

XML-Schema: Datentypen

Wie geht es weiter?

letzte Vorlesung

- Beschreibung von Dokument-Typen ✓
- Einführung von DTDs und XML-Schema anhand eines einheitlichen Beispiels ✓

heute

- XML-Schema: Definition von Datentypen

Wozu Datentypen?

```
<location>
  <latitude>
    {x ∈ Float : -90 ≤ x ≤ 90}
  </latitude>
  <longitude>
    {x ∈ Float : -180 ≤ x ≤ 180}
  </longitude>
  <uncertainty units=" {m, ft}" >
    {x ∈ Float : x ≥ 0}
  </uncertainty>
</location>
```

Datentypen:

- gültiger Inhalt von latitude, longitude, uncertainty und units
- gültiger Inhalt von location

z.B. wichtig, wenn Parameter von Prozeduren ausgetauscht werden (→ Web Services)

Was sind Datentypen?

- **Datentyp** beschreibt gültigen Inhalt von Elementen oder Attributen.
- Formal repräsentiert ein Datentyp eine Menge von gültigen Werten, den so genannten **Wertebereich**.
- in XML-Schemata zwei verschiedene Arten von Datentypen:
 - benannte Datentypen
 - anonyme Datentypen

Anonyme vs. Benannte Datentypen

```
<xsd:element name="BookStore">
  <xsd:complexType>
    Liste von Büchern
  </xsd:complexType>
</xsd:element>
```

- **anonymer Datentyp**
- **lokale Definition**

```
<xsd:complexType name="BookStore">
  Liste von Büchern
</xsd:complexType>
```

- **benannter Datentyp**
- **globale Definition**
- **wieder verwendbar**

Element-Deklaration: 1. Möglichkeit

- Element kann mit einem Datentyp deklariert werden, der woanders definiert ist:

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

```
<BookStore>
  <Book
    BookType
  </Book>
  ...
</BookStore>
```

Instanz

Element-Deklaration: 1. Möglichkeit

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

- **name**: Name des deklarierten Elementes
- **type**: 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 (oder ähnlichen Konstrukten) ist!

Element-Deklaration: 2. Möglichkeit

- Element kann auch mit anonymen Datentyp deklariert werden:

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

```
<BookStore>
  <Book> ... </Book>
  <Book> ... </Book>
</BookStore>
```

Instanz

Element-Deklaration: 2. Möglichkeit

- entweder ist anonymer Datentyp komplex:

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

- oder er ist einfach:

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

Einfache vs. komplexe Datentypen

Einfache Datentypen (*simple types*)

beschreiben *unstrukturierten* Inhalt *ohne* Elemente oder Attribute (PCDATA)

Komplexe Datentypen (*complex types*)

beschreiben *strukturierten* XML-Inhalt *mit* Elementen oder Attributen

Deklaration von Attributen

- ähnlich wie Elemente deklariert
- aber nur einfache Datentypen erlaubt
- entweder Deklaration mit Datentyp, der woanders definiert ist:

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

- oder Deklaration mit anonymen Datentyp:

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

Globale vs. lokale Deklarationen

```
<xsd:schema ...>
  <xsd:element name="root">
    <xsd:complexType>
      <xsd:sequence>
        ...
      </xsd:sequence>
      <xsd:attribute name="local-attribute" type="xsd:string"/>
    </xsd:complexType>
  </xsd:element>
  <xsd:attribute name="global-attribute" type="xsd:string"/>
</xsd:schema>
```

global: Deklaration Kind von xsd:schema

lokal: Deklaration *kein* direktes Kind von xds:schema

Deklaration von Attributen

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

- **use:** Attribut optional, obligatorisch oder unzulässig:
 - use="optional" optional
 - use="required" obligatorisch
 - use="prohibited" unzulässig
- **Beachte:** Wenn nichts anderes angegeben, ist das Attribut optional!
- **default:** Standard-Wert für das Attribut

Primitive vs. abgeleitete Datentypen

Abgeleitete Datentypen (*derived types*)

auf Basis von anderen Datentypen definiert, z.B. durch Einschränkung oder Erweiterung

Primitive Datentypen (*primitive types*)

nicht von anderen Datentypen abgeleitet

Kategorien von Datentypen

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

Einfache Datentypen

Einfache Datentypen

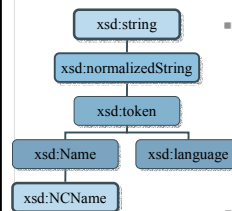
Einfache Datentypen (*simple types*)

beschreiben *unstrukturierten* Inhalt ohne Elemente oder Attribute (PCDATA)

Schema der Schemata definiert 44 einfache Datentypen: sowohl primitive, als auch abgeleitete

Hierarchie einfacher Datentypen (Auszug)

- **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 Namenskonvention von XML entspricht (mit oder ohne Präfix)
 - **xsd.NCName:** Name ohne Präfix.
 - **xsd.language:** Bezeichner für Sprache, wie z.B. „EN“



Abgeleitete einfache Datentypen

drei Möglichkeiten, eigene einfache Datentypen zu definieren:

Einschränkung (Teilmenge)

Einschränkung des Wertebereiches eines einfachen Datentyps

Vereinigung

Vereinigung der Wertebereiche von mehreren einfachen Datentypen

Listen

Liste, deren Elemente von einem bestimmten einfachen Datentyp sind

Einschränkung (Teilmenge)

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

Einschränkungen
konjunktiv
verknüpft

- longitudeType = { n aus xsd:integer: $n \geq -180, n \leq 180$ }
- Für jeden einfachen Datentyp bestimmte **zulässige Einschränkungen** (*constraining facets*) festgelegt.
- Z.B. sind xsd:minInclusive und xsd:maxInclusive zulässig für xsd:integer, nicht jedoch für xsd:string.

Zulässige Einschränkungen

- enumeration: Zählt erlaubte 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

Für bestimmte Datentypen nur bestimmte Einschränkungen zulässig!

Beispiel xsd:enumeration

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

xsd:integer

MyBoolean

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

Vererbung

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

xsd:integer

longitudeType

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

longitudeType erbt zulässige Einschränkungen vom ursprünglichen Datentyp xsd:integer.

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 aus xsd:string: s = unknown }

Struktur von xsd:simpleType

```
<xsd:simpleType name="MyInteger">
  <xsd:union>
    <xsd:simpleType type="xsd:integer" ❌
    <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.

Listen

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

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

```
108 99 205 23 0 108 99 205 23 0 108 99 205 23 0
```

Listen

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

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

```
108 99 205 23 0
```

- *Strukturierte* Liste könnte hingegen so aussehen:

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

Komplexe Datentypen

Komplexe Datentypen

Komplexe Datentypen (*complex types*)

- beschreiben *strukturierten* XML-Inhalt mit Elementen oder Attributen
- drei Möglichkeiten, komplexe Datentypen zu beschreiben:
 - Sequenz
 - Menge
 - Auswahl

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>
```

- Jedes Elemente erscheint so häufig, wie mit minOccurs und maxOccurs festgelegt.
- Reihenfolge der Elemente festgelegt

gültiger Wert

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

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>
```

- Jedes Element erscheint genau einmal.
- Reihenfolge der Elemente beliebig

gültiger Wert

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

Menge: minOccurs und maxOccurs

```
<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>
```

Einschränkungen:

- minOccurs: nur "0" oder "1"
- maxOccurs: nur "1"

Alternative ohne diese Einschränkungen: `xsd:any`

Auswahl

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

- Inhalt besteht aus *genau einem* der aufgezählten Elemente.
- hier also: entweder Book- oder Article-Element

gültiger Wert

```
<Book>
  BookType
</Book>
```

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.

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 normalerweise *nicht* erlaubt
- kann aber im Schema als zulässig erklärt werden

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>
```

- `mixed="true"`: Text (PCDATA) zwischen Kind-Elementen zulässig

Abgeleitete komplexe Datentypen



zwei unterschiedliche Arten, komplexe Datentypen von anderen Datentypen abzuleiten:

Erweiterung

Datentyp wird durch zusätzliche Attribute und Elemente erweitert.

Einschränkung (Teilmenge)

Einschränkung des Wertebereiches eines Datentyps

Erweiterung



- Datentyp kann durch zusätzliche Attribute und Elemente erweitert werden.
- Sowohl einfache als auch komplexe Datentypen können erweitert werden.
- Ergebnis auf jeden Fall ein komplexer Datentyp.

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

Beispiel



```
<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)

xsd:string + 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>
```

- Nur Elemente können Attribute haben.
- Unstrukturierter Inhalt (wie xsd:string) kann *keine* Attribute haben.

Wie ist also diese Erweiterung zu verstehen?

Aha!



```
<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 *keine* eigenständige Einheiten, sondern beschreiben Inhalt von Elementen oder Attributen
- Attribut-Werte immer unstrukturiert
- Komplexer Datentyp (wie StringWithLength) beschreibt daher immer Inhalt eines Elementes.
- Zusätzliches Attribute length wird diesem Element zugeordnet.

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 (StringWithLength)
</Abstract>

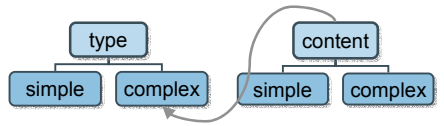
- Element Abstract hat Inhalt vom Typ StringWithLength.
- Attribut length von StringWithLength wird Element Abstract zugeordnet.

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:** *unstrukturierter* Inhalt (PCDATA) mit Attributen.
- **complexContent:** *strukturierter* Inhalt (mit Elementen).
 - wird verlangt, obwohl redundant
 - erleichtert Parsen

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)

Abgeleitete komplexe Datentypen

zwei unterschiedliche Arten, komplexe Datentypen von anderen Datentypen abzuleiten:

Erweiterung

Datentyp wird durch zusätzliche Attribute und Elemente erweitert. ✓

Einschränkung (Teilmenge)

Einschränkung des Wertebereiches eines Datentyps

Einschränkung (Teilmenge)

```
<xsd:complexType name="StringWithCompactLength">
  <xsd:simpleContent>
    <xsd:restriction base="StringWithLength">
      <xsd:attribute name="length" type="xsd:unsignedShort"/>
    </xsd:restriction>
  </xsd:simpleContent>
</xsd:complexType>
```

- echte Teilmenge: Resultierender Datentyp darf nur gültige Werte des ursprünglichen Datentyps enthalten.
- **Beachte:** XML Spy 5.2 unterstützt nur Einschränkung von Attributen.

Typsubstitution

- Voraussetzung: XML-Schema S leitet Datentyp t' von Datentyp t ab (entweder mit `xsd:extension` oder `xsd:restriction`).
- Betrachten wir nun eine Instanz von S.

Typsubstitution:

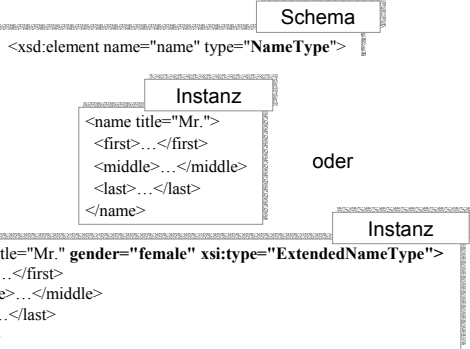
- An jeder Stelle in der Instanz, wo S den Datentyp t verlangt, kann auch t' verwendet werden.
- Verwendete Datentyp t' muss mit `xsi:type` explizit angegeben werden.

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>
```



Beispiel einer Typsubstitution



© Klaus Schild, 2004

49

Typsubstitution

- für Einschränkungen (echte Teilmengen) unproblematisch
- für Erweiterungen evtl. problematisch
- block="extension"**: Typsubstitution wird für Erweiterungen unterdrückt, z.B.:

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

© Klaus Schild, 2004

50

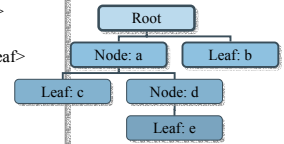
Abschließendes Beispiel

© Klaus Schild, 2004

51

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>
```



Wie sieht ein XML-Schema für Bäume dieser Art aus?

© Klaus Schild, 2004

52

Das XML-Schema

```
<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>
```

© Klaus Schild, 2004

53

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>
```

- Element Root ist vom Typ NodeType.
- NodeType besteht aus $n \geq 0$ Elementen.
- Jedes dieser Elemente ist entweder Node vom Typ NodeType oder Leaf vom Typ LeafType.
- Jedes Element vom Typ NodeType kann Attribut name haben.

© Klaus Schild, 2004

54

Binäre Bäume

```
<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>
```

einfach maxOccurs="2" statt
maxOccurs="unbounded"

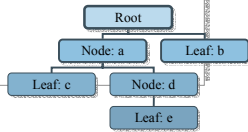
Definition der Blätter

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

Eine gültige Instanz

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



XML-Schema für Bäume

```
<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>
```

heutiges Lernziel:
dieses XML-Schema
verstehen

Wie geht es weiter?

heute

- XML-Schema: Definition von Datentypen ✓
- 16:15 Tutorium

nächste Woche

- XML-Parser