

XML-Schema im Detail

letzte Woche

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

heutige Vorlesung

- XML-Schema
 - Datentypen - Einführung
 - Element- und Attribut-Deklarationen
 - Datentypen in Detail
 - Typsubstitution

XML Schema

- eine XML basierte Alternative für ein DTD
- beschreibt die Struktur eines XML Dokuments.
- eine W3C Recommendation
- statt XML Schema wird oft die Abkürzung XSD (XML Schema Definition) benutzt
- definiert
 - Elemente/Attribute , die im Dokument vorkommen dürfen
 - die Reihenfolge & Anzahl der (Kinder-)Elemente
 - wie Inhalt eines Element auszusehen hat
 - Datentypen für Elemente und Attribute

- + Vielzahl von vordefinierten Datentypen
- + keine eigene Syntax, sondern selbst XML-Sprache
- + Vererbungshierarchien
- + unterstützen Namensräume
- + Reihenfolgeunabhängige Strukturen einfach zu definieren
- + eigene Datentypen definierbar

Datentypen

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

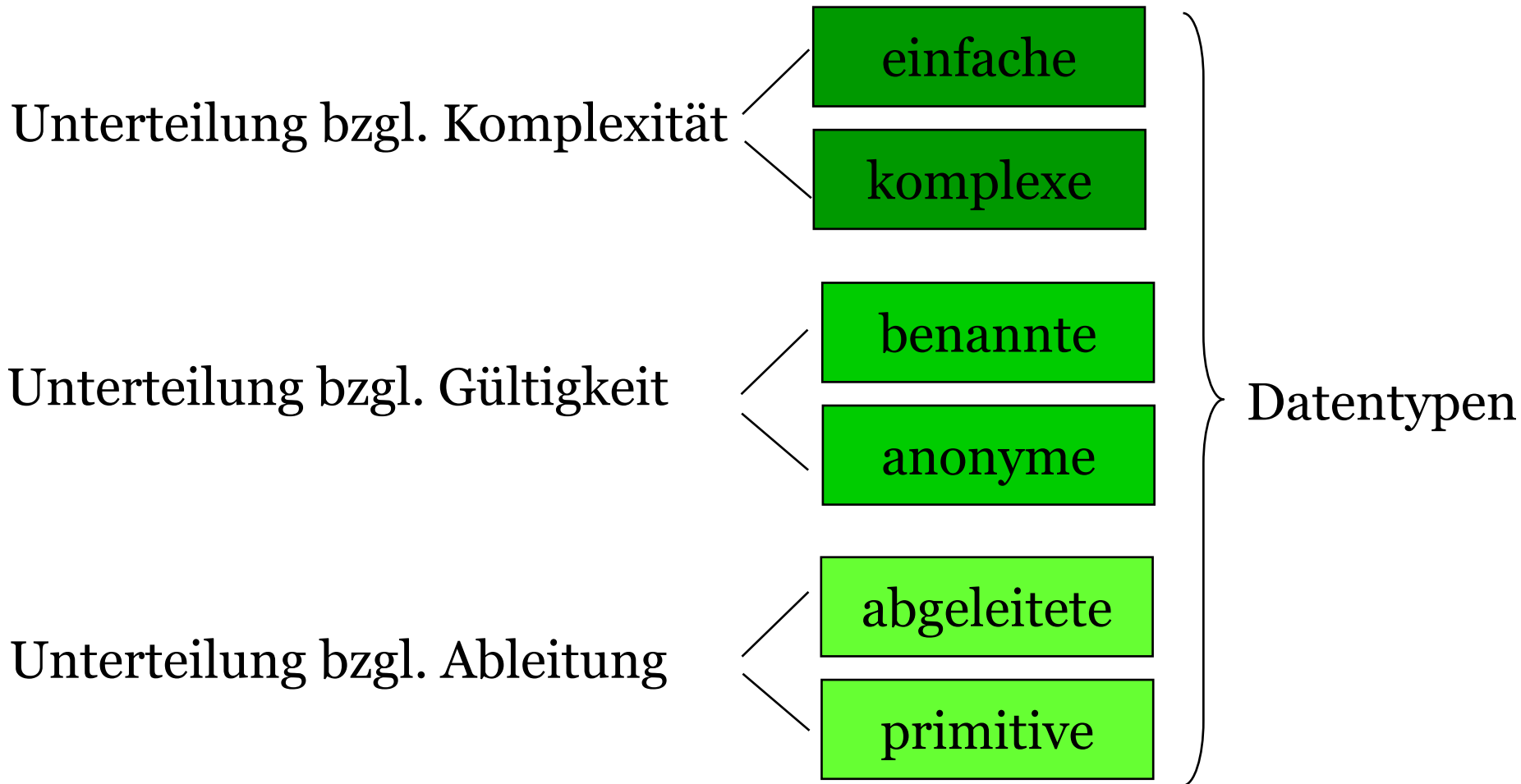
- z.B. gültigen Inhalt von latitude, longitude, uncertainty und units
- aber auch gültigen Inhalt von location

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**.

Verschiedene Arten von Datentypen



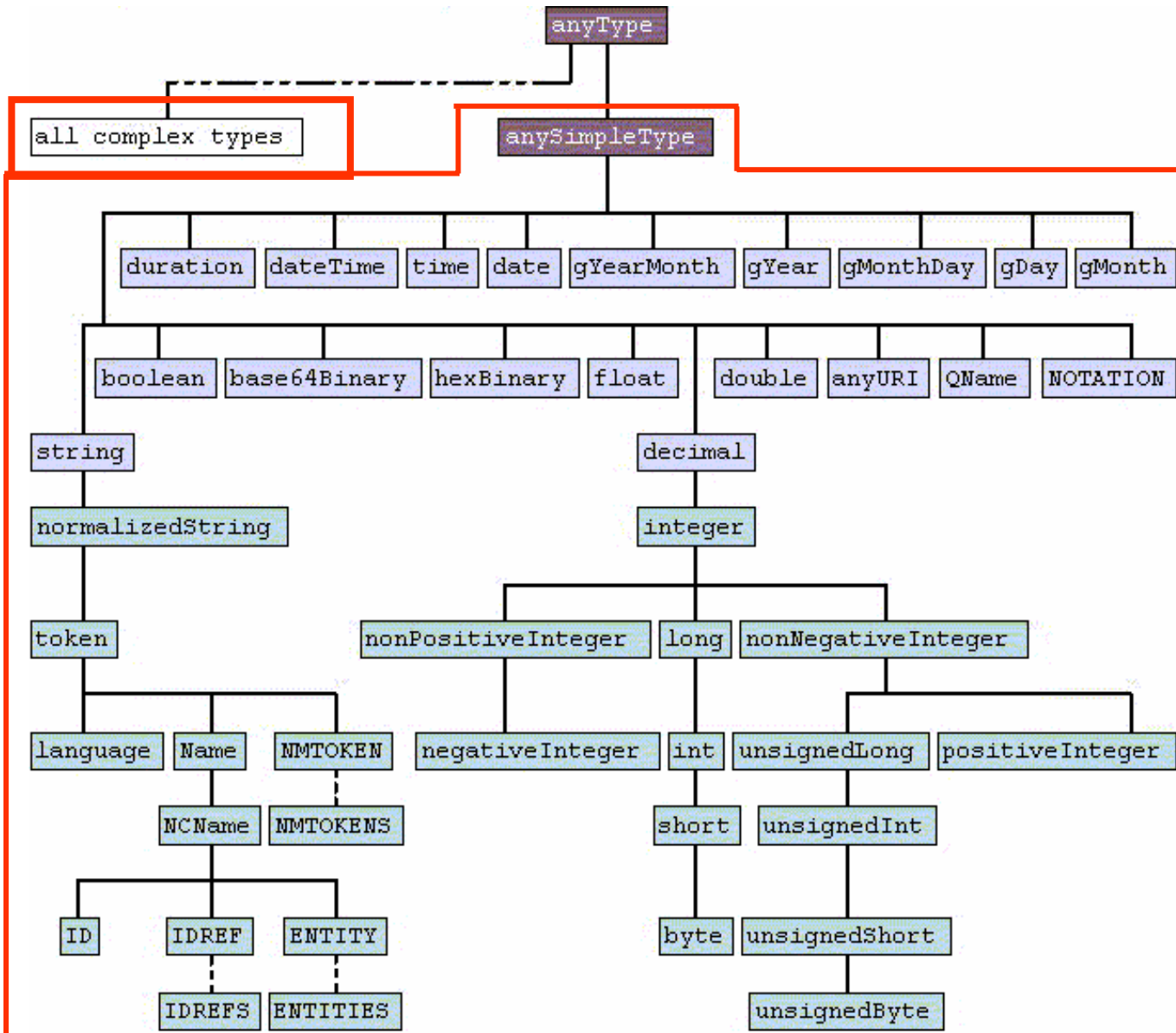
einfache Datentypen (simple types)

- beschreiben unstrukturierten Inhalt ohne Elemente oder Attribute (PCDATA)

komplexe Datentypen (complex types)

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

Hierarchie der Datentypen



- Urtyp
- vordefinierter einfacher Typ
- vordefinierter abgeleiteter Typ
- komplexer Typ
- abgeleitet durch Einschränkung
- - - - abgeleitet durch Auflistung
- · — abgeleitet durch Erweiterung oder Einschränkung

Anonyme vs. benannte Datentypen

```
<xsd:element name="BookStore">
```

```
  <xsd:complexType>
```

```
    Liste von Büchern
```

```
  </xsd:complexType>
```

```
</xsd:element>
```

- anonymer komplexer Datentyp
- lokale Deklaration

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

```
  Liste von Büchern
```

```
</xsd:complexType>
```

- benannter komplexer Datentyp
- globale Deklaration
- wiederverwendbar

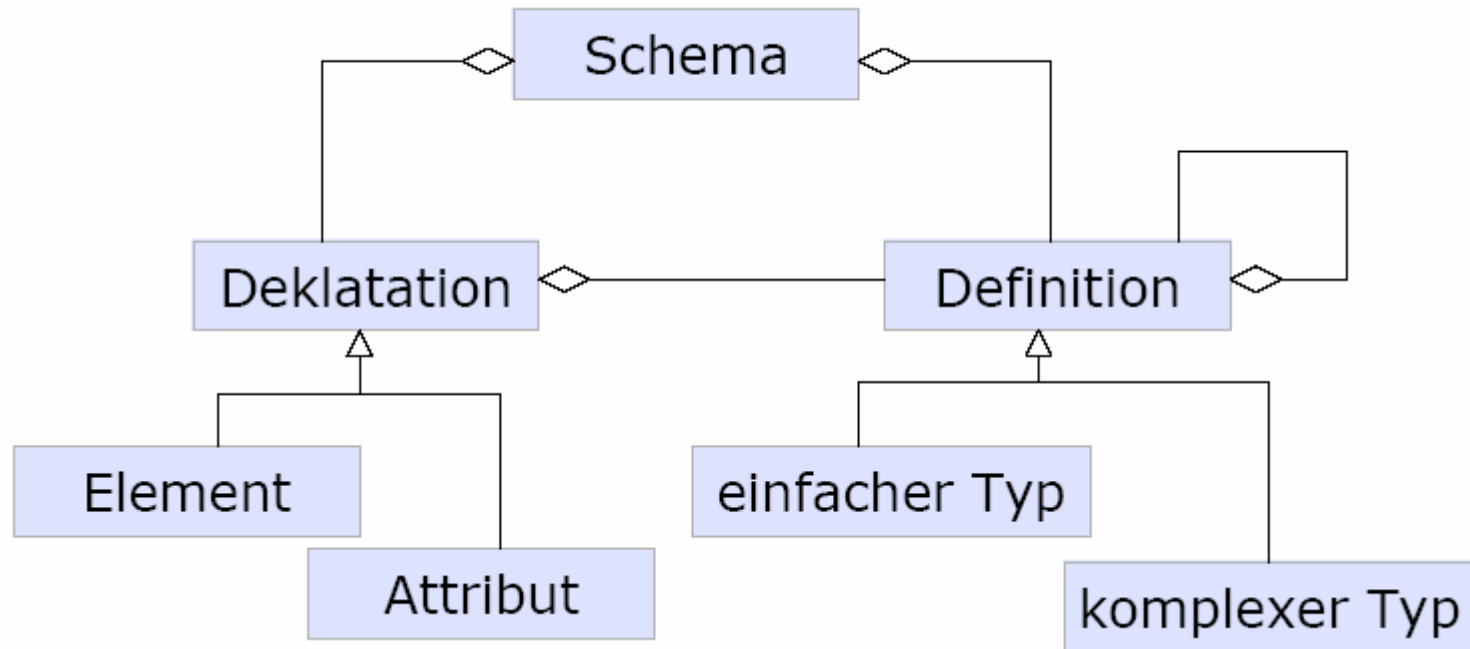
Deklaration vs. Definition

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

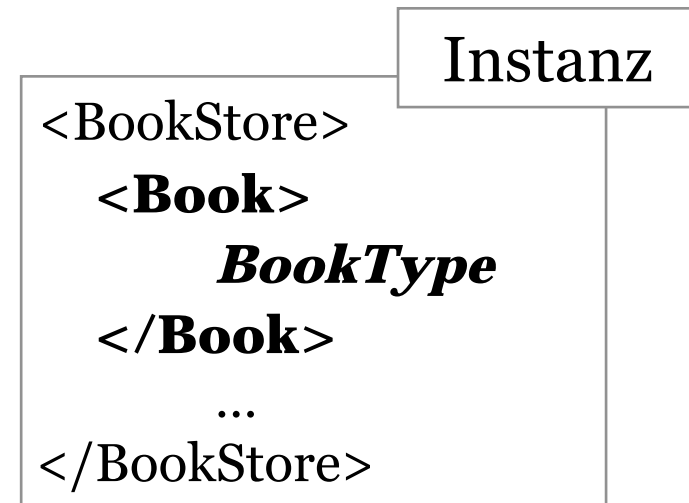


Element- Deklarationen & Definitionen

Element-Deklaration: 1. Möglichkeit

- 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>
```



Element-Deklaration: 1. Möglichkeit

```
<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

- 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 kann entweder

- komplex oder

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

} Definition

- einfach sein


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

} Definition

- Eine Element-Deklaration kann entweder ein **type** Attribut haben oder eine **anonyme Typdefinition** enthalten → Nicht 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 Elemente
- aber nur **einfache Datentypen** erlaubt

- Deklaration mit **benanntem Datentyp**:

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

globaler Typ

- oder mit **anonymem Datentyp**:

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

lokaler Typ

```
<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
- `fixed`: Standard-Fix-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

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

Erinnerung: globale Attribute in DTDs nicht möglich

Einfache vs. komplexe Datentypen

Kategorien von Datentypen

primitive

abgeleitete

einfache

xsd:string

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

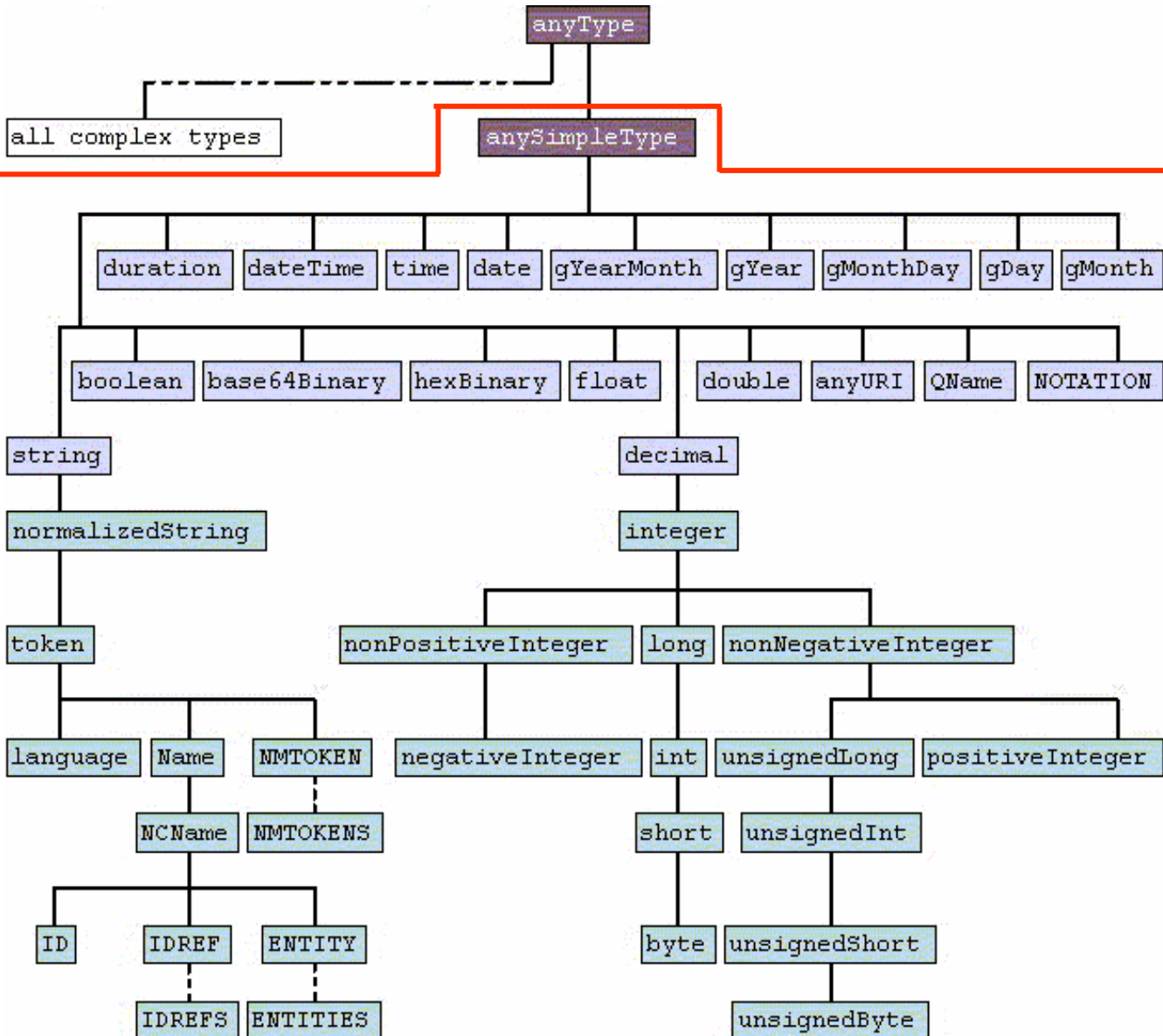
komplexe

```
<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

Hierarchie der Datentypen



- Urtyp
- vordefinierter einfacher Typ
- vordefinierter abgeleiteter Typ
- komplexer Typ
- abgeleitet durch Einschränkung
- - - - abgeleitet durch Auflistung
- · — abgeleitet durch Erweiterung oder Einschränkung

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



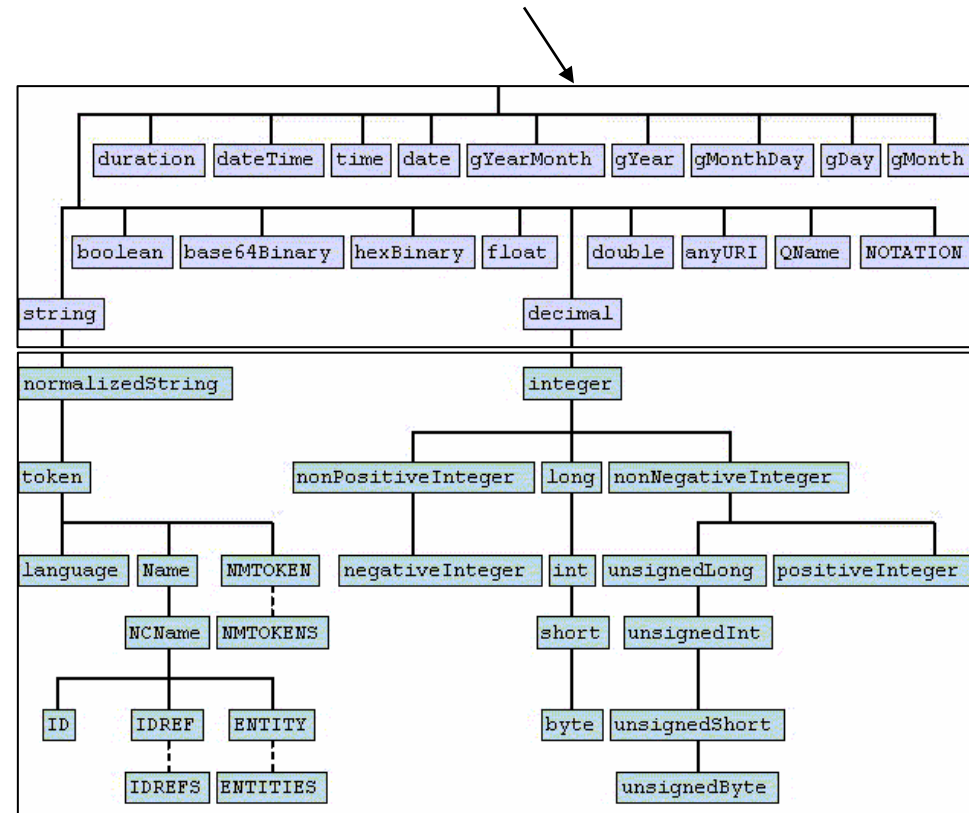
primitive Datentypen (primitive types)

- nicht von anderen Datentypen abgeleitet

abgeleitete Datentypen (derived types)

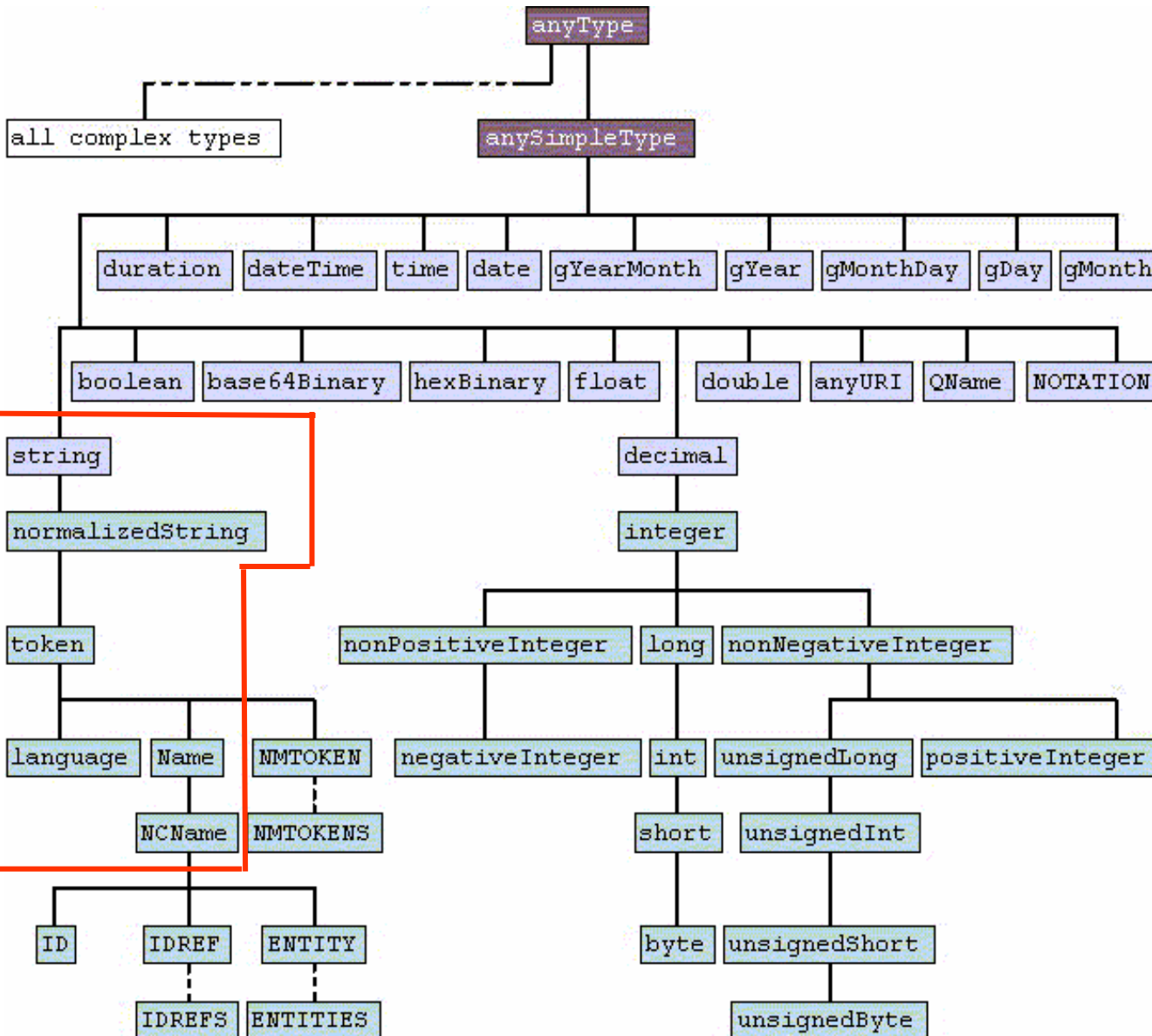
- auf Basis von anderen Datentypen definiert, z.B. durch Einschränkung oder Erweiterung

Primitive einfache Datentypen

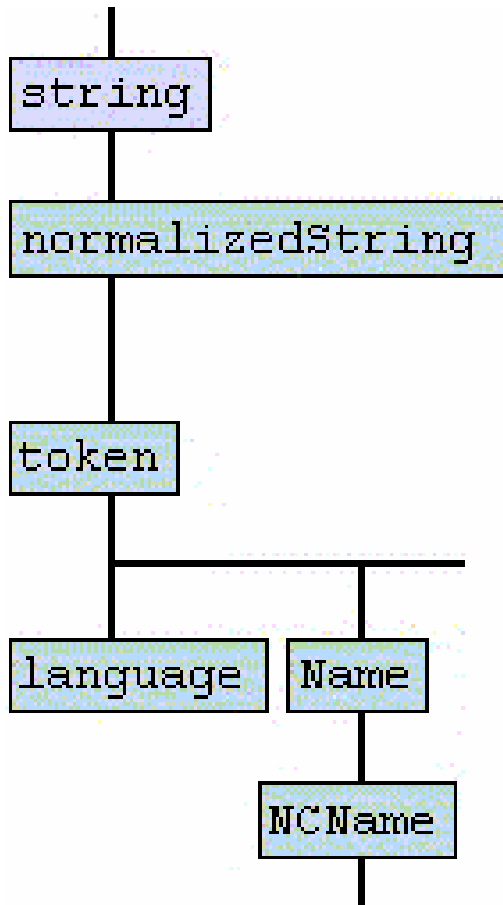


Abgeleitete einfache Datentypen

Hierarchie der Datentypen



- Urtyp
- vordefinierter einfacher Typ
- vordefinierter abgeleiteter Typ
- komplexer Typ
- abgeleitet durch Einschränkung
- - - - abgeleitet durch Auflistung
- · — abgeleitet durch Erweiterung oder Einschränkung



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

Kategorien von Datentypen

	primitive	abgeleitete
einfache	xsd:string	<pre><xsd:simpleType name="longitudeType"> <xsd:restriction base="xsd:integer"> <xsd:minInclusive value="-180"/> <xsd:maxInclusive value="180"/> </xsd:restriction> </xsd:simpleType></pre>
komplexe	<pre><xsd:complexType> <xsd:sequence> ... </xsd:sequence> </xsd:complexType></pre>	<pre><xsd:complexType name="BookTypeWithID"> <xsd:complexContent> <xsd:extension base="BookType"> <xsd:attribute name="ID" type="xsd:token"/> </xsd:extension> </xsd:complexContent> </xsd:complexType></pre>

1. Einschränkung (Teilmenge)

Einschränkung des Wertebereiches eines einfachen Datentyps

2. Vereinigung

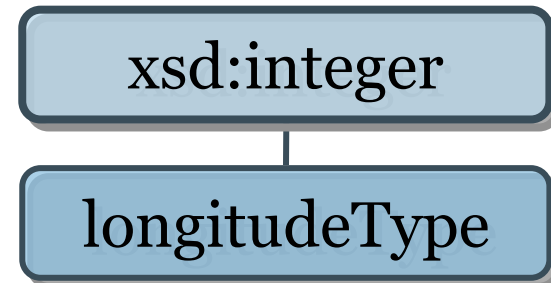
Vereinigung der Wertebereiche mehrerer einfacher Datentypen

3. Listen

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

1. Einschränkung (Teilmenge)

```
<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: $n \geq -180$, $n \leq 180$ }
- 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

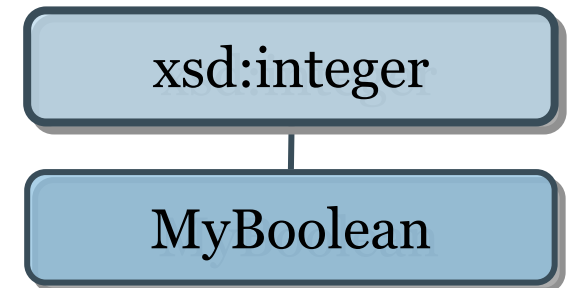
Zulässige Einschränkungen

- 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
- maxLength: max. 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>
```



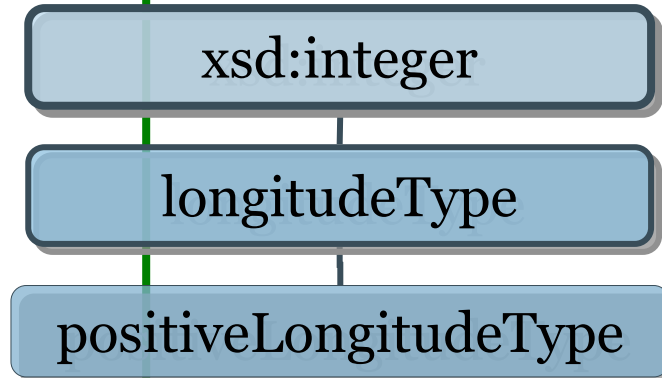
hier disjunktiv
verknüpft!

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

longitudeType erbt zulässige Einschränkungen von ursprünglichen Datentyp `xsd:integer`.

2. Vereinigung

```
<xsd:simpleType name="MyInteger">
```

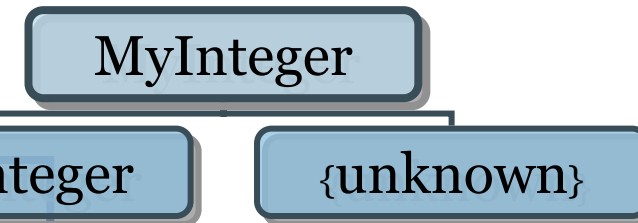
```
<xsd:union>
```

```
<xsd:simpleType>  
  <xsd:restriction base="xsd:integer"/>  
</xsd:simpleType>
```

```
<xsd:simpleType>  
  <xsd:restriction base="xsd:string">  
    <xsd:enumeration value="unknown"/>  
  </xsd:restriction>  
</xsd:simpleType>
```

```
</xsd:union>
```

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



```
MyInteger =  
  xsd:integer U { s aus xsd:string: s = unknown }
```

3. Listen

```
<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

```
<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>
```

Struktur von xsd:simpleType

```
<xsd:simpleType name="MyInteger">
```


```
<xsd:union>
```

```
<xsd:simpleType>
```

```
<xsd:restriction base="xsd:integer"/>
```

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

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



```
<xsd:simpleType>
```

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

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

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

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

```
</xsd:union>
```

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

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

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.

komplexe Datentypen

- bilden / beschreiben
 1. Sequenz
 2. Menge
 3. Auswahl

- ableiten
 1. Erweiterung
 2. Teilmenge

Komplexe Datentypen bilden

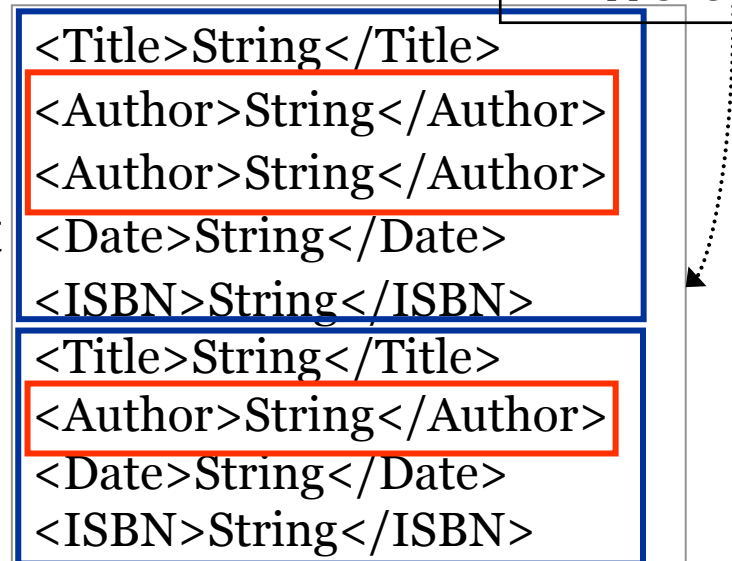
Kategorien von Datentypen

	primitive	abgeleitete
einfache	xsd:string	<pre><xsd:simpleType name="longitudeType"> <xsd:restriction base="xsd:integer"> <xsd:minInclusive value="-180"/> <xsd:maxInclusive value="180"/> </xsd:restriction> </xsd:simpleType></pre>
komplexe	<pre><xsd:complexType> <xsd:sequence> ... </xsd:sequence> </xsd:complexType></pre>	<pre><xsd:complexType name="BookTypeWithID"> <xsd:complexContent> <xsd:extension base="BookType"> <xsd:attribute name="ID" type="xsd:token"/> </xsd:extension> </xsd:complexContent> </xsd:complexType></pre>

1. Sequenz

```
<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>
```

- **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>
```

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

gültiger Wert

```
<Author>String</Author>
```

```
<Title>String</Title>
```

```
<Date>String</Date>
```

```
<Publisher>String</Publisher>
```

```
<ISBN>String</ISBN>
```

Menge: minOccurs und maxOccurs

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

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

3. Auswahl

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

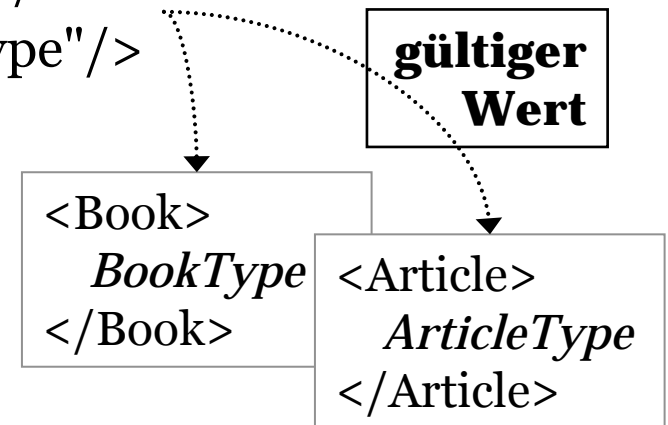
```
<xsd:choice>
```

```
<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:

```
<xsd:element name="Chap" type="ChapType"/>
```

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

```
<xsd:sequence>
```

```
<xsd:element name="Title" type="TitleType"/>
```

```
<xsd:choice maxOccurs="unbounded">
```

```
<xsd:element name="Para" type="ParaType"/>
```

```
<xsd:element name="Chap" type="ChapType"/>
```

```
</xsd:choice>
```

```
</xsd:sequence>
```

```
</xsd:complexType>
```

entspricht:

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

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

```
<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
- DTDs: (%PCDATA?, Title, %PCDATA?, ...) > in Element-Deklarationen nicht erlaubt, nur (#PCDATA|Title|...)*!

Komplexe Datentypen ableiten

Kategorien von Datentypen

	primitive	abgeleitete
einfache	xsd:string	<pre><xsd:simpleType name="longitudeType"> <xsd:restriction base="xsd:integer"> <xsd:minInclusive value="-180"/> <xsd:maxInclusive value="180"/> </xsd:restriction> </xsd:simpleType></pre>
komplexe	<pre><xsd:complexType> <xsd:sequence> ... </xsd:sequence> </xsd:complexType></pre>	<pre><xsd:complexType name="BookTypeWithID"> <xsd:complexContent> <xsd:extension base="BookType"> <xsd:attribute name="ID" type="xsd:token"/> </xsd:extension> </xsd:complexContent> </xsd:complexType></pre>

1. Erweiterung

Datentyp wird durch zusätzliche Attribute und Elemente erweitert.

2. Teilmenge

Einschränkung des Wertebereiches eines Datentyps

Erinnerung: drei Möglichkeiten einfache Datentypen abzuleiten

- 1. Einschränkung** (Teilmenge): Einschränkung des Wertebereiches eines einfachen Datentyps
- 2. Vereinigung**: Vereinigung der Wertebereiche mehrerer einfacher Datentypen
- 3. Listen**: Liste als String, wobei Elemente durch Leerzeichen getrennt sind

1. Erweiterung

- 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)

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?

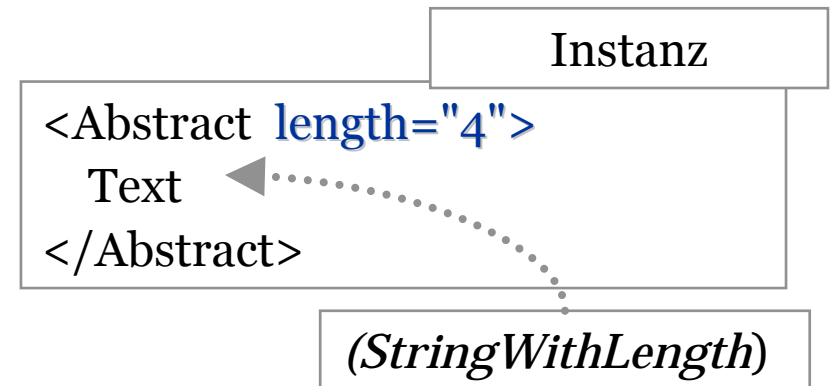
```
<xsd:complexType name="StringWithLength">
  <xsd:simpleContent>
    <xsd:extension base="xsd:string">
      <xsd:attribute name="length" type="xsd:nonNegativeInteger"/>
    </xsd:extension>
  </xsd:simpleContent>
</xsd:complexType>
```

- 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>
```

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

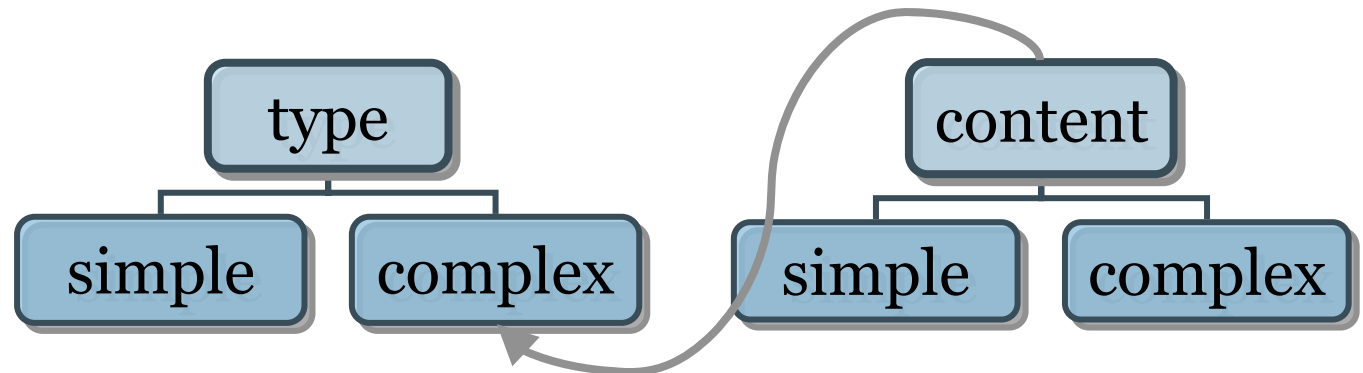


simpleContent vs. complexContent

```
<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!

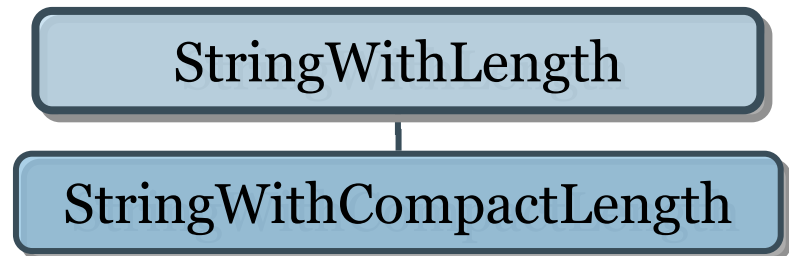


<u>Elemente:</u>	nein	ja	nein	ja
<u>Attribute:</u>	<input type="checkbox"/> nein	ja	<input type="checkbox"/> ja	ja

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

2. Teilmenge

```
<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`

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>
```

Datentyp t

```
<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' 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>
```

- 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">
```

Wie geht es weiter?

heutige Vorlesung

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

Übung nächste Woche

- DTD & XML-Schema
- Relax NG als Alternative?

Vorlesung nächste Woche

- XML-Parser