



XML-Schema

Warum XML-Schema?

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

XML-Schema

DTD

- Ortsangabe: 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

- XML-Sprache statt eigener Syntax
- Vielzahl von vordefinierten Datentypen
- eigene Datentypen ableitbar
- Namensraumunterstützung
- Reihenfolgeunabhängige Strukturen

Die Beispiel-DTD

```
<!ELEMENT BookStore (Book+)>
<!ELEMENT Book
    (Title, Author, Date, ISBN, Publisher)>
<!ELEMENT Title (#PCDATA)>
<!ELEMENT Author (#PCDATA)>
<!ELEMENT Date (#PCDATA)>
<!ELEMENT ISBN (#PCDATA)>
<!ELEMENT Publisher (#PCDATA)>
```

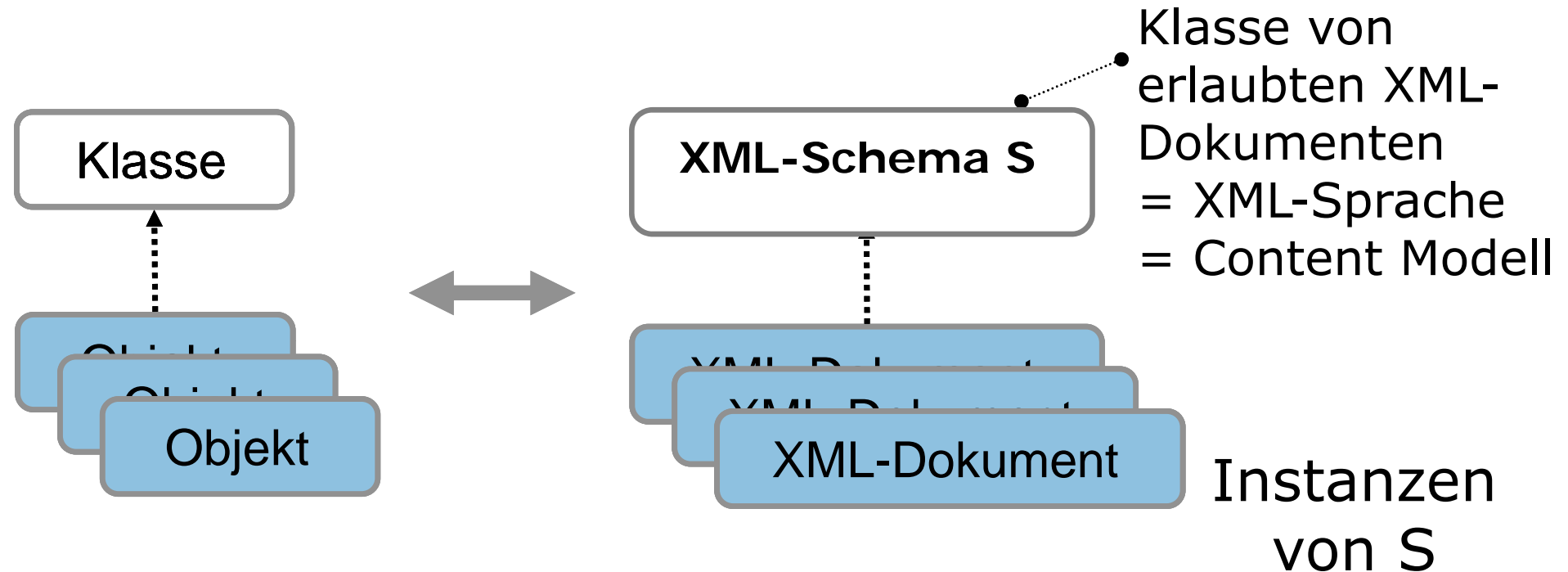
```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.books.org">
  <xsd:element name="BookStore">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="Book" maxOccurs="unbounded">
          <xsd:complexType>
            <xsd:sequence>
              <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>
        </xsd:element>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

- Schemaäquivalent für jede DTD
- Keine DTD für alle Schema
- XML-Schema ausdrucksmächtiger

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

- <http://www.w3.org/2001/XMLSchema> als XML-Schema für XML-Schema
- Ziel-Namensraum für im XML-Schema definiertes Vokabular (target namespace)
- Definiertes Vokabular kann über URI identifiziert werden

Instanz eines XML-Schemas



Instanz eines XML-Schemas S

ist ein XML-Dokument, das

1. dem Ziel-Namensraum von S zugeordnet ist und
2. gültig (valid) bzgl. S ist

```
<?xml version="1.0"?>
<BookStore xmlns="http://www.books.org"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.books.org
  http://www.books.org/BookStore.xsd">
  ...
</BookStore>
```

- Dokumententyp im Wurzel-Element und Namensraum festgelegt
- Wurzel-Element muss in XML-Schema global deklariert sein.
- Reicht bei bekannten Namensräumen wie <http://www.w3.org/1999/xhtml>


```
<?xml version="1.0"?>
<BookStore xmlns="http://www.books.org"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation=
    "http://www.books.org
    http://www.books.org/BookStore.xsd">
  ...
</BookStore>
```

- Attribut schemaLocation gibt Hinweis, wo das XML-Schema zu finden ist.
- XML-Prozessor darf diese Information ignorieren und anderes XML-Schema berücksichtigen!

Instanz

XML-Schema

schemaLocation=
"http://www.books.org BookStore.xsd"

targetNamespace=
"http://www.books.org"

BookStore.xml

BookStore.xsd

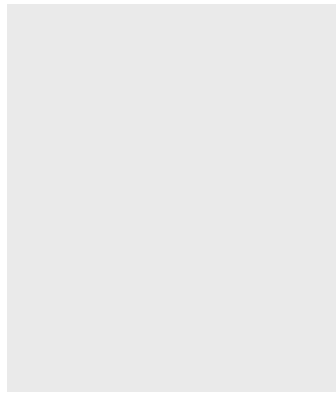
benutzt Namensraum
http://www.books.org

definiert Namensraum
http://www.books.org

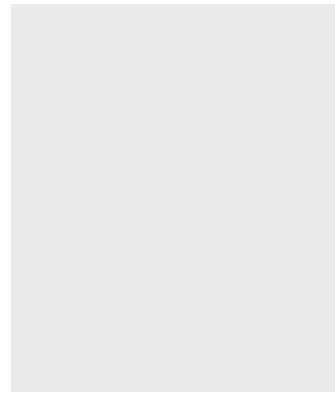
Validierung auf mehreren Ebenen

Instanz
= XML-Dokument

XML-Schema



BookStore.xml



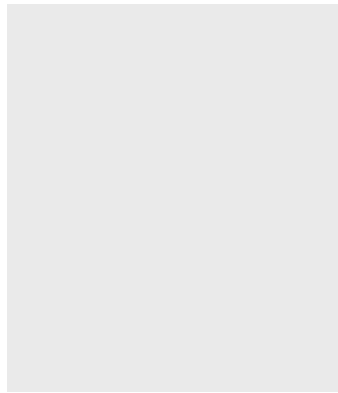
BookStore.xsd



zulässiges BookStore-
Dokument?

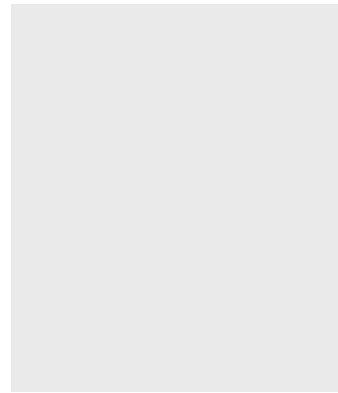
Validierung auf mehreren Ebenen

Instanz
= XML-Dokument



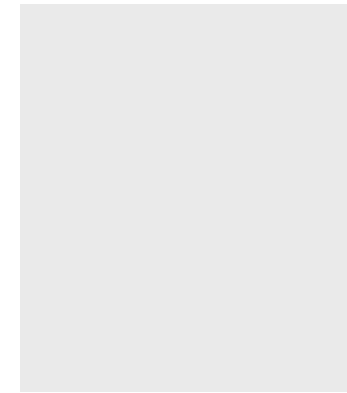
BookStore.xml

XML-Schema
= **XML-Dokument**



BookStore.xsd

Schema der
Schemata



XMLSchema.xsd



zulässiges BookStore-
Dokument?



zulässiges XML-Schema?

Validierung auf mehreren Ebenen

Instanz
= XML-Dokument



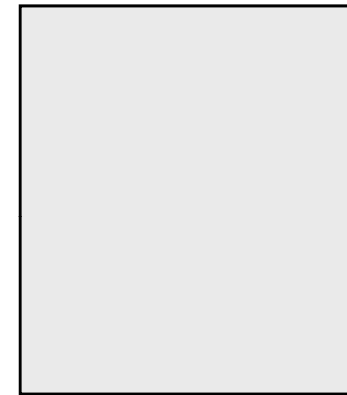
BookStore.xml

XML-Schema
= XML-Dokument

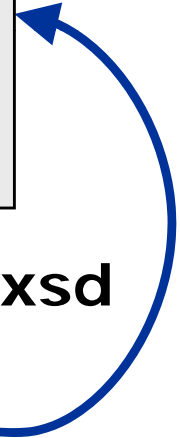


BookStore.xsd

Schema der Schemata
= XML-Dokument



XMLSchema.xsd



zulässiges BookStore-Dokument?

zulässiges XML-Schema?

Deklaration der Element-Struktur

```
<!ELEMENT BookStore (Book+)>  
<!ELEMENT Book (Title, Author, Date, ISBN, Publisher)>  
<!ELEMENT Title (#PCDATA)>  
<!ELEMENT Author (#PCDATA)>  
<!ELEMENT Date (#PCDATA)>  
<!ELEMENT ISBN (#PCDATA)>  
<!ELEMENT Publisher (#PCDATA)>
```

- Wie werden diese Element-Deklarationen mit einem XML-Schema ausgedrückt?

<!ELEMENT BookStore (Book+)>

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

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

- `xsd:element`: Element wird deklariert
- `xsd:complexType`: strukturierter Inhalt
- `xsd:sequence`: Sequenz von Elementen

<!ELEMENT BookStore (Book+)>

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

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

- **minOccurs:** minimale Anzahl der Wiederholungen
- **maxOccurs:** maximale Anzahl der Wiederholungen
- Standard-Werte jeweils 1



<!ELEMENT Book (Title, Author, Date, ISBN, Publisher)>

```
<xsd:complexType name="BookType">
  <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>
```


- Kind-Elemente: Title, Author, Date, ISBN und Publisher
- wegen `xsd:sequence`: feste Reihenfolge
- jeweils genau einmal

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

<!ELEMENT Book (Title, Author, Date, ISBN, Publisher)>

```
<xsd:complexType name="BookType">
  <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>
```

- **xsd:string:**
vordefinierter Datentyp
- **43 weitere vordefinierte**
Datentypen



```
<Book>
  <Title> PCDATA </Title>
  <Author>...</Author>
  <Date>...</Date>
  <ISBN>...</ISBN>
  <Publisher>...</Publisher>
</Book>
```

<!ELEMENT Book (Title, Author, Date, ISBN, Publisher)>

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

- **xsd:date: vordefinierter Datentyp**

```
<Book>
  <Title> PCDATA </Title>
  <Author>...</Author>
  <Date> Kalenderdatum </Date>
  <ISBN>...</ISBN>
  <Publisher>...</Publisher>
</Book>
```

Benannte Datentypen

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

```
<xsd:complexType name="BookType">  
  <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>
```

- BookType hier benannter Datentyp (named type)
- auch globale Typ-Definition genannt

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

- äquivalente Formulierung mit anonymen Datentyp
- auch als lokale Typ-Definition bezeichnet
- Nachteil: kann an anderer Stelle nicht wieder verwendet werden

<!ELEMENT Book (Title, Author+,
Date, ISBN?, Publisher)>

```
<xsd:complexType name="BookType">  
  <xsd:sequence>  
    <xsd:element name="Title" type="xsd:string"/>  
    <xsd:element name="Author" type="xsd:string"  
      minOccurs="1" maxOccurs="unbounded" />  
    <xsd:element name="Date" type="xsd:string"/>  
    <xsd:element name="ISBN" type="xsd:string" minOccurs="0" />  
    <xsd:element name="Publisher" type="xsd:string"/>  
  </xsd:sequence>  
</xsd:complexType>
```

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

Das vollständige XML-Schema

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            targetNamespace="http://www.books.org">
  <xsd:element name="BookStore">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="Book" maxOccurs="unbounded">
          <xsd:complexType>
            <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>
        </xsd:element>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```



Datentypen

<location>

<latitude>

$\{x \in \text{Float} : -90 \leq x \leq 90\}$

</latitude>

<longitude>

$\{x \in \text{Float} : -180 \leq x \leq 180\}$

</longitude>

<uncertainty units="{m, ft}">

$\{x \in \text{Float} : x \geq 0\}$

</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

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

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

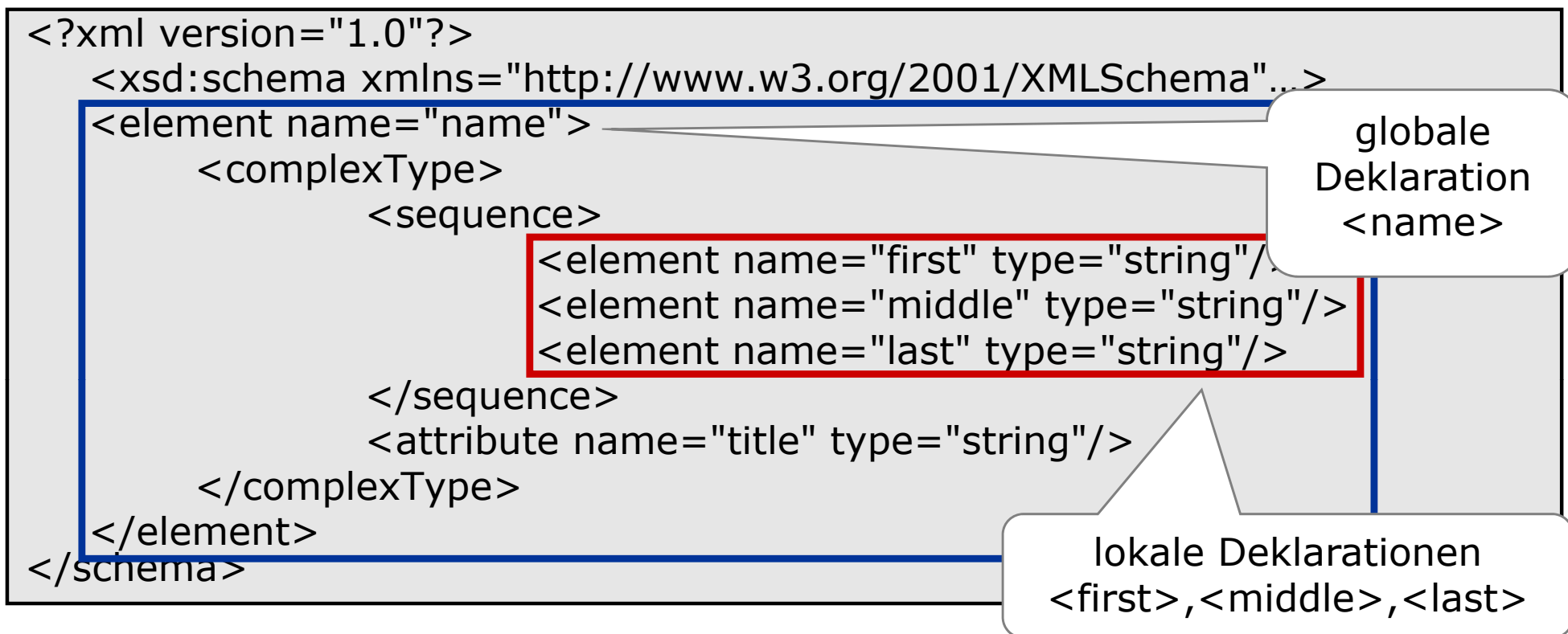
```
<xsd:element name="BookStore">  
  <xsd:complexType>  
    Liste von Büchern  
  </xsd:complexType>  
</xsd:element>
```

- anonymer Datentyp
- lokale Definition

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

- benannter Datentyp
- globale Definition
- wiederverwendbar

Globale vs. lokale Deklarationen – Beispiel



- **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 Datentypen

(primitive)

abgeleitete

einfache

xsd:string
xsd:language
xsd:integer
...

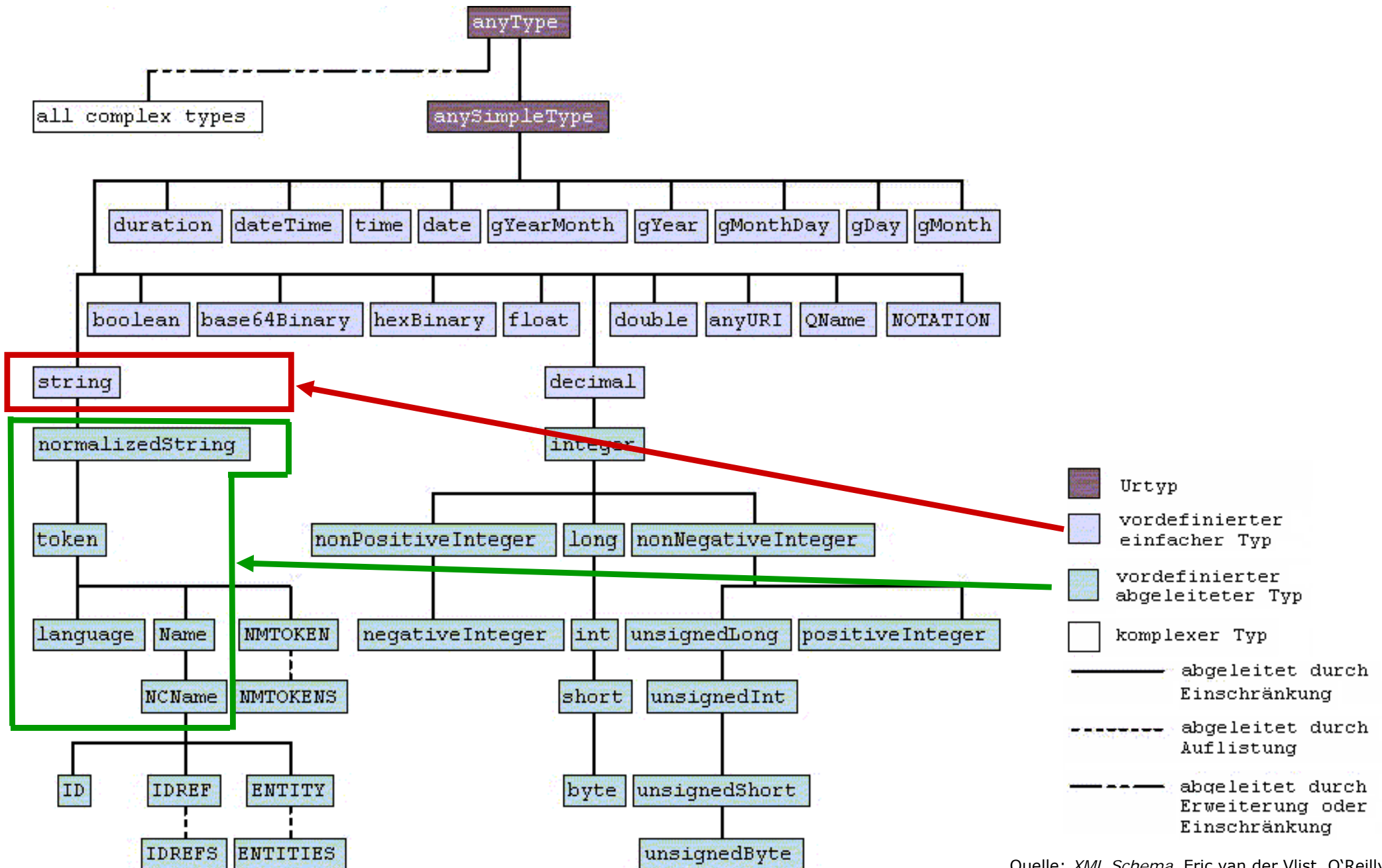
```
<xsd:simpleType name="longitudeType">
  <xsd:restriction base="xsd:integer">
    <xsd:minInclusive value="-180"/>
    <xsd:maxInclusive value="180"/>
  </xsd:restriction>
</xsd:simpleType>
```

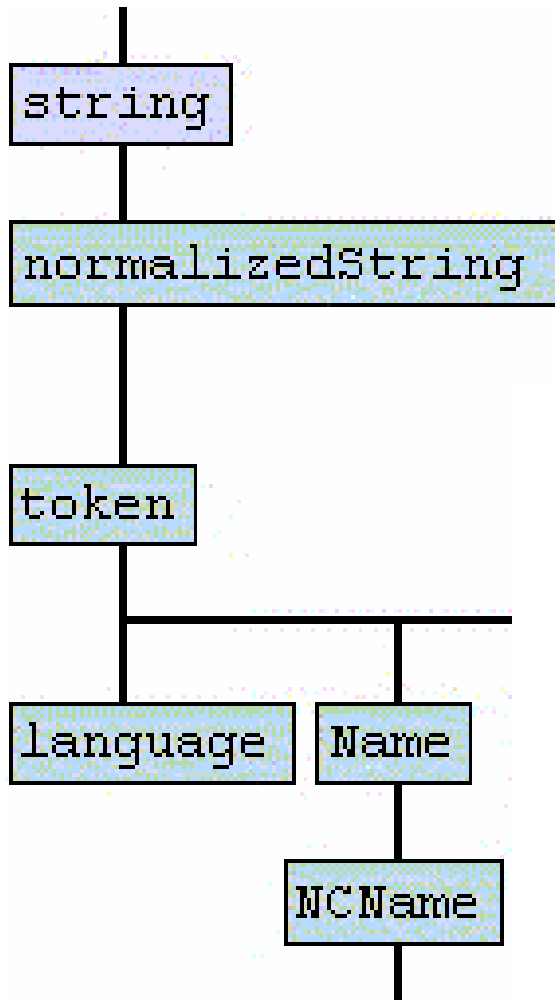
komplexe

```
<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





- **xsd:normalizedString: string** ohne Wagenrücklauf (CR), Zeilenvorschub (LF) und Tabulator.
- **xsd:token: normalizedString** ohne 2 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“

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>
```

xsd:integer

longitudeType

hier 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. `xsd:minInclusive` und `xsd:maxInclusive` zulässig für `xsd:integer`, nicht jedoch für `xsd:string`

- enumeration: Zählt erlaubte Werte explizit auf
- maxExclusive: <
- maxInclusive: \leq
- minExclusive: >
- minInclusive: \geq
- 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>
```

xsd:integer

MyBoolean

hier disjunktiv
verknüpft!

- 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

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

xsd:integer

longitudeType

positiveLongitudeType

```
<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>
```

MyInteger

```
<xsd:simpleType>
```

```
<xsd:restriction base="xsd:string">
```

```
<xsd:enumeration value="unknown"/>
```

```
</xsd:restriction>
```

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

xsd:integer

{unknown}

```
</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

```
<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 bilden

(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

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

```
<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>
```

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**

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

```
<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>
```

- 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>
```

**gültiger
Wert**

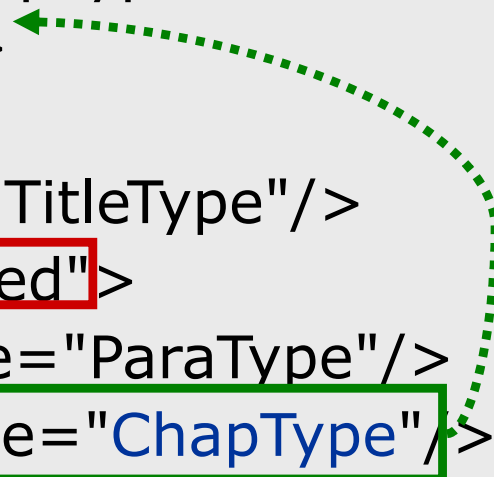
```
<Book>
  BookType
</Book>
```

```
<Article>
  ArticleType
</Article>
```

- Inhalt besteht aus **genau einem** der aufgezählten Alternativen
- hier also: entweder Book- oder Article-Element
- choice selbst kann minOccurs und maxOccurs spezifizieren

- 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)+>
```



```
<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
- Text (PCDATA) zwischen Elementen erlaubt

(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. 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

<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 ?

```
<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?

```
<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

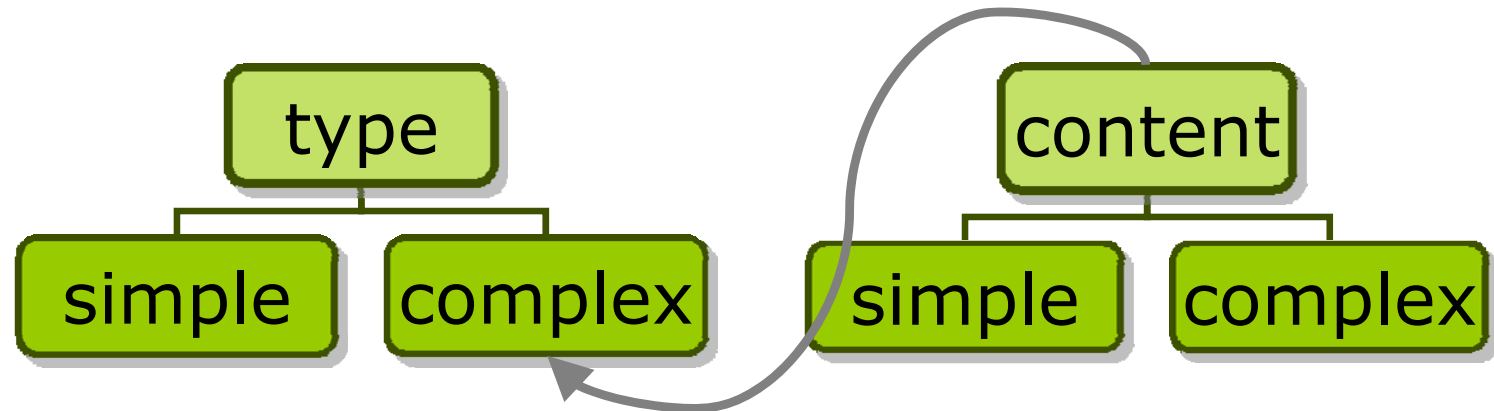
(StringWithLength)

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

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

```
<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



<u>Elemente:</u>	nein	ja	nein	ja
<u>Attribute:</u>	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>
```

StringWithLength

StringWithCompactLength

- 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 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>
```

<BookStore>

Instanz

<Book>

BookType

</Book>

...

</BookStore>

```
<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 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>
```

- **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>
```

- 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"/>
```

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



<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 `targetNamespace`
- „`targetNamespace`“ erlaubt Elemente aus `targetNamespace`



Attribut-Deklarationen

- 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>
```

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

- **use="optional"** Attribut optional
- **use="required"** Attribut obligatorisch
- **use="prohibited"** Attribut unzulässig
(Vererbung vom komplexen Elterndatentyp unterbinden)
- Wenn nichts anderes angegeben, ist das Attribut optional!

- **default:** Standard-Wert für das Attribut

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`



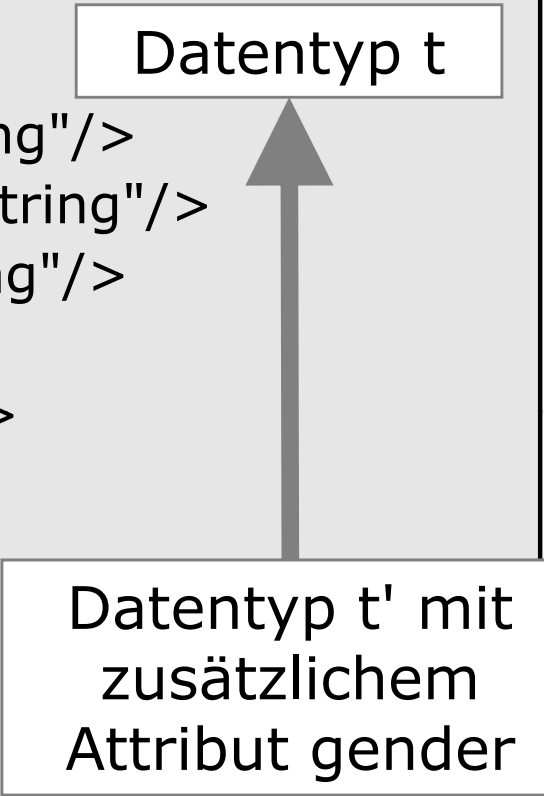
Typsubstitution

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

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>
```


- 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.

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

- **<xsd:include> und <xsd:import>**
 - immer vor allen anderen Komponenten
 - immer Kinder des Wurzelements <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 = "..."/>
  <xsd:element name="...">
    ...
  </xsd:element>
</xsd:schema>
```

- 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/>