens* erscheinen (nicht-negative Zahl oder „unbounded“).
- Default-Werte von minOccurs und maxOccurs jeweils 1
- Beachte: minOccurs und maxOccurs nur erlaubt, wenn Element-Deklaration Kind-Element von xsd:sequence ist!

© Klaus Schild, 2003

Element-Deklaration: 2. Möglichkeit

- Element-Inhalt kann auch mit anonymen Datentyp deklariert werden:

```
<xsd:element name="BookStore">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="Book" type="BookType"
                     minOccurs="1" maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
```

Instanz

```
<BookStore>
  <Book> … </Book>
  <Book> … </Book>
</BookStore>
```


Element-Deklaration: 2. Möglichkeit

- anonymer Datentyp ist komplex:

```
<xsd:element name="name" minOccurs="int" maxOccurs="int">
    <xsd:complexType>
        …
    </xsd:complexType>
</xsd:element>
```

- anonymer Datentyp ist einfach:

```
<xsd:element name="name" minOccurs="int" maxOccurs="int">
    <xsd:simpleType>
        …
    </xsd:simpleType>
</xsd:element>
```


Einfache vs. komplexe Datentypen

- Komplexe Datentypen (engl. *complex types*) beschreiben *strukturierten* XML-Inhalt mit Elementen oder Attributen.
- Einfache Datentypen (engl. *simple types*) beschreiben *unstrukturierten* Inhalt ohne Elemente oder Attribute (PCDATA).

Deklaration von Attributen

- werden ähnlich wie Elemente deklariert
- es sind aber nur einfache Datentypen erlaubt
- Deklaration mit benannten Datentyp, der woanders definiert ist:

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

- Deklaration mit anonymen Datentyp:

```
<xsd:attribute name= "name">
    <xsd:simpleType>
     …
    </xsd:simpleType>
</xsd:attribute>
```


Deklaration von Attributen

```
<xsd:attribute name= "name" type= "type" use="use"
        default= "value" />
```

- Ein Attribut kann als optional, obligatorisch oder unzulässig deklariert werden:
- use="optional" optional
- use="required" obligatorisch
- use="prohibited" unzulässig
- Beachte: Wenn nichts anderes angegeben, ist das Attribut optional!
- Mit default kann ein Standard-Wert für das Attribut festgelegt werden.

Primitive vs. abgeleitete Datentypen

- Abgeleitete Datentypen (engl. *derived types*) sind auf der Basis von anderen Datentypen definiert (z.B. durch Einschränkung oder Erweiterung).
- Primitive Datentypen (engl. *primitive types*) sind hingegen *nicht* von anderen Datentypen abgleitet.
- Schema der Schemata definiert 44 unterschiedliche Datentypen, die alle einfach sind.
- Unter diesen 44 einfachen Datentypen gibt es sowohl primitive, als auch abgeleitete Datentypen.

Kategorien von Datentypen

	primitiv	abgeleitet
einfach	xsd:integer	<xsd:simpleType name="longitudeType">**<xsd:restriction base="xsd:integer">** <xsd:minInclusive value="-180"/> <xsd:maxInclusive value="180"/> **</xsd:restriction>** </xsd:simpleType>
komplex	<xsd:complexType> <xsd:sequence> … </xsd:sequence> </xsd:complexType>	<xsd:complexType name="BookTypeWithID">**<xsd:complexContent>** **<xsd:extension base="BookType">** <xsd:attribute name="ID" type="xsd:token"/> **</xsd:extension>** </xsd:complexContent> </xsd:complexType>

© Klaus Schild, 2003

Einfache Datentypen

© Klaus Schild, 2003

Beispiel xsd:date

```
<xsd:complexType name="BookPublication">
    <xsd:sequence>
        <xsd:element name="Title" type="xsd:string"/>
        <xsd:element name="Author" type="xsd:string"/>
        <xsd:element name="Date" type="xsd:date"/>
        <xsd:element name="ISBN" type="xsd:string"/>
        <xsd:element name="Publisher" type="xsd:string"/>
    </xsd:sequence>
</xsd:complexType>
```

- einfacher Datentyp aus dem Schema der Schemata
- beschreibt ein Datum der Form von CCYY-MM-DD (A.D.) oder -CCYY-MM-DD (B.C.)
- 2003-07-23 ist z.B. ein gültiger Wert von xsd:date.

© Klaus Schild, 2003

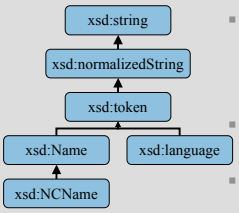
Beispiel xsd:token

```
<xsd:complexType name="BookPublication">
    <xsd:sequence>
        <xsd:element name="Title" type="xsd:string"/>
        <xsd:element name="Author" type="xsd:token"/>
        <xsd:element name="Date" type="xsd:date"/>
        <xsd:element name="ISBN" type="xsd:string"/>
        <xsd:element name="Publisher" type="xsd:string"/>
    </xsd:sequence>
</xsd:complexType>
```

- einfacher Datentyp aus dem Schema der Schemata
- von xsd:string abgeleitet

© Klaus Schild, 2003

Hierarchie einfacher Datentypen



- xsd:normalizedString: string ohne Wagenrücklauf (CR), Zeilenvorschub (LF) und Tabulator.
- xsd:token: normalizedString ohne zwei aufeinander folgende Leerzeichen und ohne Leerzeichen am Anfang und Ende.
- xsd:Name: token der Namenskonventionen von XML entspricht.
- xsd:NCName: Name ohne Namensraum-Präfix.
- xsd:language: Bezeichner für Sprache, wie z.B. „EN“

© Klaus Schild, 2003

Abgeleitete einfache Datentypen

- Es gibt drei verschiedene Möglichkeiten, einfache Datentypen abzuleiten, d.h. auf der Basis anderer einfacher Daten zu definieren:
- Einschränkung (Teilmenge): Einschränkung des Wertebereiches eines einfachen Datentyps
- Vereinigung: Vereinigung der Wertebereiche von mehreren einfachen Datentypen
- Listen: eine Liste, deren Elemente von einem bestimmten einfachen Datentyp sind

Damit können *neue* einfache Datentypen definiert werden.

© Klaus Schild, 2003

Einfache Datentypen: Einschränkung

```
<xsd:simpleType name="longitudeType">
 <xsd:restriction base="xsd:integer">
  <xsd:minInclusive value="-180"/>
  <xsd:maxInclusive value="180"/>
 </xsd:restriction>
</xsd:simpleType>
```

Einschränkungen werden konjunktiv verknüpft.

- longitudeType = { *n* in xsd:integer: *n* ≥ -180, *n* ≤ 180 }
- Für jeden einfachen Datentyp sind im Schema der Schemata zulässige Einschränkungen (engl. *constraining facets*) festgelegt.
- Z.B. sind xsd:minInclusive und xsd:maxInclusive zulässig für xsd:integer, nicht jedoch für xsd:string.

© Klaus Schild, 2003

Zulässige Einschränkungen

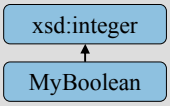
- enumeration: Zählt erlaube Werte explizit auf
- maxExclusive: <
- maxInclusive: ≤
- minExclusive: >
- minInclusive: ≥
- fractionDigits: max. Anzahl von Stellen hinter dem Komma
- length: Anzahl von Zeichen oder Listenelemente
- minlength: min. Anzahl von Zeichen oder Listenelemente
- pattern: Zeichenketten als reguläre Ausdrücke
- whiteSpace: legt fest, wie Formatierungen behandelt werden

Beachte: Für bestimmte Datentypen nur bestimmte Einschränkungen zulässig!

© Klaus Schild, 2003

Beispiel xsd:enumeration

```
<xsd:simpleType name="MyBoolean">
 <xsd:restriction base="xsd:integer">
  <xsd:enumeration value="0"/>
  <xsd:enumeration value="1"/>
 </xsd:restriction>
</xsd:simpleType>
```



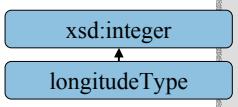
- MyBoolean = { *n* in xsd:integer: *n* = 0 *oder* *n* =1 }
- xsd:enumeration zählt alle Elemente des Wertebereiches explizit auf.
- xsd:enumeration auch für xsd:string zulässig.

© Klaus Schild, 2003

Vererbung

```
<xsd:simpleType name="longitudeType">
 <xsd:restriction base="xsd:integer">
  <xsd:minInclusive value="-180"/>
  <xsd:maxInclusive value="180"/>
 </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="PostiveLongitudeType">
 <xsd:restriction base="longitudeType">
  <xsd:minInclusive value="0"/>
 </xsd:restriction>
</xsd:simpleType>
```



Abgeleiteter Datentyp longituedType erbt zulässige Einschränkungen vom ursprünglichen Datentyp xsd:integer.

© Klaus Schild, 2003

Einfache Datentypen: Vereinigung

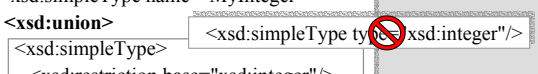
```
<xsd:simpleType name="MyInteger">
 <xsd:union>
  <xsd:simpleType>
   <xsd:restriction base="xsd:integer"/>
  </xsd:simpleType>
  <xsd:simpleType>
   <xsd:restriction base="xsd:string">
    <xsd:enumeration value="unknown"/>
   </xsd:restriction>
  </xsd:simpleType>
 </xsd:union>
</xsd:simpleType>
```

MyInteger =
xsd:integer U
{ *s* in xsd:string: *s* = unknown }

© Klaus Schild, 2003

Struktur von xsd:simpleType

```
<xsd:simpleType name="MyInteger">
 <xsd:union>
  <xsd:simpleType>
   <xsd:restriction base="xsd:integer"/>
  </xsd:simpleType>
  <xsd:simpleType>
   <xsd:restriction base="xsd:string">
    <xsd:enumeration value="unknown"/>
   </xsd:restriction>
  </xsd:simpleType>
 </xsd:union>
</xsd:simpleType>
```



Beachte: simpleType muss immer restriction, union oder list als Kind-Element haben.

© Klaus Schild, 2003

Einfache Datentypen: Liste

```
<xsd:simpleType name="IntegerList">
 <xsd:list itemType="xsd:integer"/>
</xsd:simpleType>
```

- IntergerList ist eine Liste von Integern.
- Elemente einer Liste werden durch beliebige Formatierungen (*white space*) getrennt.
- gültige Werte von IntegerList sind also:

```
108  99  205  23 0
```

```
108  99          205  23 0
```

```
108 99
  205 23 0
```

© Klaus Schild, 2003

Einfache Datentypen: Liste

```
<xsd:simpleType name="IntegerList">
 <xsd:list itemType="xsd:integer"/>
</xsd:simpleType>
```

- Beachte: IntegerList ist ein *einfacher* Datentyp, beschreibt also *unstrukturierten* Inhalt (PCDATA):

```
108  99  205  23 0
```

- Eine *strukturierte* Liste von Integern könnte hingegen so aussehen:

```
<element>108</element>
<element>99</element>
<element>205</element>
<element>23</element>
<element>0</element>
```

© Klaus Schild, 2003

Komplexe Datentypen

© Klaus Schild, 2003

Komplexe Datentypen

- Komplexe Datentypen (engl. *complex types*) beschreiben *strukturierten* XML-Inhalt.
- Es gibt drei Möglichkeiten die komplexe Datentypen zu beschreiben: Sequenz, Menge und Auswahl von Elementen.

© Klaus Schild, 2003

Komplexe Datentypen: Sequenz

```
<xsd:complexType name="BookType">
    <xsd:sequence>
      <xsd:element name="Title" type="xsd:string"/>
      <xsd:element name="Author" type="xsd:string" maxOccurs="unbounded" />
      <xsd:element name="Date" type="xsd:string"/>
      <xsd:element name="ISBN" type="xsd:string"/>
      <xsd:element name="Publisher" type="xsd:string" minOccurs="0" />
    </xsd:sequence>
</xsd:complexType>
```

gültiger Wert

```
<Title>String</Title>
<Author>String</Author>
<Author>String</Author>
<Date>String</Date>
<ISBN>String</ISBN>
```

- Reihenfolge der Elemente ist genau festgelegt.
- Jedes Elemente erscheint so häufig, wie mittels minOccurs und maxOccurs festgelegt

© Klaus Schild, 2003

Komplexe Datentypen: Menge

```
<xsd:complexType name="BookType">
    <xsd:all>
      <xsd:element name="Title" type="xsd:string"/>
      <xsd:element name="Author" type="xsd:string"/>
      <xsd:element name="Date" type="xsd:string"/>
      <xsd:element name="ISBN" type="xsd:string"/>
      <xsd:element name="Publisher" type="xsd:string"/>
    </xsd:all>
</xsd:complexType>
```

gültiger Wert

```
<Author>String</Author>
<Title>String</Title>
<Date>String</Date>
<Publisher>String</Publisher>
<ISBN>String</ISBN>
```

- BookType besteht aus einem Title-, Author-, Date-, ISBN- und Publisher-Element, wobei die Reihenfolge gleichgültig ist.

© Klaus Schild, 2003

Komplexe Datentypen: Menge

```
<xsd:complexType name="BookPublication">
    <xsd:all>
      <xsd:element name="Title" type="xsd:string"/>
      <xsd:element name="Author" type="xsd:string"/>
      <xsd:element name="Date" type="xsd:string"/>
      <xsd:element name="ISBN" type="xsd:string"/>
      <xsd:element name="Publisher" type="xsd:string" minOccurs="0"/>
    </xsd:all>
</xsd:complexType>
```

- Einzelne Elemente können mit minOccurs="0" als optional deklariert werden.
- Beachte: Außer minOccurs="0" und den Default-Werten (jeweils 1) sind *keine* anderen Werte für minOccurs und maxOccurs erlaubt.

© Klaus Schild, 2003

Komplexe Datentypen: Auswahl

```
<xsd:complexType name="PublicationType">
  <xsd:choice>
    <xsd:element name="Book" type="BookType"/>
    <xsd:element name="Article" type="ArticleType"/>
  </xsd:choice>
</xsd:complexType>
```

gültiger Wert

```
<Book>
  <Title>String</Title>
  <Author>String</Author>
  <Date>String</Date>
  <ISBN>String</ISBN>
  <Publisher>String</Publisher>
</Book>
```

- Struktur enthält *genau eines* der aufgezählten Elemente.
- PublicationType besteht entweder aus einem Book- oder einem Article-Element.

© Klaus Schild, 2003

Verschachtelungen

```
<xsd:complexType name="Publication">
  <xsd:choice>
    <xsd:sequence>
      <xsd:element name="Title" type="xsd:string"  minOccurs="0"/>
      <xsd:element name="Author" type="xsd:string"/>
      <xsd:element name="Date" type="xsd:string"/>
      <xsd:element name="ISBN" type="ISBNType"/>
      <xsd:element name="Publisher" type="xsd:string"/>
    </xsd:sequence>
    <xsd:sequence>
      <xsd:element name="Title" type="xsd:string"  minOccurs="0"/>
      <xsd:element name="Author" type="xsd:string"/>
      <xsd:element name="AppearedIn" type="ISBNType"/>
    </xsd:sequence>
  </xsd:choice>
</xsd:complexType>
```

sequence, choice und all können beliebig verschachtelt werden.

© Klaus Schild, 2003

Gemischter Inhalt

```
<Book>
  <Title>My Life and Times</Title>
  <Author>Paul McCartney</Author>
  <Date>July, 1998</Date>
  <ISBN>94303-12021-43892</ISBN>
        Dies ist unzulässiger Text…
  <Publisher>McMillin Publishing</Publisher>
</Book>
```

- Text (PCDATA) zwischen Elementen ist normalerweise *nicht* erlaubt.
- kann aber im Schema als zulässig erklärt werden

© Klaus Schild, 2003

Gemischter Inhalt

```
<xsd:complexType name="BookType" mixed="true">
    <xsd:sequence>
      <xsd:element name="Title" type="xsd:string"/>
      <xsd:element name="Author" type="xsd:string"/>
      <xsd:element name="Date" type="xsd:string"/>
      <xsd:element name="ISBN" type="xsd:string"/>
      <xsd:element name="Publisher" type="xsd:string"/>
    </xsd:sequence>
</xsd:complexType>
```

- Text (PCDATA) zwischen Elementen wird durch mixed="true" als zulässig erklärt.

© Klaus Schild, 2003

Abgeleitete komplexe Datentypen

- Es gibt zwei unterschiedliche Arten komplexe Datentypen von anderen Datentypen abzuleiten:
- Erweiterung: Ein Datentyp kann durch zusätzliche Attribute und Elemente erweitert werden.
- Einschränkung (Teilmenge): Einschränkung des Wertebereiches eines Datentyps

© Klaus Schild, 2003

Komplexe Datentypen: Erweiterung

- Ein Datentyp kann durch zusätzliche Attribute und Elemente erweitert werden.
- Es können sowohl einfache als auch komplexe Datentypen erweitert werden.
- Wird ein einfacher Datentyp erweitert, wird daraus ein komplexer Datentyp.

Basis-Datentyp (einfach oder komplex) + zusätzliche Attribute oder Elemente = erweiterter Datentyp (komplex)

© Klaus Schild, 2003

Komplexe Datentypen: Erweiterung

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

xsd:string + Attribut length = StringWithLength

Basis-Datentyp (einfach) + zusätzliches Attribut = erweiterter Datentyp (komplex)

© Klaus Schild, 2003

Unstrukturierter Inhalt + Attribut

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

- Beachte: In XML können nur Elemente Attribute haben. Unstrukturierter Inhalt (wie xsd:string) kann *kein* Attribut haben.
- Wie ist also diese Erweiterung von xsd:string durch ein Attribut zu verstehen?

© Klaus Schild, 2003

Unstrukturierter Inhalt + Attribut

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

- Datentypen sind *keine* eigenständige Einheiten, sondern beschreiben den Inhalt von Elementen oder Attributen.
- Da Attribute immer unstrukturierten Inhalt haben, beschreiben komplexe Datentypen immer den Inhalt eines Elementes.
- Die Attribute des komplexen Datentyps werden diesem Element zugeordnet.

© Klaus Schild, 2003

Beispiel

```
<xsd:element name="Abstract" type="StringWithLength"/>
<xsd:complexType name="StringWithLength">
 <xsd:simpleContent>
  <xsd:extension base="xsd:string">
   <xsd:attribute name="length" type="xsd:nonNegativeInteger"/>
  </xsd:extension>
 </xsd:simpleContent>
</xsd:complexType>
```

Instanz

```
<Abstract length="4">
  Text
</Abstract>
```

- Das Element Abstract hat einen Inhalt vom Typ StringWithLength.
- Das Attribut length von StringWithLength wird dem Element Abstract zugeordnet.

© Klaus Schild, 2003

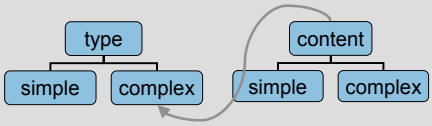
simpleContent vs. complexContent

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

- simpleContent bezeichnet *unstrukturierten* Inhalt (PCDATA) mit Attributen.
- complexContent bezeichnet dagegen *strukturierten* Inhalt (mit Elementen, mit oder ohne Attribute).
- eigentlich redundant, erleichtert aber das Parsen

© Klaus Schild, 2003

simpleContent: Etwas kompliziert!



Elemente:	nein	ja	nein	ja
Attribute:	nein	ja	ja	ja

- simpleContent und complexContent dienen zur Unterscheidung komplexer Datentypen:
- strukturierter Inhalt (complexContent) vs. unstrukturierter Inhalt mit Attributen (simpleContent)

© Klaus Schild, 2003

complexContent

```
<xsd:complexType name="BookTypeWithID">
 <xsd:complexContent>
  <xsd:extension base="BookType">
   <xsd:attribute name="ID" type="xsd:token"/>
  </xsd:extension>
 </xsd:complexContent>
</xsd:complexType>
```

BookType + Attribut ID = BookTypeWithID

Basis-Datentyp (komplex) + zusätzliches Attribut = erweiterter Datentyp (komplex)

© Klaus Schild, 2003

complexContent

```
<xsd:element name="Book" type="BookTypeWithID"/>
```

```
<xsd:complexType name="BookTypeWithID">
 <xsd:complexContent>
  <xsd:extension base="BookType">
   <xsd:attribute name="ID" type="xsd:token"/>
  </xsd:extension>
 </xsd:complexContent>
</xsd:complexType>
```

Instanz

```
<Book ID="ISBN12345567">
   <Title>String</Title>
   <Author>String</Author>
   <Author>String</Author>
   <Date>String</Date>
   <ISBN>String</ISBN>
</Book>
```

- Das Attribut ID von BookTypeWithID wird dem Element Book zugeordnet.

© Klaus Schild, 2003

Komplexe Datentypen

- Es gibt zwei unterschiedliche Arten komplexe Datentypen von anderen Datentypen abzuleiten:
- Erweiterung: Ein Datentyp kann durch zusätzliche Attribute und Elemente erweitert werden. ✓
- Einschränkung (Teilmenge): Einschränkung des Wertebereiches eines Datentyps

© Klaus Schild, 2003

Komplexe Datentypen: Einschränkung

```
<xsd:complexType name="BookTypeWithCompactID">
 <xsd:complexContent>
  <xsd:restriction base="BookTypeWithID">
   <xsd:attribute name="ID" type="xsd:unsignedShort"/>
  </xsd:restriction>
 </xsd:complexContent>
</xsd:complexType>
```

- Resultierender Datentyp darf nur gültige Werte des ursprünglichen Datentyps enthalten (echte Teilmenge)
- Eingeschränkter Datentyp erbt alle Eigenschaften des ursprünglichen Datentyps.
- XML Spy 5.2 unterstützt Spezialisierung von Attributen, aber nicht von Elementen.

© Klaus Schild, 2003

Typsubstitution

- Angenommen, ein Schema *S* leitet einen Datentyp *t'* von einem anderen Datentyp *t* ab (entweder mit xsd:extension oder xsd:restriction).
- Betrachten wir nun eine Instanz des Schemas *S*.
- An jeder Stelle in der Instanz, wo das Schema *S* den Datentyp *t* verlangt, kann auch *t'* verwendet werden.
- In diesem Fall muss aber der verwendete Datentyp *t'* mit xsi:type explizit angegeben werden.

© Klaus Schild, 2003

Beispiel

```
<xsd:complexType name="NameType">
   <xsd:sequence>
         <xsd:element name="first" type="xsd:string"/>
         <xsd:element name="middle" type="xsd:string"/>
         <xsd:element name="last" type="xsd:string"/>
   </xsd:sequence>
   <xsd:attribute name="title" type="xsd:string"/>
</xsd:complexType>
<xsd:complexType name="ExtendedNameType">
   <xsd:complexContent>
         <xsd:extension base="target:NameType">
            <xsd:attribute name="gender" type="xsd:string"/>
         </xsd:extension>
   </xsd:complexContent>
</xsd:complexType>
```

Datentyp *t*

Datentyp *t'* mit zusätzlichem Attribut gender

© Klaus Schild, 2003

Beispiel einer Typsubstitution

Schema

```
<xsd:element name="name" type="NameType">
```

Instanz

```
<name title="Mr.">
  <first>…</first>
  <middle>…</middle>
  <last>…</last>
</name>
```

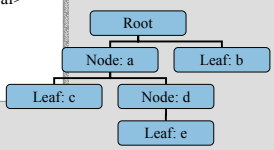
Instanz

```
<name title="Mr." gender="female" xsi:type="ExtendedNameType">
  <first>…</first>
  <middle>…</middle>
  <last>…</last>
</name>
```

© Klaus Schild, 2003

Abschließendes Beispiel: Bäume

```
<?xml version="1.0" encoding="UTF-8"?>
<Root>
  <Node name="a">
     <Leaf name="c">Text</Leaf>
     <Node name="d">
        <Leaf name="e">Text</Leaf>
     </Node>
  </Node>
  <Leaf name="b">Text</Leaf>
</Root>
```



© Klaus Schild, 2003

Ein XML-Schema für Bäume

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

  <xs:element name="Root" type="NodeType"/>

  <xs:complexType name="NodeType">
     <xs:choice maxOccurs="unbounded" minOccurs="0">
        <xs:element name="Node" type="NodeType"/>
        <xs:element name="Leaf"  type="LeafType"/>
     </xs:choice>
     <xs:attribute name="name" type="xs:string"/>
  </xs:complexType>

  <xs:complexType name="LeafType" mixed="true">
     <xs:complexContent>
        <xs:restriction base="NodeType">
              …
        </xs:restriction>
     </xs:complexContent>
  </xs:complexType>
</xs:schema>
```


Rekursive Definition eines Baumes

```
<xs:element name="Root" type="NodeType"/>

<xs:complexType name="NodeType">
     <xs:choice maxOccurs="unbounded" minOccurs="0">
        <xs:element name="Node" type="NodeType"/>
        <xs:element name="Leaf"  type="LeafType"/>
     </xs:choice>
     <xs:attribute name="name" type="xs:string"/>
</xs:complexType>
```

- Das Wurzel-Element Root ist vom Typ NodeType.
- NodeType besteht aus n ≥ 0 Elementen.
- Jedes dieser Elemente ist entweder Node vom Typ NodeType oder Leaf vom Typ LeafType.
- Jedes Elemente vom Typ NodeType kann ein Attribut name haben.

© Klaus Schild, 2003

… und eines binären Baumes

```
<xs:element name="Root" type="NodeType"/>

<xs:complexType name="NodeType">
     <xs:choice maxOccurs="2" minOccurs="0">
        <xs:element name="Node" type="NodeType"/>
        <xs:element name="Leaf"  type="LeafType"/>
     </xs:choice>
     <xs:attribute name="name" type="xs:string"/>
</xs:complexType>
```

© Klaus Schild, 2003

Definition der Blätter

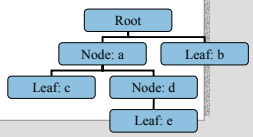
```
<xs:complexType name="LeafType" mixed="true">
    <xs:complexContent>
      <xs:restriction base="NodeType">
        <xs:choice maxOccurs="0" minOccurs="0">
          <xs:element name="Node" type="NodeType"/>
          <xs:element name="Leaf" type="LeafType"/>
        </xs:choice>
      </xs:restriction>
    </xs:complexContent>
</xs:complexType>
```

- LeafType ist eine spezielle Form von NodeType ohne Kind-Elemente, erlaubt Text außerhalb von Elementen.
- Ergebnis: LeafType = unstrukturierter Inhalt + ererbtes Attribut name.

© Klaus Schild, 2003

Eine gültige Instanz

```
<?xml version="1.0" encoding="UTF-8"?>
<Root xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:noNamespaceSchemaLocation="Trees.xsd">
  <Node name="a">
    <Leaf name="c">Text</Leaf>
    <Node name="d">
      <Leaf name="e">Text</Leaf>
    </Node>
  </Node>
  <Leaf name="b">Text</Leaf>
</Root>
```



© Klaus Schild, 2003

Ein XML-Schema für Bäume

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

  <xs:element name="Root" type="NodeType"/>

  <xs:complexType name="NodeType">
    <xs:choice maxOccurs="unbounded" minOccurs="0">
      <xs:element name="Node" type="NodeType"/>
      <xs:element name="Leaf"  type="LeafType"/>
    </xs:choice>
    <xs:attribute name="name" type="xs:string"/>
  </xs:complexType>

  <xs:complexType name="LeafType" mixed="true">
    <xs:complexContent>
      <xs:restriction base="NodeType">
          …
      </xs:restriction>
    </xs:complexContent>
  </xs:complexType>
</xs:schema>
```