Freie Universität Berlin

XML-Schema im Detail

Robert Tolksdorf
Freie Universität Berlin
Institut für Informatik
Netzbasierte Informationssysteme
tolk@ag-nbi.de

Wie geht es weiter?

bisher

- ☑ Definition von XML-Sprachen
- ☑ DTDs und XML-Schema anhand eines Beispiels

heutige Vorlesung

- XML-Schema
  - Allgemeines
  - Element- und Attribut-Deklarationen
  - Datentypen
  - Typsubstitution
  - Schemaübernahme

XML Schema: allgemeine Infos

XML Schema

- DTD reichten nicht aus
  → Nachfrage nach einem mächtigerem Format für Beschreibung von XML-Dokumenten
- seit 1999 W3C Arbeiten an XML Schema
- seit Mai 2001 W3C Recommendation
- seit Oktober 2004 Second Edition

- XML Schema ist
  - eine XML basierte Alternative für ein DTD
  - formale Beschreibung eines Vokabulars

- statt XML Schema wird oft die Abkürzung XSD (XML Schema Definition) benutzt

DTDs vs. XML Schema

DTDs	XML Schema
vereinfachte SGML-DTD, Teil von XML 1.0/1.1	eigener W3C-Standard
eigene Sprache/Syntax	XML-Schema = XML-Sprache
kompakter und lesbarer	ausdrucksstark & mächtig
nur wenige „Datentypen“	unterstützt Datentypen
reihenfolgeunabhängige Strukturen schwierig zu definieren	reihenfolgeunabhängige Strukturen einfach zu definieren
Datentypen nicht erweiterbar, d.h. keine eigenen Datentypen	Datentypen erweiterbar, d.h. Definition von eigenen Datentypen möglich
keine Namenräume	Unterstützt Namenräume
zur Beschreibung von Text-Dokumenten ausreichend	zur Beschreibung von Daten besser geeignet

XML Schemata …

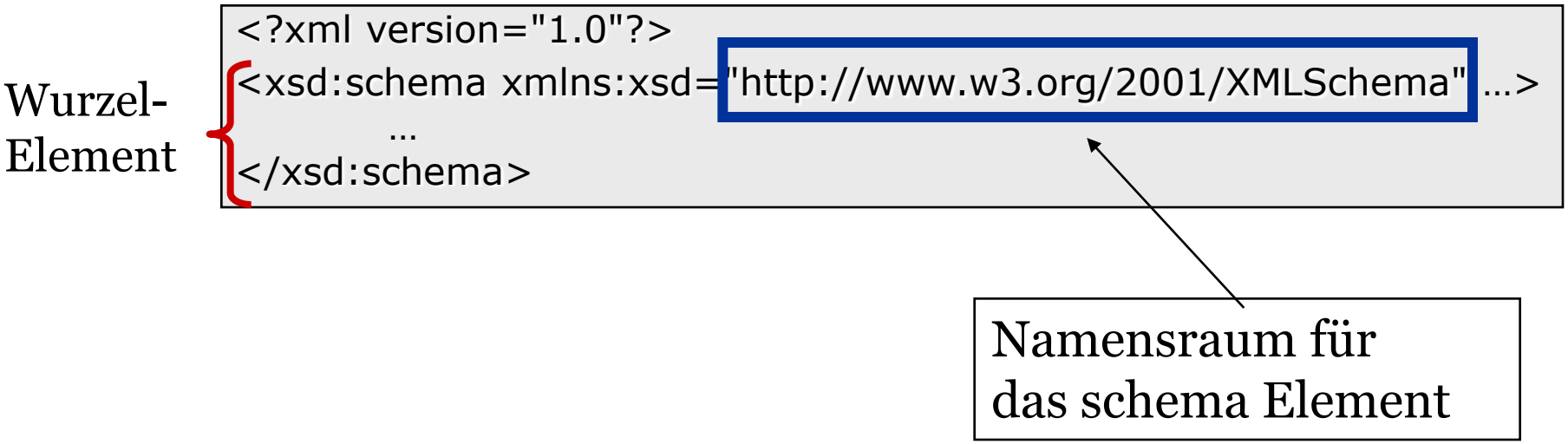
- …legen die das Vokabular und die Grammatik von XML Dokumenten fest, d.h. beschreiben u.a.:
  - Elemente, die in einem Dokument vorkommen können
  - Attribute, die in einem Dokument vorkommen können
  - welche Elemente sind Kinder-Elemente von welchen Elementen
  - Reihenfolge & Anzahl der Kinder-Elemente
  - Datentypen von Elementen und Attributen
  - die „default-“ und „fixed“-Werte von Elementen und Attributen
- … sind erweiterbar, weil sie in XML geschrieben sind
- mit extensible Schema Definition kann man
  - Schemata in anderen Schemata benutzen
  - eigene Datentypen von den Standard-Datentypen ableiten

XML Schema – Dokumentation

- XML Schema Teil 0: Primer
  - kurze Einführung mit Beschreibung der Möglichkeiten von XML Schema
  - http://www.w3.org/TR/xmlschema-0/
- XML Schema Part 1: Structures Second Edition
  - Strukturen der Definitionssprache XML Schema
  - http://www.w3.org/TR/xmlschema-1/
- XML Schema Part 2: Datatypes Second Edition
  - Beschreibung der Datentypen
  - http://www.w3.org/TR/2004/REC-xmlschema-2-20041028/

Alle W3C Standards & Drafts: http://www.w3.org/TR/

XML Schema



- Wurzel-Element: <schema> aus W3C-Namensraum http://www.w3.org/2001/XMLSchema
  - hier XML-Schema für XML-Schema hinterlegt: Schema der Schemata

Datentypen

Warum XML-Schema?


```
<location>
   <latitude>32.904237</latitude>
   <longitude>73.620290</longitude>
   <uncertainty units="meters">2</uncertainty>
</location>
```

XML-Schema

DTD

- Ortsangabe: besteht aus Breitengrad, Längengrad und Unsicherheit.
- Breitengrad: Dezimalzahl zwischen -90 und +90
- Längengrad: Dezimalzahl zwischen -180 und +180
- Unsicherheit: nicht-negative Zahl
- Maßeinheit für Unsicherheit: Meter oder Fuß

Wozu Datentypen?

<location>
<latitude>
$\{x \in Float : -90 \leq x \leq 90\}$
</latitude>
<longitude>
$\{x \in Float : -180 \leq x \leq 180\}$
</longitude>
<uncertainty units="{m, ft}">
$\{x \in Float : x \geq 0\}$
</uncertainty>
</location>

Datentypen definieren

- z.B. gültigen Inhalt von latitude, longitude, uncertainty und units
- aber auch gültigen Inhalt von location

können z.B. verwendet werden, um Schnittstelle eines Web Services zu beschreiben

Was sind Datentypen?

- **Dokument-Typ**: gültiger Inhalt eines gesamten XML-Dokumentes
- **Datentyp**: gültiger Inhalt von Elementen oder Attributen
- Formal repräsentiert ein Datentyp eine Menge von gültigen Werten, den so genannten **Wertebereich**.

Mit Datentypen ist es einfacher…

- erlaubten Dokument-Content zu beschreiben
- Korrektheit der Daten zu validieren
- mit Daten von einer DB zu arbeiten
- Facetten (Einschränkungen von Datentypen) zu definieren
- Data Patterns zu definieren
- Daten zwischen verschiedenen Datentypen zu konvertieren

Deklaration vs. Definition

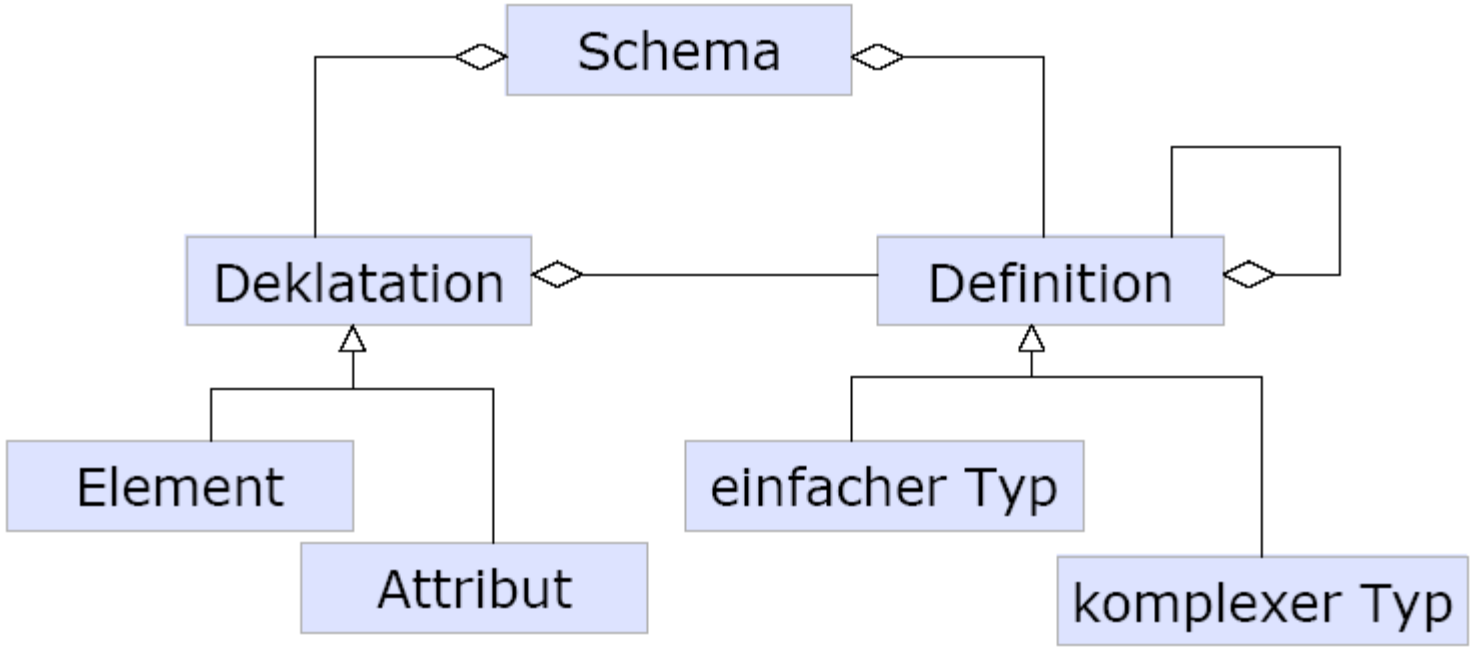
Deklaration

- Beschreibt/spezifiziert ein Element oder Attribut, das im Instanzdokument vorkommen darf

Definition

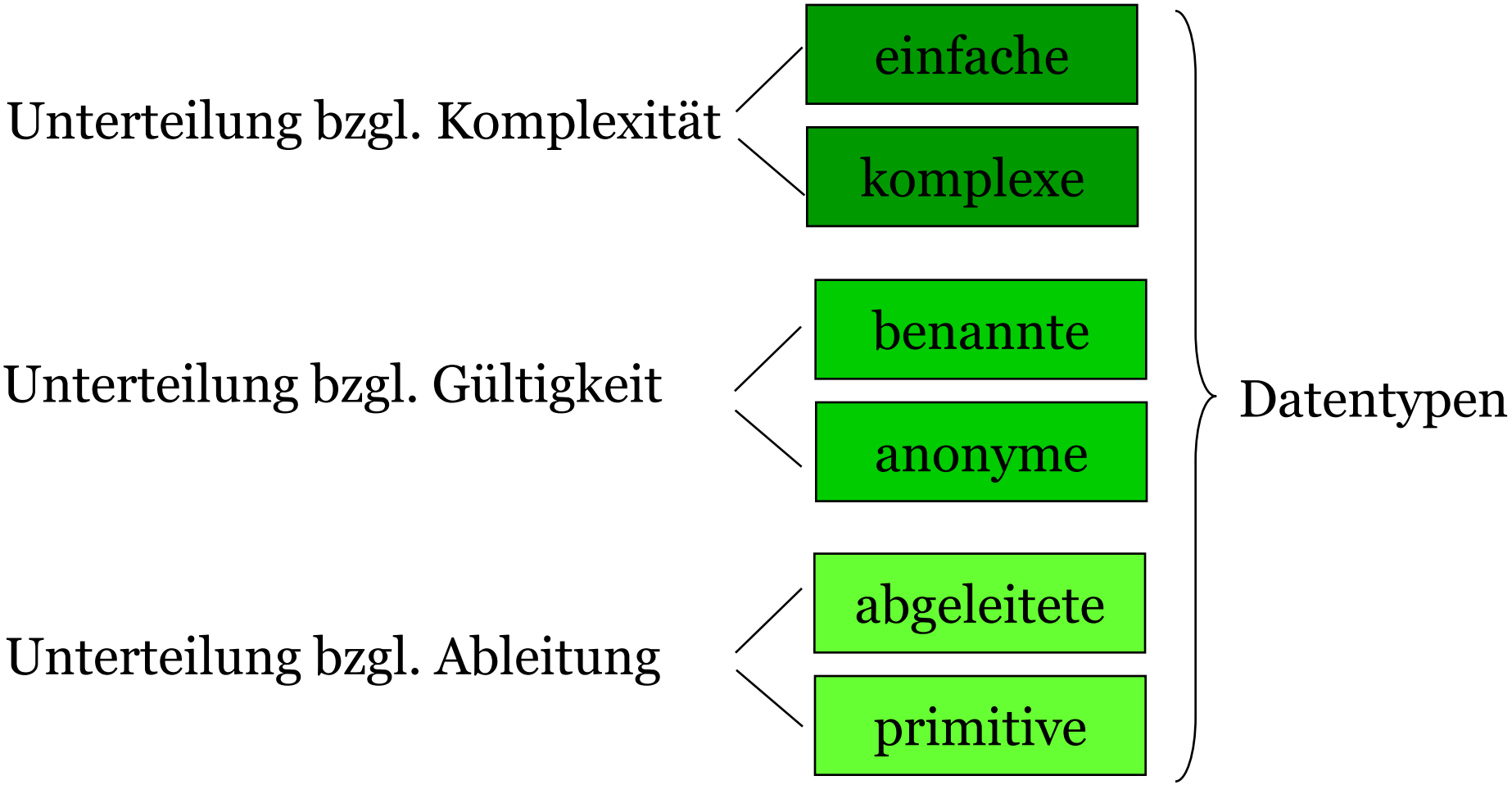
- definiert einen Typ, der in einer Element- oder Attribut-Deklaration verwendet werden kann

Aufbau



Quelle: http://www.sws.bfh.ch/~amrhein/XSL/Skripten/XSD.pdf

Verschiedene Arten von Datentypen



Einfache vs. komplexe Datentypen

Freie Universität Berlin

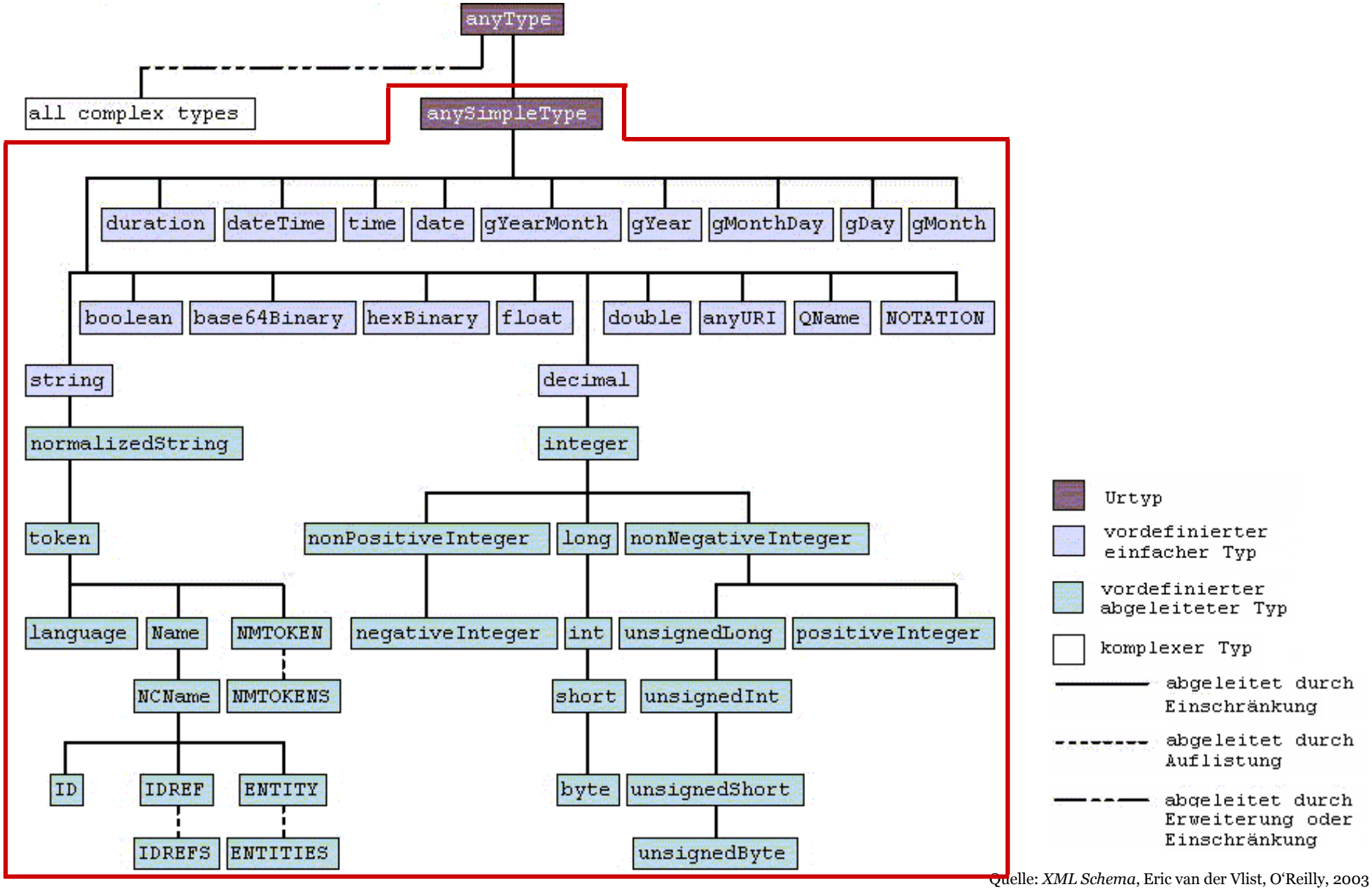
einfache Datentypen (simple types)

- beschreiben unstrukturierten Inhalt ohne Elemente oder Attribute (PCDATA)

komplexe Datentypen (complex types)

- beschreiben strukturierten XML-Inhalt mit Elementen oder Attributen
- natürlich auch gemischten Inhalt

Hierarchie der Datentypen



Quelle: *XML Schema*, Eric van der Vlist, O'Reilly, 2003

Anonyme vs. benannte Datentypen

Freie Universität Berlin

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

- anonymer Datentyp
- lokale Definition

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

- benannter Datentyp
- globale Definition
- wiederverwendbar

Globale vs. lokale Deklarationen – Beispiel

```
<?xml version="1.0"?>
    <xsd:schema xmlns="http://www.w3.org/2001/XMLSchema"…>
    <element name="name">
          <complexType>
                <sequence>
                      <element name="first" type="string"/>
                      <element name="middle" type="string"/>
                      <element name="last" type="string"/>
                </sequence>
                <attribute name="title" type="string"/>
          </complexType>
    </element>
</schema>
```

globale Deklaration <name>

lokale Deklarationen <first>,<middle>,<last>

- Globale Deklaration eines Datentypen
  - erscheint als direktes Nachkommen des Elements <xsd:schema>
  - kann wiederverwendet werden
- Lokale Deklaration eines Datentypen
  - keine Kinder vom Element <schema>
  - gültig nur in dem gegebenen Kontext

Einfache vs. komplexe Datentypen

Kategorien von Datentypen

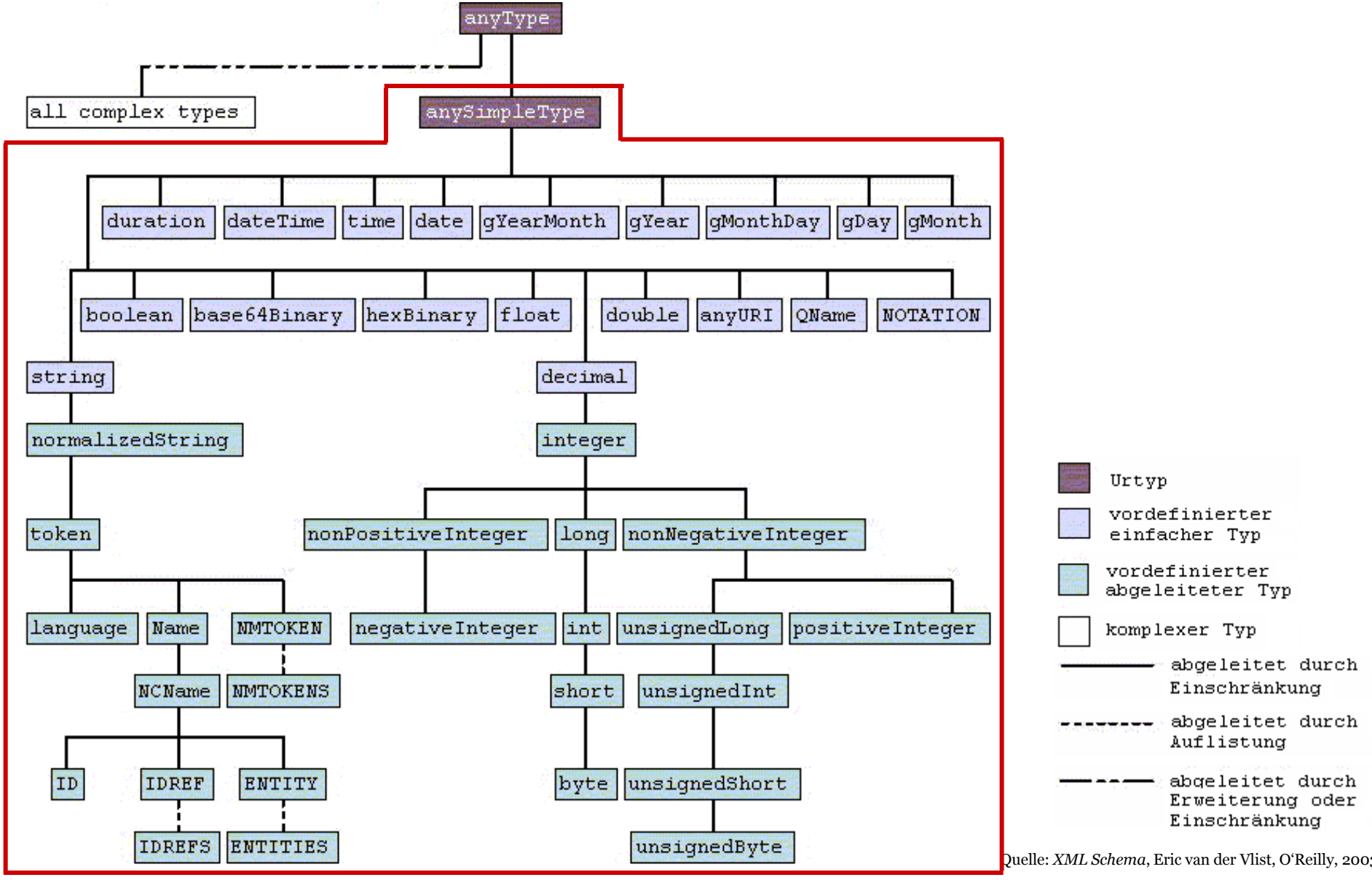
	(primitive)	abgeleitete
einfache	xsd:string xsd:language xsd:integer ...	<xsd:simpleType name="longitudeType"> **<xsd:restriction base="xsd:integer">** <xsd:minInclusive value="-180"/> <xsd:maxInclusive value="180"/> **</xsd:restriction>** </xsd:simpleType>
komplexe	<xsd:complexType> <xsd:sequence> ... </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>

Einfache Datentypen

Kategorien von Datentypen

	(primitive)	abgeleitete
einfache	xsd:string xsd:language xsd:integer ...	<xsd:simpleType name="longitudeType"> <**xsd:restriction base="xsd:integer"**> <xsd:minInclusive value="-180"/> <xsd:maxInclusive value="180"/> <**/xsd:restriction**> </xsd:simpleType>
komplexe	<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>

Hierarchie der Datentypen



Quelle: *XML Schema*, Eric van der Vlist, O'Reilly, 2003

Einfache Datentypen

einfache Datentypen (simple types)

- beschreiben unstrukturierten Inhalt ohne Elemente oder Attribute (PCDATA)
- Schema der Schemata definiert 44 einfache Datentypen
- eigene einfache Datentypen können definiert werden

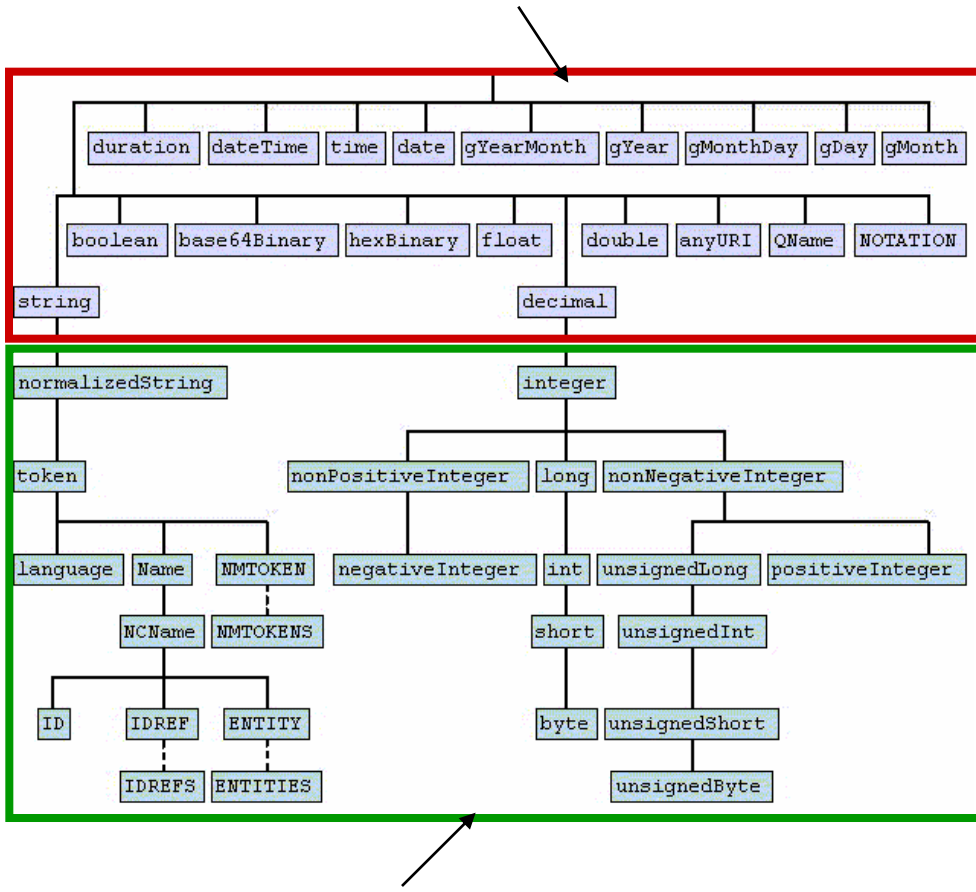
Primitive vs. abgeleitete Datentypen

primitive Datentypen
(primitive types)

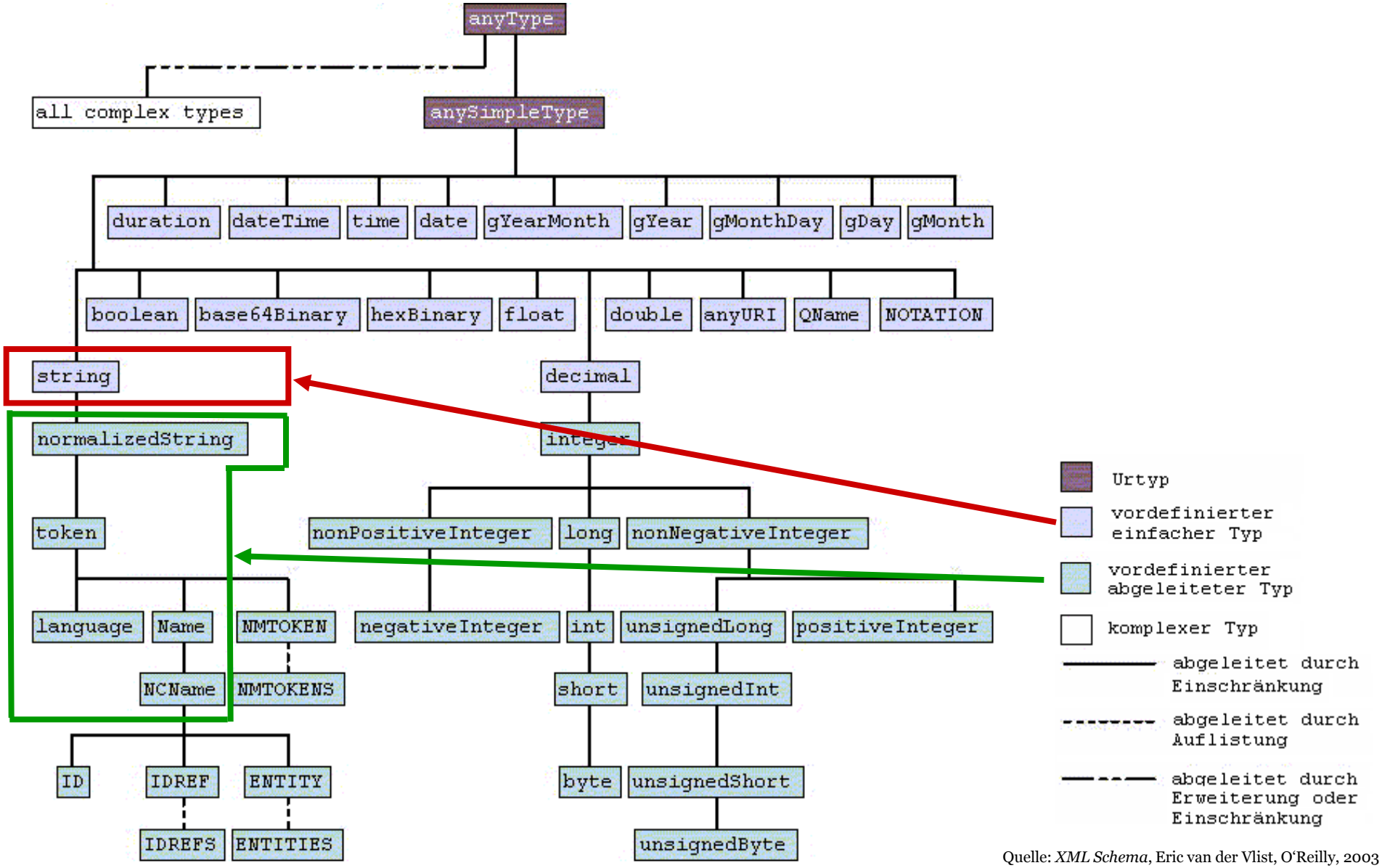
- nicht von anderen Datentypen abgeleitet

abgeleitete Datentypen
(derived types)

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

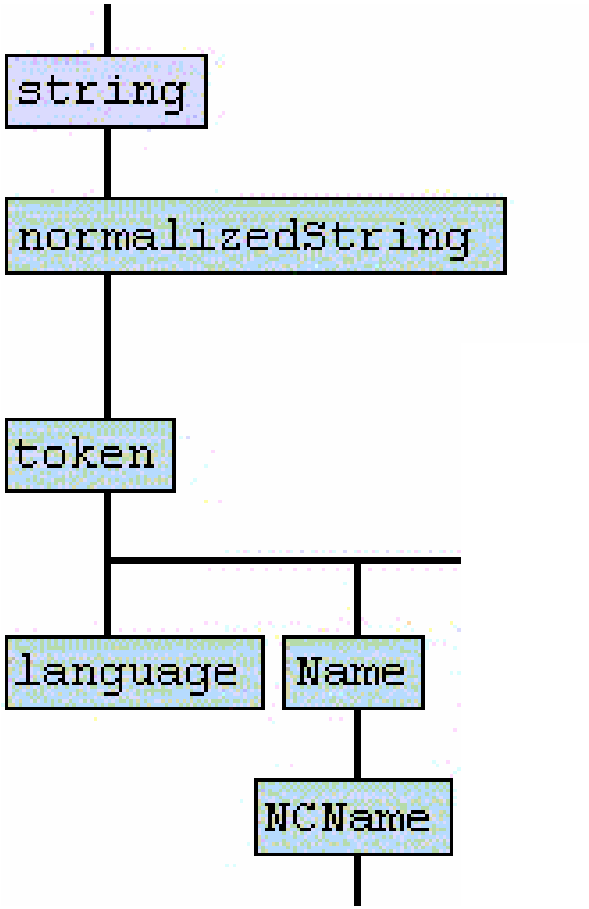


Hierarchie der Datentypen



Quelle: *XML Schema*, Eric van der Vlist, O'Reilly, 2003

Vordefinierte Datentypen (Auszug)



- **xsd:normalizedString: string** ohne Wagenrücklauf (CR), Zeilen-vorschub (LF) und Tabulator.
- **xsd:token: normalizedString** ohne 2 aufeinander folgende Leerzeichen und ohne Leerzeichen am Anfang und Ende.
- **xsd:Name: token**, der Namens-konvention von XML entspricht (mit oder ohne Präfix)
- **xsd:NCName: Name** ohne Präfix.
- **xsd:language**: Bezeichner für Sprache, wie z.B. „EN“

Abgeleitete einfache Datentypen

Freie Universität Berlin

1. **Einschränkung (Teilmenge)**

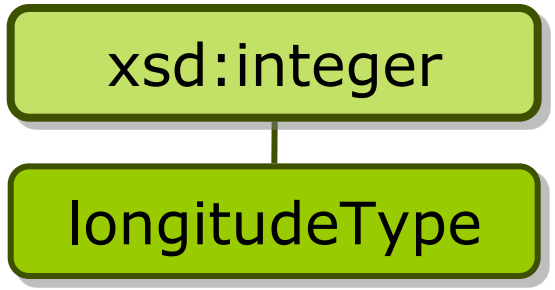
   Einschränkung des Wertebereiches eines einfachen Datentyps
2. **Vereinigung**

   Vereinigung der Wertebereiche mehrerer einfacher Datentypen
3. **Listen**

   Liste als String (PCDATA): einzelne Elemente durch White Spaces getrennt

1. Einschränkung <xsd:restriction>

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



hier konjunktiv verknüpft!

- longitudeType = { n aus xsd:integer: n ≥ -180, n ≤ 180 }
- Für jeden einfachen Datentyp bestimmte **zulässige Einschränkungen** (**constraining facets**) festgelegt.
- z.B. xsd:minInclusive und xsd:maxInclusive zulässig für xsd:integer, nicht jedoch für xsd:string

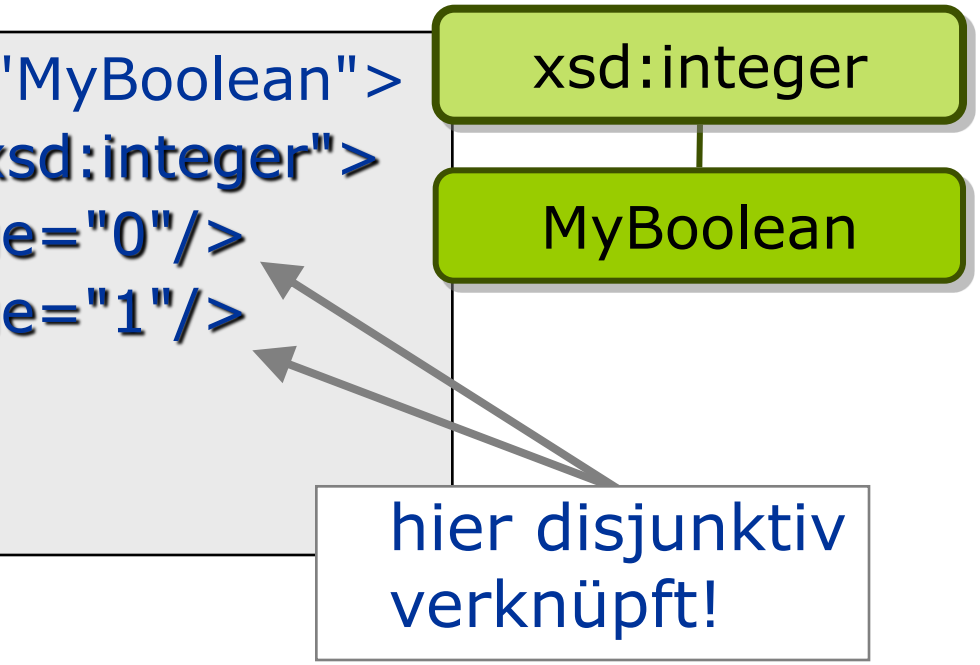
Zulässige Facetten

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

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>
```

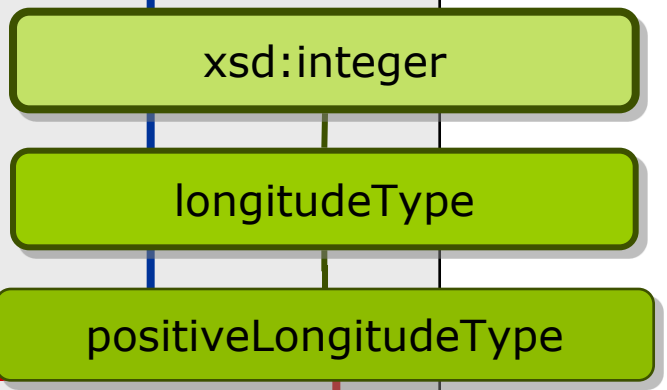


- MyBoolean = { n aus xsd:integer: n = 0 oder n =1 }
- xsd:enumeration: zählt alle Elemente des Wertebereiches explizit auf
- auch für xsd:string zulässig

Vererbung zulässiger Einschränkungen

Freie Universität Berlin

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

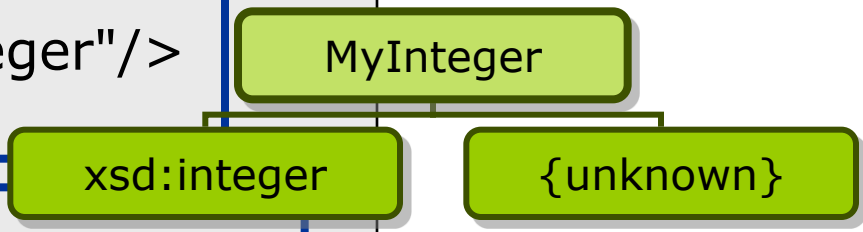



```
<xsd:simpleType name="postiveLongitudeType">
 <xsd:restriction base="longitudeType">
   <xsd:minInclusive value="0"/>
 </xsd:restriction>
</xsd:simpleType>
```

longitudeType erbt zulässige Einschränkungen von xsd:integer.

2. Vereinigung <xsd:union>

```
<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>
     <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>
```

```
<xsd:simpleType>
    xsd:integer
</xsd:simpleType>
```

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

3. Listen <xsd:list>

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

- IntegerList ist Liste von Integern (xsd:integer)
- einzelne Elemente der Liste durch beliebige White Spaces getrennt
- gültige Werte von IntegerList z.B.:

```
108  99  205  23 0
```

```
108 99
  205 23 0
```

```
108  99     205  23 0
```

Unstrukturierte Listen

Freie Universität Berlin

```
<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
- natürlich auch gemischten Inhalt

Elemente mit komplexen Typen können andere Elemente und/oder Attribute enthalten.

Reminder:

einfache Datentypen (simple types) beschreiben unstrukturierten Inhalt ohne Elemente oder Attribute (PCDATA)

Erzeugung von komplexen Datentypen

komplexe Datentypen

- bilden / beschreiben
  1. Sequenz <xsd:sequence>
  2. Menge <xsd:all>
  3. Auswahl <xsd:choice>

- ableiten
  1. Erweiterung <xsd:extension>
  2. Teilmenge <xsd:restriction>

Komplexe Datentypen bilden

Kategorien von Datentypen

	(primitive)	abgeleitete
einfache	xsd:string xsd:language xsd:integer ...	<xsd:simpleType name="longitudeType"> <**xsd:restriction base="xsd:integer"**> <xsd:minInclusive value="-180"/> <xsd:maxInclusive value="180"/> <**/xsd:restriction**> </xsd:simpleType>
komplexe	<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>

1. Sequenz <xsd:sequence>

```
<xsd:complexType name="BookType">
      <xsd:sequence maxOccurs="unbounded">
         <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>
```

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

- **Reihenfolge vorgegeben**
- Elemente erscheinen so oft, wie mit minOccurs/maxOccurs festgelegt.
- sequence selbst kann minOccurs und maxOccurs spezifizieren

2. 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>
```

- Jedes Element erscheint hier genau einmal.
- **Reihenfolge der Elemente beliebig**
- all selbst kann minOccurs und maxOccurs spezifizieren

Menge: minOccurs und maxOccurs

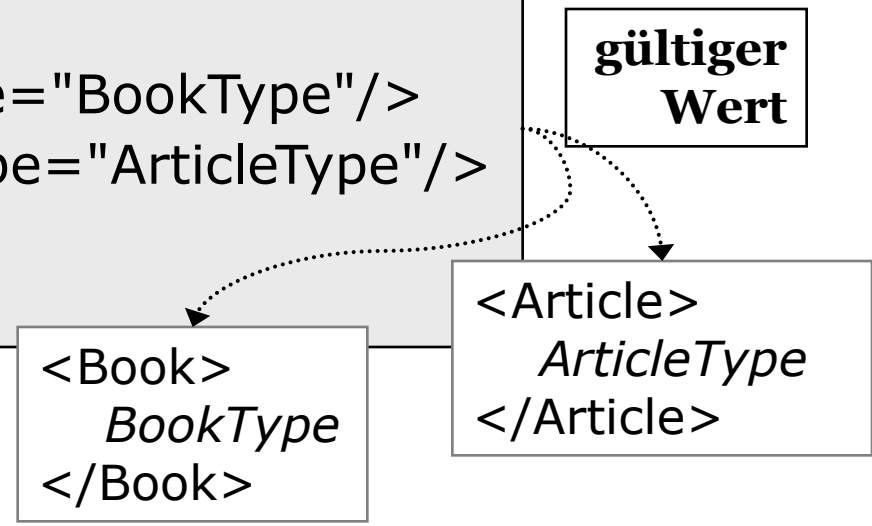
Freie Universität Berlin

```
<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>
```

- folg. Einschränkungen für minOccurs und maxOccurs:
  - minOccurs: nur "0" oder "1"
  - maxOccurs: nur "1"

3. 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 Alternativen
- hier also: entweder Book- oder Article-Element
- choice selbst kann minOccurs und maxOccurs spezifizieren

Verschachtelungen

- sequence, choice, all und Rekursion können verschachtelt werden:

```
<xs:element name="Chap" type="ChapType"/>
<xs:complexType name="ChapType">
  <xs:sequence>
    <xs:element name="Title" type="TitleType"/>
    <xs:choice maxOccurs="unbounded">
      <xs:element name="Para" type="ParaType"/>
      <xs:element name="Chap" type="ChapType"/>
    </xs:choice>
  </xs:sequence>
</xs:complexType>
```

entspricht:

```
<!ELEMENT Chap (Title, (Para | Chap)+)>
```

Gemischter Inhalt

Instanz

```
<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 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

Komplexe Datentypen ableiten

Kategorien von Datentypen

Freie Universität Berlin

(primitive) abgeleitete

einfache

xsd:string

xsd:language

xsd:integer

...

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

komplexe

```
<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>
```

Abgeleitete komplexe Datentypen

Freie Universität Berlin

1. **Erweiterung**

   Datentyp wird durch zusätzliche Attribute und Elemente erweitert.

2. **Teilmenge**

   Einschränkung des Wertebereiches eines Datentyps

Erinnerung: drei Möglichkeiten einfache Datentypen abzuleiten

1. Teilmenge
2. Vereinigung
3. Listen

1. Erweiterung <xsd:extension>

- Datentyp kann durch zusätzliche Attribute und Elemente erweitert werden.
- Sowohl **einfache als auch komplexe Datentypen** können **erweitert werden**.

- Ergebnis: **immer komplexer Datentyp (!)**

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

<xsd:extension> 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 ?

Freie Universität Berlin

```
<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 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 Objekte: beschreiben immer Inhalt von Element oder Attribut
- Attribut-Werte immer unstrukturiert
- Komplexer Datentyp StringWithLength kann nur Inhalt eines Elementes beschreiben.
- Zusätzliches Attribut 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
</Abstract>
```

(StringWithLength)

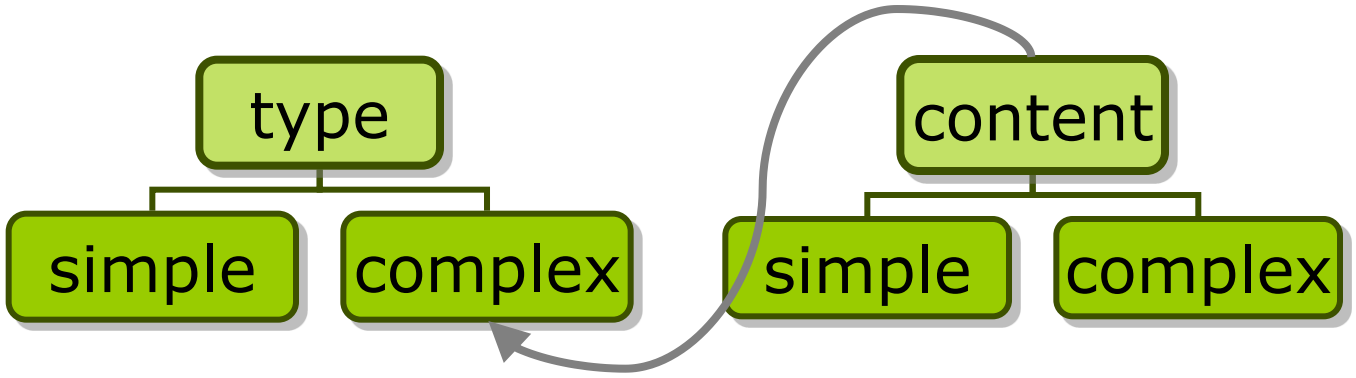
- Element Abstract hat Inhalt vom Typ StringWithLength.
- Attribut length von StringWithLength wird Element Abstract zugeordnet.

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 oder gemischter Inhalt (mit Elementen).
  - wird verlangt, obwohl eigentlich redundant
  - erleichtert aber Verarbeitung

Etwas kompliziert!

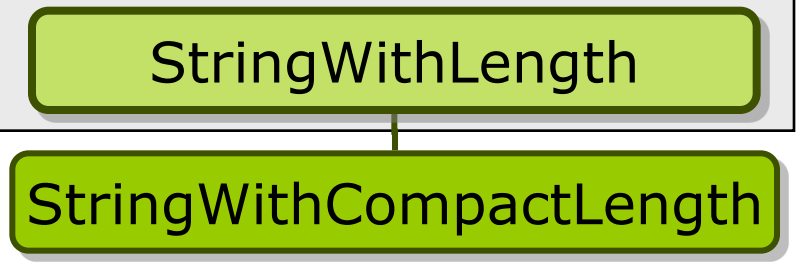


Elemente:	nein	ja	nein	ja
Attribute:	nein	ja	ja	ja

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

2. Teilmenge <xsd:restriction>

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



- Resultierender Datentyp darf nur gültige Werte des ursprünglichen Datentyps enthalten (echte Teilmenge).
- hier wäre z.B. xsd:string statt xsd:unsignedShort nicht erlaubt:

  xsd:string keine Teilmenge von xsd:nonNegativeInteger

Element-Deklarationen

Element-Deklaration: 1. Möglichkeit

Freie Universität Berlin

- Element kann mit benanntem Datentypen 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>
```

Instanz

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

Element-Deklaration: 1. Möglichkeit

Freie Universität Berlin

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

- **name**: Name des deklarierten Elementes
- **type**: Datentyp (benannt oder vordefiniert)
- **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: abhängig vom Kontext gibt es Einschränkungen von minOccurs und maxOccurs

Element-Deklaration: 2. Möglichkeit

Freie Universität Berlin

- 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>
```

Instanz

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

Element-Deklaration: 2. Möglichkeit

Freie Universität Berlin

- anonymer Datentyp ist entweder komplex:

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

- oder einfach:

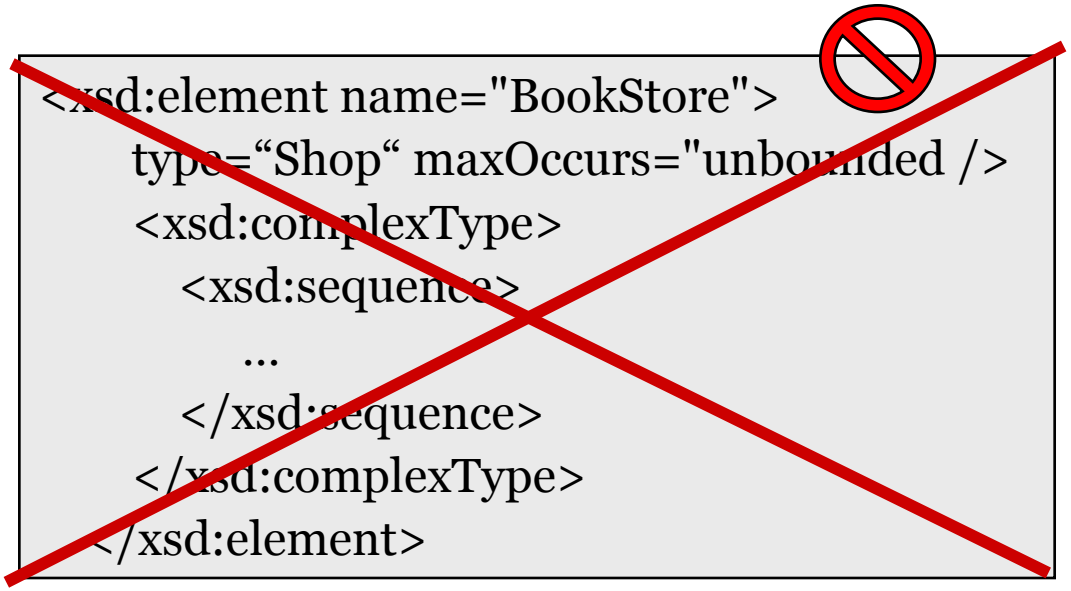
```
<xsd:element name="name" minOccurs="int" maxOccurs="int">
    <xsd:simpleType>
        ...
    </xsd:simpleType>
</xsd:element>
```

Element-Deklaration

- Eine Element-Deklaration kann entweder ein type Attribut haben oder eine anonyme Typdefinition enthalten → nie beides gleichzeitig!

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

```
<xsd:element name="BookStore"
   type="ShopType" maxOccurs="unbounded/>
```



<any>


```
<xsd:element name="BookStore">
 <xsd:complexType>
  <xsd:sequence>
   <xsd:element name="Book" type="BookType" maxOccurs="unbounded" />
   <xsd:any namespace= "##any " minOccurs= "0 " maxOccurs="unbounded"/>
  </xsd:sequence>
 </xsd:complexType>
</xsd:element>
```

- ##any erlaubt beliebige Elemente aus beliebigem Namensraum
- ##other erlaubt Elemente aus Namensraum ungleich targetNamenspace
- ##targetNamens erlaubt Elemente aus targetNamenspace

Attribut-Deklarationen

Deklaration von Attributen

- ähnlich wie bei Elementen
- aber nur **einfache Datentypen** erlaubt
- Deklaration mit **benanntem Datentyp**:

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

- oder Deklaration mit **anonymem Datentyp**:

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

Deklaration von Attributen


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

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

Wozu use="prohibited"?

Antwort: Um Vererbung vom komplexen Elterndatentyp zu unterbinden!

Globale vs. lokale Attribute

```
<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>
```

lokal: optional für root

global: optional für alle Elemente

global: Deklaration Kind von xsd:schema

lokal: Deklaration kein direktes Kind von xsd:schema

Verbieten von globalen Attributen

- Globales Attribut kann lokal mit use="prohibited" verboten werden
- Voraussetzung: globales Attribut wurde als optional deklariert

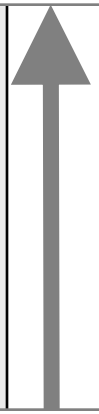
Erinnerung: globale Attribute in DTDs nicht möglich

Typsubstitution

Betrachte folg. XML-Schema

```
<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

Typsubstitution in der Instanz

Schema

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

Instanz

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

oder (!)

Instanz

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

Typsubstitution

- Voraussetzung: XML-Schema S leitet Datentyp t' von Datentyp t ab:

  entweder mit xsd:extension oder xsd:restriction
- Betrachten wir 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.

Mögliche Probleme

Freie Universität Berlin

t' Teilmenge (restriction) **von t**

- Laut Schema S müssen Anwendungen sowieso mit allen gültigen Werten von t umgehen, also auch mit t'.

⇨ unproblematisch

t' Erweiterung (extension) **von t**

- Laut Schema S müssen Anwendungen mit allen gültigen Werten von t umgehen, nicht aber mit zusätzlichen Attributen und Elementen

⇨ evtl. problematisch

⇨ Typsubstitution für Erweiterungen evtl. unterdrücken:

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

Schemaübernahme

Übernahme von Schema-Definitionen

Freie Universität Berlin

- <xsd:include> und <xsd:import>
  - immer vor allen anderen Komponenten
  - immer Kinder des Wurzelelements <xsd:schema>

- <xsd:include>
  - Datentypen aus einem anderen Schema mit gleichen Namensraum übernehmen
  - Angabe zur Herkunft des genutzten Schemas

```
<xsd:schema …>
  <xsd:include
        schemaLocation =“…“ />
  <xsd:element name=“…“>
    …
  </xsd:element>
</xsd:schema>
```

- <xsd:import>
  - Datentypen aus einem anderen Schema mit anderem Namensraum übernehmen
  - Angabe zur Herkunft des genutzten Schemas + Angabe des Namensraum

```
<xsd:schema …>
  <xsd:import namespace =“…“
        schemaLocation =“…“ />
  <xsd:element name=“…“>
    …
  </xsd:element>
</xsd:schema>
```

Wie geht es weiter?

heutige Vorlesung

- ☑ XML-Schema
  - ☑ Datentypen
  - ☑ Element- und Attribut-Deklarationen

Nächste Vorlesung

- XML-Parser

Erinnerung an die Mailingliste!

Freie Universität Berlin

- https://lists.spline.inf.fu-berlin.de/mailman/listinfo/nbi_v_xml

- Ankündigungen nur dort
  - Ausfall wegen Krankheit
  - Antworten auf Ihre Fragen
  - …