Freie Universität Berlin

Übung 3

Freie Universität Berlin

Praktische XSLT Tipps

http://www.oreilly.de/artikel/java_xslt_tips.html

Syntax

```
<xsl:attribute name="attributename" namespace="uri">

  <!-- Content:template -->

</xsl:attribute>
```

Attributes

Attribute	Value	Description
name	attributename	Required. Specifies the name of the attribute
namespace	URI	Optional. Defines the namespace URI for the attribute

Example 1

Add a source attribute to the picture element:

```
<picture>
  <xsl:attribute name="source"/>
</picture>
```

Example 2

Add a source attribute to the picture element and fill it with values from "images/name" :

```
<picture>
  <xsl:attribute name="source">
    <xsl:value-of select="images/name" />
  </xsl:attribute>
</picture>
```

```
<?xml version="1.0" encoding="iso-8859-1"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:attribute-set name="sichtbar">
  <xsl:attribute name="border">1</xsl:attribute>
  <xsl:attribute name="cellpadding">3</xsl:attribute>
  <xsl:attribute name="cellspacing">1</xsl:attribute>
</xsl:attribute-set>

<xsl:attribute-set name="unsichtbar">
  <xsl:attribute name="border">0</xsl:attribute>
  <xsl:attribute name="cellpadding">3</xsl:attribute>
  <xsl:attribute name="cellspacing">1</xsl:attribute>
</xsl:attribute-set>

<xsl:template match="/">
 <html>
 <head>
 </head>
 <body>
 <xsl:element name="table" use-attribute-sets="sichtbar">
 <xsl:for-each select="test/eintrag">
 <tr>
  <td valign="top"><b><xsl:value-of select="begriff" /></b></td>
  <td valign="top"><xsl:value-of select="definition" /></td>
 </tr>
 </xsl:for-each>
 </xsl:element>
 </body>
 </html>
</xsl:template>

</xsl:stylesheet>
```

Erfolgt die Ausgabe innerhalb von Attributen kann man anstelle eines value-ofs eine geschweifte Klammer verwenden

```
<xsl:template match="img">
  <img src="{@src}" alt="{@alt}"/>
</xsl:template>
```

Ist etwas eleganter als die längere Lösung (Ein kleiner Tipp: Sie könnten gar nicht <img name="<xsl:value-of select="."/>"/> verwenden weil Tags grundsätzlich beendet (nicht geschlossen) sein müssen wenn man eine XSL:value-of-Ausgabe anfängt. Sie müssten mit <xsl:element> arbeiten).

Freie Universität Berlin

XML Schema

Element oder Attribut?

Sollte ein Element oder ein Attribut verwendet werden?

Elemente werden zum Kapseln von Datenelementen verwendet, während Attribute in der Regel zum Bereitstellen von Begleitinformationen über ein Element, und nicht zum Kapseln der unformatierten Daten selbst dienen. Ob Sie dann ein Element oder ein Attribut auch wirklich verwenden, hängt von den Anforderungen Ihrer Anwendung ab.

Quelle: http://msdn.microsoft.com/de-de/library/7f0tkwcx(v=vs.80).aspx

Element oder Attribut?

Verwenden Sie Attribute, wenn Ihre Informationen Daten eines einfachen Typs erfordern und Folgendes gilt:

-Die Informationen erfordern einen Standard- oder einen festen Wert

-Die Informationen erfordern Daten, die Metadaten für ein vorhandenes Element sind

-Wenn die Größe der XML-Datei von Bedeutung ist, gilt: Attribute nehmen weniger Byte als Elemente in Anspruch

Quelle: http://msdn.microsoft.com/de-de/library/7f0tkwcx(v=vs.80).aspx

Element oder Attribut?

Die folgende Liste enthält die wesentlichen Unterschiede zwischen Elementen und Attributen aus Schemasicht:

-Im Schema kann definiert sein, ob die Reihenfolge von Elementen von Bedeutung ist; Attribute können jedoch in beliebiger Reihenfolge auftreten.

-Elemente können mit dem <choice>-Tag geschachtelt werden, d. h., lediglich eines der aufgelisteten Elemente kann auftreten.

- Elemente können im Gegensatz zu Attributen mehrmals vorkommen.

Quelle: http://msdn.microsoft.com/de-de/library/7f0tkwcx(v=vs.80).aspx

Type in XML Schema

Typen in XML-Schemas definieren den gültigen Datentyp, den Elemente oder Attribute enthalten können. Typen können einfach oder komplex sein. Außerdem können Typen benannt oder unbenannt sein

Quelle: http://msdn.microsoft.com/de-de/library/7f0tkwcx(v=vs.80).aspx

Type in XML Schema

Einfache Typen

Es gibt die folgenden beiden Hauptkategorien von einfachen Typen:

Integrierte Typen, die durch die XML-Schemaspezifikation des World Wide Web Consortium definiert werden – beispielsweise **string**, **boolean** und **float**.

Zu integrierten Typen gehören Primitivtypen und abgeleitete Typen.

Quelle: http://msdn.microsoft.com/de-de/library/7f0tkwcx(v=vs.80).aspx

Type in XML Schema

Primitive Datentypen werden nicht von anderen Datentypen abgeleitet. Beispielsweise ist **float** ein mathematischer Begriff, der nicht von anderen Datentypen abgeleitet wird. Weitere Informationen finden Sie unter Primitive XML-Datentypen

Beispiel:

anyURI	length, pattern, maxLength, minLength, enumeration, whiteSpace	Stellt einen in RFC 2396 definierten URI dar. Ein **anyURI**-Wert kann absolut oder relativ sein und einen optionalen Fragmentbezeichner aufweisen.

Quelle: http://msdn.microsoft.com/de-de/library/7f0tkwcx(v=vs.80).aspx

Type in XML Schema

Abgeleitete Datentypen werden anhand von vorhandenen Datentypen definiert. Beispiel: Integer ist ein besonderer Fall, der vom dezimalen Datentyp abgeleitet wird. Weitere Informationen finden Sie unter Abgeleitete XML-Datentypen.

Benutzerdefinierte einfache Typen werden von den integrierten W3C-Typen durch Anwenden von benutzerdefinierten Werten (Facets) auf Elemente abgeleitet. Weitere Informationen finden Sie unter Benutzerdefinierte einfache Typen in XML-Schemas.

Quelle: http://msdn.microsoft.com/de-de/library/7f0tkwcx(v=vs.80).aspx

Facets

Facets schränken die zulässigen Werte von einfachen Typen ein. Benutzerdefinierte einfache Typen können durch Anwenden von Facets erstellt werden. Weitere Informationen finden Sie unter Datentypfacets.
Im folgenden Beispiel wird das maxInclusive-Facet auf einen einfachen Typ mit der Bezeichnung qtyLimiter angewendet, um die zulässigen Werte des Typs positiveInteger auf Mengen zwischen 1 und 100 zu beschränken:

```
<xs:simpleType name="qtyLimiter">
        <xs:restriction base="xs:positiveInteger">
        <xs:maxInclusive value="100" />
        </xs:restriction>
</xs:simpleType>
```

Quelle: http://msdn.microsoft.com/de-de/library/7f0tkwcx(v=vs.80).aspx

Typen in XML Schema

Freie Universität Berlin

Komplexe Typen

Komplexe Typen sind Elementdefinitionen, in denen andere Elemente, Attribute und Gruppen enthalten sein können. Ein wichtiger Unterschied zwischen komplexen und einfachen Typen besteht darin, dass komplexe Typen Elemente und Attribute enthalten können, die als einfache oder komplexe Typen deklariert sind, während einfache Typen keine anderen Elemente, Attribute oder Gruppen, sondern nur Facets enthalten können. Weitere Informationen finden Sie unter Komplexe Typen in XML-Schemas.

Quelle: http://msdn.microsoft.com/de-de/library/7f0tkwcx(v=vs.80).aspx

Benannt oder unbenannt?

Datentypen und Gruppen in XML-Schemas können entweder benannt oder unbenannt sein. Wenn Sie einen Datentyp in Ihrem Schema *mehrmals verwenden* möchten, erstellen Sie einen benannten Typ, und verweisen Sie in allen Elementen, die diesen Datentyp erfordern, auf ihn. Durch benannte Typen wird Ihr Schema möglicherweise auch leichter lesbar.

Quelle: http://msdn.microsoft.com/de-de/library/7f0tkwcx(v=vs.80).aspx

Benannte Typen

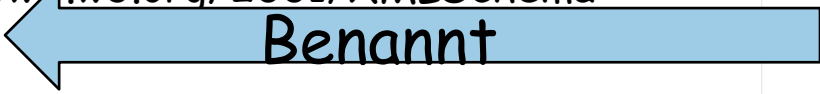
Ein benannter Typ ist ein *global deklarierter* Datentyp, der über ein **name**-Attribut verfügt. Global deklarierte Elemente sind direkt untergeordnete Elemente des <schema>-Elements und nicht in anderen Elementdefinitionen geschachtelt.

Das folgende Beispiel demonstriert die globale Deklarierung des komplexen Typs usAddress. Das Element customerInfo verwendet usAddress zweimal: als Teil des **BillTo**- und des **ShipTo**-Elements. Die Elemente name, city, ShipTo usw. werden geschachtelt in Definitionen deklariert.

Quelle: http://msdn.microsoft.com/de-de/library/7f0tkwcx(v=vs.80).aspx

Benannte Typen

```
<?xml version="1.0" encoding="utf-8" ?>
<xs:schema id="XMLSchema1" xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:complexType name="usAddress">
    <xs:sequence>
      <xs:element name="name" type="xs:string" />
      <xs:element name="street" type="xs:string" />
          ...
    </xs:sequence>
  </xs:complexType>
  <xs:element name="customerInfo">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="Name" type="xs:string" />
        <xs:element name="ShipTo" type="usAddress" />
        <xs:element name="BillTo" type="usAddress" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```



Quelle: http://msdn.microsoft.com/de-de/library/3ffy143k(v=vs.80).aspx

Unbenannte Typen

Ein unbenannter Typ ist ein Datentyp, der in der Definition eines Elements geschachtelt (inline) ist. Sie können einen unbenannten Typ verwenden, wenn der Typ nur einmal in Ihrem Schema erforderlich ist.

Im Gegensatz zum vorhergehenden Beispiel zeigt das folgende Beispiel die Definition einer Adresse als unbenannter Typ – geschachtelt im ShipTo-Element. Bei diesem Schema wissen Sie, dass Sie lediglich eine Adresse benötigen.

Quelle: http://msdn.microsoft.com/de-de/library/3ffy143k(v=vs.80).aspx

Unbenannte Typen

```
<?xml version="1.0" encoding="utf-8" ?>
<xs:schema id="XMLSchema1" xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="customerInfo">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="Name" type="xs:string" />
        <xs:element name="ShipTo">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="name" type="xs:string" />
              <xs:element name="street" type="xs:string" />
                                ...
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Unbenannt

Quelle: http://msdn.microsoft.com/de-de/library/3ffy143k(v=vs.80).aspx

Ref oder lokales Element?

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

<xs:element name="note">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="to" type="xs:string"/>
      <xs:element name="from" type="xs:string"/>
      <xs:element name="heading" type="xs:string"/>
      <xs:element name="body" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

</xs:schema>
```

Quelle: http://msdn.microsoft.com/de-de/library/3ffy143k(v=vs.80).aspx

Ref oder lokales Element?

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

<xs:element name="note">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="to" type="xs:string"/>
      <xs:element name="from" type="xs:string"/>
      <xs:element name="heading" type="xs:string"/>
      <xs:element name="body" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

</xs:schema>
```

Lokales Element

Quelle: http://msdn.microsoft.com/de-de/library/3ffy143k(v=vs.80).aspx

Ref oder lokales Element?

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:element name="note">
  <xs:complexType>
    <xs:sequence>
    <xs:element ref="to"/>
    <xs:element ref="from"/>
    <xs:element ref="heading"/>
    <xs:element ref="body"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="to" type="xs:string"/>
<xs:element name="from" type="xs:string"/>
<xs:element name="heading" type="xs:string"/>
<xs:element name="body" type="xs:string"/>

</xs:schema>
```

Referenz auf globales Element

Wiederverwendung ist toll

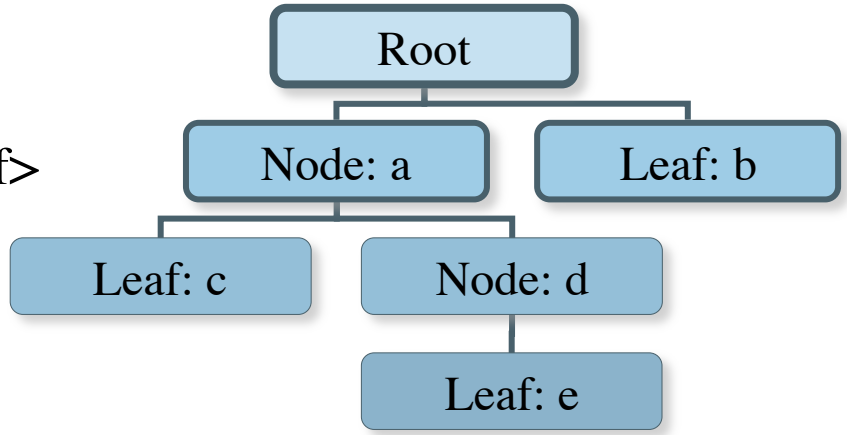
Quelle: http://www.w3schools.com/schema/el_element.asp

Freie Universität Berlin

XML-Schema für Bäume

Bäume in XML kodiert

```
<?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>
```



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

Anforderungen

```
<?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">
      <a>…</a>Text<b>…</b>
  </Leaf>
</Root>
```

- Root kann auch leer sein.
- Blätter können beliebigen XML-Inhalt haben.
- Root, Node und Leaf ohne Namensraum

Grundstruktur des XML-Schemas

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

- Beachte: kein Ziel-Namensraum
- durchaus erlaubt
- manchmal sogar sinnvoll!

Warum kein Ziel-Namensraum?

- globale Elemente des XML-Schemas:

  in Instanz immer Zielnamensraum zugeordnet

globale Elemente:

- Element-Deklaration direktes Kind-Element von xsd:schema

lokale Elemente:

- alle anderen Element-Deklarationen
- analoge Definitionen für Attribute und Typ-Definitionen
- Wurzel-Element der Instanz muss immer globales Element im XML-Schema sein

Beispiel

```
<?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:sequence minOccurs="0" maxOccurs="unbounded">
      <xs:choice>
        <xs:element name="Node" type="INodeType"/>
        <xs:element name="Leaf" type="LeafType"/>
      </xs:choice>
    </xs:sequence>
    <xs:attribute name="name" type="xs:string"/>
  </xs:complexType>
        …
</xs:schema>
```

global

lokal

Lokales Element ⇨ globales Element

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="Root" type="NodeType"/>
  <xs:element name="Node" type="INodeType"/>
  <xs:element name="Leaf" type="LeafType"/>
  <xs:complexType name="NodeType">
    <xs:sequence minOccurs="0" maxOccurs="unbounded">
      <xs:choice>
        <xs:element ref="Node"/>
        <xs:element ref="Leaf"/>
      </xs:choice>
    </xs:sequence>
    <xs:attribute name="name" type="xs:string"/>
  </xs:complexType>
        …
</xs:schema>
```

global

lokal

ElementFormDefault

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
                    targetNamespace="http://www.trees.org"
                    elementFormDefault="unqualified">
  <xs:element name="Root" type="NodeType"/>
        …
</xs:schema>
```

- qualified: Lokale Elemente der Instanz müssen namensraumeingeschränkt sein.
- unqualified: Lokale Elemente dürfen nicht namensraumeingeschränkt sein (Standard-Wert).

Beachte: ElementFormDefault betrifft nur lokalen Elemente, nicht die globalen!

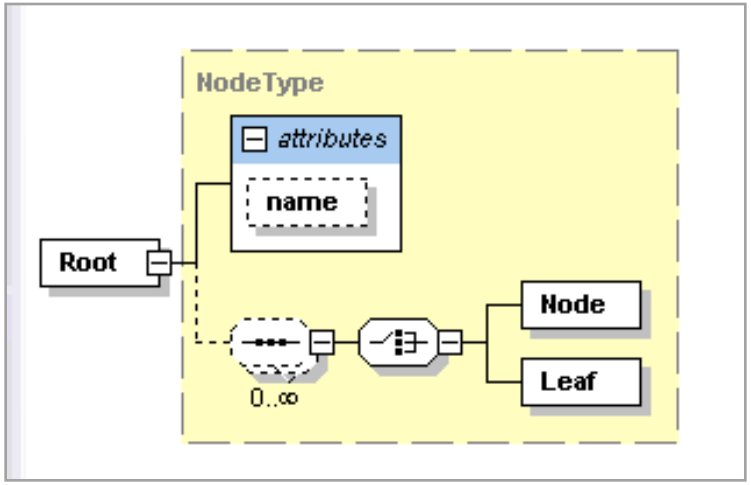
Aha!

Freie Universität Berlin

- Ziel-Namensraum enthält alle
  - globalen Elemente
  - globalen Attribute
  - globalen Typ-Definitionen
- Instanzen: Elemente und Attribute des Ziel-Namensraumes immer namensraumeingeschränkt
- elementFormDefault betrifft nur lokale Elemente
- wenn kein Element namensraumeingeschränkt sein soll: kein Zielnamensraum angegeben!

Deklaration von Root

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



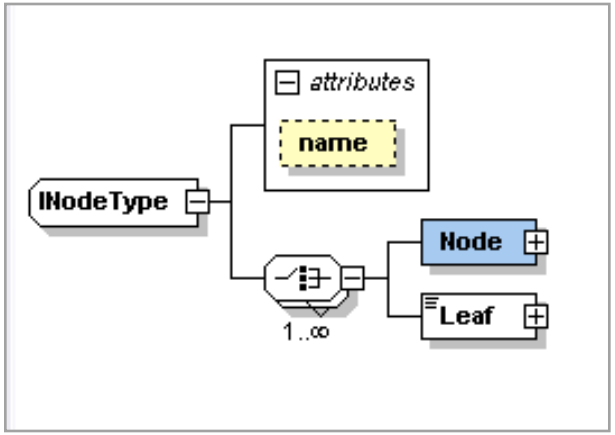
Abkürzende Schreibweise

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

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

Datentyp INodeType: Zwischenknoten

```
<xs:complexType name="INodeType">
  <xs:complexContent>
    <xs:restriction base="NodeType">
      <xs:choice minOccurs="1" maxOccurs="unbounded">
        <xs:element name="Node" type="INodeType"/>
        <xs:element name="Leaf" type="LeafType"/>
      </xs:choice>
    </xs:restriction>
  </xs:complexContent>
</xs:complexType>
```



- INodeType spezieller NodeType
- Einschränkung: minOccurs="1" statt minOccurs="0"
- Beachte: INodeType erbt von NodeType Attribut name

Binäre Bäume

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

- Beachte: diese Änderung reicht nicht aus!

Warum reicht das nicht?

```
<xs:complexType name="NodeType">
    <xs:sequence minOccurs="0" maxOccurs="2">
      <xs:choice>
        <xs:element name="Node" type="INodeType"/>
        <xs:element name="Leaf" type="LeafType"/>
      </xs:ch
    </xs:sequ
    <xs:attrib
</xs:complex
```

```
<xs:complexType name="INodeType">
  <xs:complexContent>
    <xs:restriction base="NodeType">
      <xs:choice minOccurs="1" maxOccurs="unbounded">
        <xs:element name="Node" type="INodeType"/>
        <xs:element name="Leaf" type="LeafType"/>
      </xs:choice>
    </xs:restriction>
  </xs:complexContent>
</xs:complexType>
```

INodeType keine Untermenge von NodeType

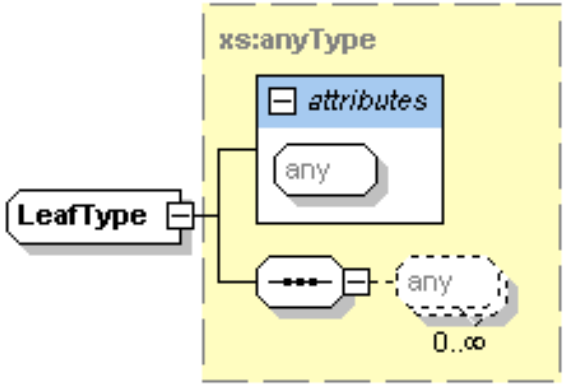
Zweite notwendige Änderung

```
<xs:complexType name="NodeType">
    <xs:sequence minOccurs="0" maxOccurs="2">
      <xs:choice>
        <xs:element name="Node" type="INodeType"/>
        <xs:element name="Leaf" type="LeafType"/>
      </xs:choice>
    </xs:sequen
    <xs:attribut
</xs:complexTy
```

```
<xs:complexType name="INodeType">
  <xs:complexContent>
    <xs:restriction base="NodeType">
      <xs:choice minOccurs="1" maxOccurs="2">
        <xs:element name="Node" type="INodeType"/>
        <xs:element name="Leaf" type="LeafType"/>
      </xs:choice>
    </xs:restriction>
  </xs:complexContent>
</xs:complexType>
```

LeafType: Blätter

```
<xs:complexType name="LeafType">
  <xs:complexContent mixed="true">
    <xs:extension base="xs:anyType"/>
  </xs:complexContent>
</xs:complexType>
```

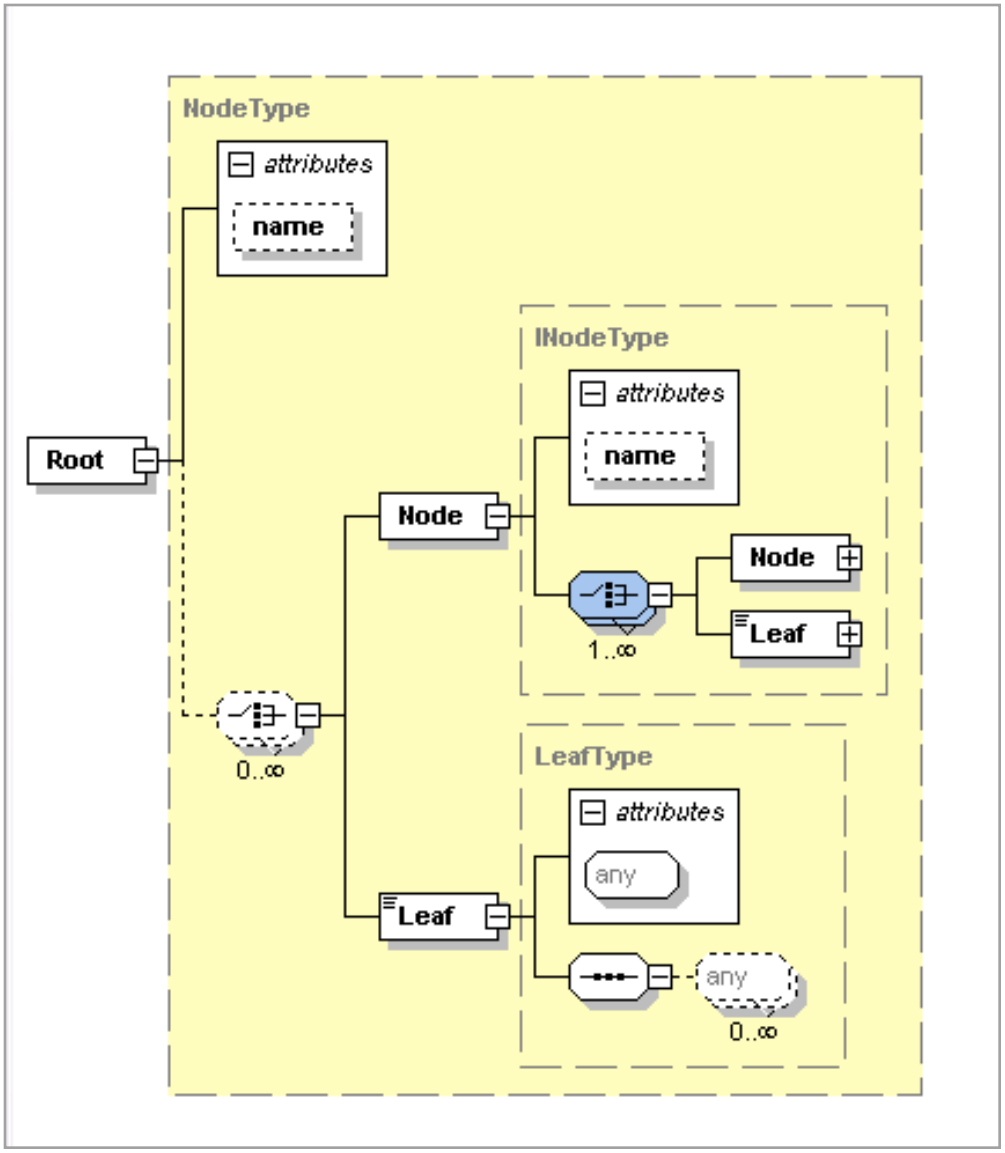


- beliebige Kind-Elemente erlaubt
- beliebige Attribute erlaubt, insbesondere also name
- xs:anyType erlaubt keinen gemischten Inhalt, deshalb Erweiterung
- Beachte: LeafType keine Untermenge von NodeType, kann daher nicht mit xs:restriction definiert werden!

Eine Lösung

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
3    <xs:element name="Root" type="NodeType"/>
4    <xs:complexType name="NodeType">
5      <xs:choice minOccurs="0" maxOccurs="unbounded">
6        <xs:element name="Node" type="INodeType"/>
7        <xs:element name="Leaf" type="LeafType"/>
8      </xs:choice>
9      <xs:attribute name="name" type="xs:string"/>
10   </xs:complexType>
11   <xs:complexType name="INodeType">
12     <xs:complexContent>
13       <xs:restriction base="NodeType">
14         <xs:choice maxOccurs="unbounded">
15           <xs:element name="Node" type="INodeType"/>
16           <xs:element name="Leaf" type="LeafType"/>
17         </xs:choice>
18       </xs:restriction>
19     </xs:complexContent>
20   </xs:complexType>
21   <xs:complexType name="LeafType" mixed="true">
22     <xs:complexContent mixed="true">
23       <xs:extension base="xs:anyType"/>
24     </xs:complexContent>
25   </xs:complexType>
26 </xs:schema>
```

Das gesamte XML-Schema



Freie Universität Berlin

Musterlösung des Übungsblattes 3

Grundstruktur Instanz

```
1   <?xml version="1.0"?>
2   <purchaseOrder xmlns:xsi="
    http://www.w3.org/2001/XMLSchema-instance" xmlns="
    http://www.altova.com/IPO" xmlns:ipo="
    http://www.altova.com/IPO" orderDate="1999-12-01">
3     <shipTo export-code="1" xsi:type="ipo:EU-Address">
9     <billTo xsi:type="ipo:US-Address">
16    <Items>
57  </purchaseOrder>
```

Grundstruktur des XML-Schemas

```
1   <?xml version="1.0" encoding="UTF-8"?>
2   <schema xmlns="http://www.w3.org/2001/XMLSchema"
    xmlns:ipo="http://www.altova.com/IPO"
    targetNamespace="http://www.altova.com/IPO"
    elementFormDefault="qualified" attributeFormDefault="
    unqualified">
3     <include schemaLocation="address.xsd"/>
4     <element name="purchaseOrder" type="
    ipo:PurchaseOrderType"/>
5     <element name="comment" type="string"/>
6     <complexType name="PurchaseOrderType">
15    <complexType name="Items">
37    <simpleType name="Sku">
42  </schema>
```

address.xsd: Grundstruktur

```
1   <?xml version="1.0" encoding="UTF-8"?>
2   <schema xmlns="http://www.w3.org/2001/XMLSchema"
    xmlns:ipo="http://www.altova.com/IPO" targetNamespace="
    http://www.altova.com/IPO" elementFormDefault="qualified"
    attributeFormDefault="unqualified">
3     <complexType name="Address" abstract="true">
10    <complexType name="US-Address">
20    <complexType name="EU-Address">
30    <simpleType name="US-State">
89    <simpleType name="EU-Postcode">
93  </schema>
```

- gleicher Zielnamensraum wie Haupt-Schema
- 5 globale Datentypen (➔ Zielnamensraum)

Datentyp Address

```
3   <complexType name="Address" abstract="true">
4     <sequence>
5       <element name="name" type="string"/>
6       <element name="street" type="string"/>
7       <element name="city" type="string"/>
8     </sequence>
9   </complexType>
```

- gemeinsame Bestandteile von US-Address & EU-Address
- abstrakt: in Instanzen nicht erlaubt, auch wenn XML-Schema diesen Datentyp verlangt!
- muss in Instanz durch abgeleitete Datentypen ersetzt werden

Datentyp US-Address

```
10  <complexType name="US-Address">
11    <complexContent>
12      <extension base="ipo:Address">
13        <sequence>
14          <element name="state" type="ipo:US-State"/>
15          <element name="zip" type="positiveInteger"/>
16        </sequence>
17      </extension>
18    </complexContent>
19  </complexType>
```

- kann in Instanz ipo:Address ersetzen
- dann xsi:type="ipo:US-Address" angegeben

⇨ Zusammenhang zwischen xsi:type="ipo:US-Address" und Struktur hergestellt

Datentyp US-State

```
30  <simpleType name="US-State">
31    <restriction base="string">
32      <enumeration value="AK"/>
33      <enumeration value="AL"/>
34      <enumeration value="AR"/>
35      ...
36      <enumeration value="WA"/>
37      <enumeration value="WI"/>
38      <enumeration value="WV"/>
39      <enumeration value="WY"/>
40    </restriction>
41  </simpleType>
```

Datentyp EU-Address

```
20  <complexType name="EU-Address">
21    <complexContent>
22      <extension base="ipo:Address">
23        <sequence>
24          <element name="postcode" type="ipo:EU-Postcode"/>
25        </sequence>
26        <attribute name="export-code" type="positiveInteger"
      fixed="1"/>
27      </extension>
28    </complexContent>
29  </complexType>
```

- kann in Instanz ipo:Address ersetzen
- dann xsi:type="ipo:EU-Address" angegeben

⇨ Zusammenhang zwischen xsi:type="ipo:EU-Address" und Struktur hergestellt

Erinnerung: Hauptschema

```
1   <?xml version="1.0" encoding="UTF-8"?>
2   <schema xmlns="http://www.w3.org/2001/XMLSchema"
    xmlns:ipo="http://www.altova.com/IPO"
    targetNamespace="http://www.altova.com/IPO"
    elementFormDefault="qualified" attributeFormDefault="
    unqualified">
3       <include schemaLocation="address.xsd"/>
4       <element name="purchaseOrder" type="
    ipo:PurchaseOrderType"/>
5       <element name="comment" type="string"/>
6       <complexType name="PurchaseOrderType">
15      <complexType name="Items">
37      <simpleType name="Sku">
42  </schema>
```

Datentyp PurchaseOrderType

```
 6  <complexType name="PurchaseOrderType">
 7    <sequence>
 8      <element name="shipTo" type="ipo:Address"/>
 9      <element name="billTo" type="ipo:Address"/>
10      <element ref="ipo:comment" minOccurs="0"/>
11      <element name="Items" type="ipo:Items"/>
12    </sequence>
13    <attribute name="orderDate" type="date"/>
14  </complexType>
```

- ipo:Address darf in Instanzen nicht vorkommen, da abstrakt
- muss in Instanz entweder durch xsi:type="ipo:US-Address" oder xsi:type="ipo:EU-Address" ersetzt werden

Datentyp Items

```
15  <complexType name="Items">
16    <sequence>
17      <element name="item" maxOccurs="unbounded">
18        <complexType>
19          <sequence>
20            <element name="productName" type="string"/>
21            <element name="quantity">
22              <simpleType>
27            </element>
28            <element name="price" type="decimal"/>
29            <element ref="ipo:comment" minOccurs="0"/>
30            <element name="shipDate" type="date"/>
31          </sequence>
32          <attribute name="partNum" type="ipo:Sku"/>
33        </complexType>
34      </element>
35    </sequence>
36  </complexType>
```

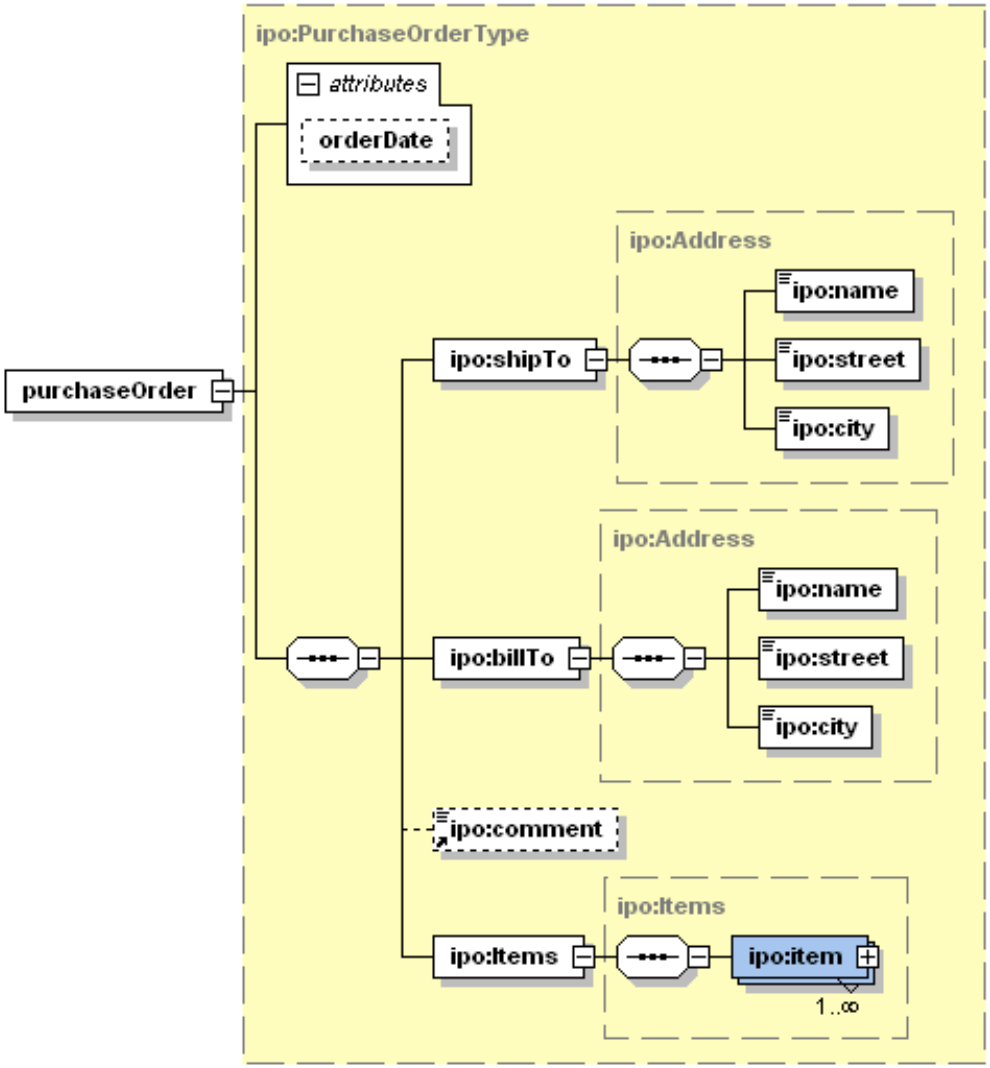
Element quantity

```
21  <element name="quantity">
22    <simpleType>
23      <restriction base="positiveInteger">
24        <maxExclusive value="1000"/>
25      </restriction>
26    </simpleType>
27  </element>
```

Datentyp Sku

```
37  <simpleType name="Sku">
38    <restriction base="string">
39      <pattern value="\d{3}-[A-Z]{2}"/>
40    </restriction>
41  </simpleType>
```

Das XML-Schema visualisiert



Musterfragen

Frage 1

Which of the following elements is a valid XML Schema root element?

A. <schema xmlns:xs=„http://www.w3.org/2001/XMLSchema“ targetNamespace=„http://www.example.org“>

B. <xs:schema targetNamespace=„http://www.example.org“>

C. <xs:schema
xmlns=„http://www.example.org“>
xmlns:xs=„http://www.w3.org/2001/XMLSchema“>

D. <schema xmlns=„http://www.example.org“>

Frage 2

<data href="http://www.example.com/datAdat"/>

How would you write the most appropriate XML Schema definition for the href attribute?

A. <xs:attribute name="href" type="xs:string"/>

B. <xs:attribute name="href" type="xs:anyURI"/>

C. <xs:attribute name="href" type="xs:NMTOKEN"/>

D. <xs:attribute name="href" type="xs:anyType"/>

E. <xs:attribute name="href" type="xs:QName"/>

Frage 3

Consider the following XML Schema fragment:

```
<xs:simpleType name='strList'>
  <xs:list itemType='xs:string'/>
</xs:simpleType>
```

Suppose this schema were instantiated in an XML document as follows:

```
<myStuff xsi:type='strList'>
  Mary had a little lamb, Its fleece was white as snow, And
  everywhere that Mary went, The lamb was sure to go.
</myStuff>
```

What is the length of the myStuff list?

A. 1 **B.** 4 **C.** 22 **D.** 107

Frage 4

Which XML Schema elements should occur before the following type definition?

<xsd:extension base=„xsd:integer“>

<xsd:attribute name=„currency“ type=„xsd:string“ />

</xsd:extension>

A. <xsd:simpleType> <xsd:simpleContent>

B. <xsd:simpleType> <xsd:complexContent>

C. <xsd:complexType> <xsd:simpleContent>

D. <xsd:complexType> <xsd:complexContent> <<<<<

Frage 5

<item SKU="19724814" />

How would you write an XML Schema to validate this fragment?

A.
```
<xs:element name="item">
  <xs:complexType>
    <xs:complexContent>
      <xs:restriction base="xs:integer">
        <xs:attribute name="SKU" type="xs:positiveInteger"/>
      </xs:restriction>
    </xs:complexContent>
  </xs:complexType>
</xs:element>
```

Frage 5

<item SKU="19724814" />

How would you write an XML Schema to validate this fragment?

A.
```
<xs:element name="item">
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="xs:integer">
        <xs:attribute name="SKU" type="xs:positiveInteger"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>
```

Frage 5

<item SKU="19724814" />

How would you write an XML Schema to validate this fragment?

B.
```
<xs:element name="item">
  <xs:complexType>
    <xs:attribute name="SKU" type="xs:positiveInteger"/>
  </xs:complexType>
</xs:element>
```

Frage 5

<item SKU="19724814" />

How would you write an XML Schema to validate this fragment?

B.
```
<xs:element name="item">
  <xs:complexType>
   <xs:complexContent>
    <xs:restriction base=„xs:anyType">
      <xs:attribute name="SKU" type="xs:positiveInteger"/>
    </xs:restriction>
   </xs:complexContent>
  </xs:complexType>
</xs:element>
```

Anhang: Eine 2. Lösung für Bäume

Root: Wurzel eines Baumes

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

- Beachte: kein Ziel-Namensraum
- durchaus erlaubt
- manchmal sogar sinnvoll!

Warum kein Ziel-Namensraum?

Freie Universität Berlin

- globale Elemente immer namensraumeingeschränkt
- globales Element: Element-Deklaration direktes Kind-Element von xsd:schema
- lokale Elemente: alle anderen Element-Deklarationen
- analoge Definitionen für Attribute und Typ-Definitionen
- Wurzel-Element der Instanz muss globales Element im XML-Schema sein

Globale vs. lokale Elemente

```
<?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:sequence minOccurs="0" maxOccurs="unbounded">
      <xs:choice>
        <xs:element name="Node" type="INodeType"/>
        <xs:element name="Leaf" type="LeafType"/>
      </xs:choice>
    </xs:sequence>
    <xs:attribute name="name" type="xs:string"/>
  </xs:complexType>
        …
</xs:schema>
```

global

lokal

Lokales Element ⇨ globales Element

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="Root" type="NodeType"/>
  <xs:element name="Node" type="INodeType"/>
  <xs:element name="Leaf" type="LeafType"/>
  <xs:complexType name="NodeType">
    <xs:sequence minOccurs="0" maxOccurs="unbounded">
      <xs:choice>
        <xs:element ref="Node"/>
        <xs:element ref="Leaf"/>
      </xs:choice>
    </xs:sequence>
    <xs:attribute name="name" type="xs:string"/>
  </xs:complexType>
        …
</xs:schema>
```

global

ElementFormDefault

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
                    targetNamespace="http://www.trees.org"
                    elementFormDefault="unqualified">
  <xs:element name="Root" type="NodeType"/>
        …
</xs:schema>
```

- qualified: Lokale Elemente der Instanz müssen namensraumeingeschränkt sein.
- unqualified: Lokale Elemente dürfen nicht namensraumeingeschränkt sein (Standard-Wert).

⇨ ElementFormDefault betrifft nur lokalen Elemente, nicht die globalen!

Aha!

Freie Universität Berlin

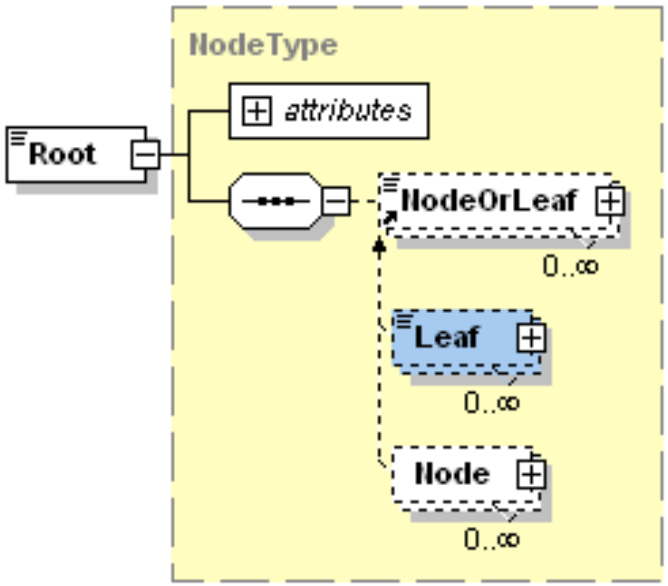
- Ziel-Namensraum enthält alle
  - globalen Elemente
  - globalen Attribute
  - globalen Typ-Definitionen
- Instanzen: Elemente und Attribute des Ziel-Namensraumes immer namensraumeingeschränkt
- elementFormDefault betrifft nur lokale Elemente
- wenn kein Element namensraumeingeschränkt sein soll:

  kein Zielnamensraum angegeben!

Und hier endlich die Lösung

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="Root" type="NodeType"/>
  <xs:element name="NodeOrLeaf" type="NodeType" abstract="true"/>
        …
</xs:schema>
```

Abstraktes Element NodeOrLeaf



- darf nicht in Instanzen verwendet werden
- muss in Instanzen ersetzt werden:
  hier durch Leaf oder Node

NodeOrLeaf: abstraktes Element

```
<xs:element name="Root" type="NodeType"/>
<xs:element name="NodeOrLeaf" type="NodeType" abstract="true"/>
<xs:complexType name="NodeType" mixed="true">
  <xs:sequence>
    <xs:element ref="NodeOrLeaf" minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:attribute name="name" type="xs:string"/>
</xs:complexType>
```

- abstraktes Element mit abstract="true" gekennzeichnet
- muss in Instanzen ersetzt werden
- mögliche Ersetzungen mit substitutionGroup gekennzeichnet
- binäre Bäume: maxOccurs="2" **statt** "unbounded"

1. Ersetzungsmöglichkeit: Node

```
<xs:element name="Node" type="INodeType"
                    substitutionGroup="NodeOrLeaf"/>
<xs:complexType name="INodeType" mixed="false">
  <xs:complexContent>
    <xs:restriction base="NodeType">
      <xs:sequence>
        <xs:element ref="NodeOrLeaf" maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="name" type="xs:string" use="required"/>
    </xs:restriction>
  </xs:complexContent>
</xs:complexType>
```

- Einschränkung Element: minOccurs="0" ➔ Standard-Wert "1"
- Einschränkung Attribut: optional ➔ obligatorisch

2. Ersetzungsmöglichkeit: Leaf

```
<xs:element name="Leaf" type="LeafType"
                    substitutionGroup="NodeOrLeaf"/>
<xs:complexType name="LeafType" mixed="true">
  <xs:complexContent>
    <xs:restriction base="NodeType">
      <xs:sequence/>
      <xs:attribute name="name" type="xs:string" use="required"/>
    </xs:restriction>
  </xs:complexContent>
</xs:complexType>
```

mixed="true", da sonst Standardwert "false" ererbten Wert "true" überschreiben würde.

- Einschränkung Element: keine Elemente
- Einschränkung Attribut: optional ➔ obligatorisch