

Semistrukturierte Daten

Sommersemester 2011

Teil 4: XML-Schema

- 4.1. Überblick
- 4.2. Element-Deklarationen
- 4.3. Attribut-Deklarationen
- 4.4. Komplexe Typen
- 4.5. Simple Typen
- 4.6. Vererbung
- 4.7. weitere XML-Schema Komponenten
- 4.8. Namespaces



4.1. Überblick

- W3C Standard
- einige Merkmale von XML-Schema
- Validierung

XML-Schema Standard

- XML Schema Definition (XSD)
 - ist selbst als XML Dokument dargestellt
 - es gibt auch DTD für XSD
- W3C Recommendation 2001
 - <http://www.w3.org/TR/xmlschema-0/>
(bzw. xmlschema-1, xmlschema-2)
 - XML Schema Teil 0: "Primer" (gute Einführung!)
 - XML Schema Teil 1: Strukturen
 - XML Schema Teil 2: Datentypen

Einige Merkmale von XML-Schema

- Ist selbst in XML Syntax
- explizite Behandlung von Namespaces
- viele built-in Datentypen und Möglichkeit der Definition neuer Typen
- Typisierung auch für Elementinhalt möglich
- Einfache Codierung beliebiger Kardinalitäten
- objektorientierte Konzepte wie Vererbung
- gute Wiederverwendbarkeit und Erweiterbarkeit
- Unterstützung von "Schlüsseln" (PK, FK)

XSD-Datei(en)

- XML Schema ist immer separate Datei
- Verknüpfung Instanzdokument / Schema:
 - XML Parser "kennt" zugehöriges XML Schema
z.B.: XML-Schema Datei als Argument beim Aufruf des XML Parsers.
 - über Attribut im Wurzelement:
 - schemaLocation Attribut
 - noNamespaceSchemaLocation Attribut
- Schemata können ineinander eingebettet werden:
 - <include>: mehrere Schemata mit demselben „Target NS“
 - <redefine>: einzelne Elemente können neu definiert werden
 - <import>: mehrere Schemata mit unterschiedlichem „TargetNS“

Validierung

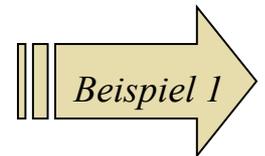
XMLLINT: (<http://xmlsoft.org/>)

- portable C Bibliothek für Linux, Unix, MacOS, Windows, ...
- Kommandozeilen-Aufruf:

```
xmlLint --schema <xsd-dateiname> <xml-dateiname>
```

DOM/SAX-Parser in Java:

- Optionen: validierend oder nicht-validierend



4.2. Element-Deklarationen

- Arten der Deklaration
- Inhaltsmodelle
- Einige Attribute bei der Element-Deklaration

Arten der Deklaration

- Global definierte Elemente
 - direkt als Kind des <schema> Wurzelementes
- Lokal definierte Elemente
 - im Kontext anderer Elemente definiert
(bzw. in complexType oder Element-Gruppe)
- Festlegen des Typs:
 - Typangabe (Attribut "type")
 - Referenz auf global definiertes Element (Attribut "ref")
 - Anonyme Typdefinition

Beispiel

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="veranstaltung">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="titel" type="xsd:string" maxOccurs="1"/>
        <xsd:element ref="schlagwort" maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

  <xsd:element name="schlagwort" type="xsd:string"/>
</xsd:schema>
```

Beispiel

global definiertes
Element

anonymer, komplexer Typ

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="veranstaltung">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="titel" type="xsd:string" maxOccurs="1"/>
        <xsd:element ref="schlagwort" maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="schlagwort" type="xsd:string"/>
</xsd:schema>
```

lokal definiertes
Element

Elementreferenz

Typangabe

benannter, simpler
Typ

Inhaltsmodelle

- beliebiges Inhaltsmodell
- leeres Inhaltsmodell
- Simpler Typ:
 - nur Zeichendaten
 - auch keine Attribute erlaubt
- Komplexer Typ:
 - enthält Elemente und/oder Attribute
 - „mixed“: Zeichendaten und Subelemente

Beliebiges Inhaltsmodell

- ohne Attribute:

```
<xsd:element name="veranstaltung" type="xsd:anyType" />
```

- Kurzschreibweise:

```
<xsd:element name="veranstaltung" />
```

(anyType ist der default, wenn kein Typ angegeben wird.)

- mit Attributen:

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

```
  <xsd:complexType>
```

```
    <xsd:complexContent>
```

```
      <xsd:extension base="xsd:anyType">
```

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

```
      </xsd:extension>
```

```
    </xsd:complexContent>
```

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

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

Leeres Inhaltsmodell

- ohne Attribute:

```
<xsd:element name="veranstaltung">  
  <xsd:complexType/>  
</xsd:element>
```

- mit Attributen:

```
<xsd:element name="veranstaltung">  
  <xsd:complexType>  
    <xsd:attribute name="jahr" type="xsd:gYear"/>  
    <xsd:attribute name="vorbesprechung"  
      type="xsd:dateTime"/>  
  </xsd:complexType>  
</xsd:element>
```

Einige Attribute bei der Elementdeklaration

- **type**: Ein vordefinierter oder anwenderdefinierter Elementtyp
- **ref**: Referenz auf globale Elementdeklaration zur Übernahme der dort spezifizierten Definitionen
- **name**: der unqualifizierte (lokale) Name
- **minOccurs**: Minimale Anzahl des Vorkommens. Default = 1.
- **maxOccurs**: Maximale Anzahl des Vorkommens. Default = 1.
- **default**: Element bekommt diesen Wert, wenn kein anderer Wert vorhanden ist.
- **fixed**: Element bekommt immer diesen Wert zugewiesen.
- **form**: (qualified, unqualified): gibt bei lokalem Element an, ob es im Target-NS ist oder nicht (überschreibt elementFormDefault)

4.3. Attribut-Deklarationen

- Arten der Deklaration
- Einige Attribute bei der Attribut-Deklaration

Arten der Deklaration

- Global definierte Attribute
 - direkt als Kind des <schema> Wurzelementes
- Lokal definierte Attribute
 - im Kontext eines complexType oder einer Attribut-Gruppe
- Festlegen des Typs
 - Typangabe (Attribut "type")
 - Referenz auf global definiertes Attribut (Attribut "ref")
 - Anonyme Typdefinition
- Bemerkung
 - Attribute haben beliebigen simplen Typ
 - Attribute als Teil eines complexType werden immer nach den Elementdeklarationen definiert

Einige Attribute bei der Attributdeklaration

- **type**: ein vordefinierter oder anwenderdefinierter Datentyp
- **ref**: Referenz auf eine andere Attributdeklaration zur Übernahme der dort spezifizierten Definitionen
- **name**: der unqualifizierte (lokale) Name
- **fixed**: Attribut bekommt immer diesen Wert zugewiesen
- **default**: Ein Attribut mit diesem Namen und diesem Wert wird ergänzt, falls das Attribut nicht angegeben wurde.
- **use**: (optional, required, prohibited): default ist "optional"
- **form**: (qualified, unqualified): gibt bei lokalem Attribut an, ob es im Target-NS ist oder nicht (überschreibt attributeFormDefault)

Beispiele

```
<xsd:attribute name="sine_tempore" default="yes"
  type="xsd:NMTOKEN" />
```

```
<xsd:attribute name="jahr" use="required"
  type="xsd:gYear" />
```

```
<xsd:attribute name="sine_tempore">
  <xsd:simpleType>
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="gestern" />
      <xsd:enumeration value="heute" />
      <xsd:enumeration value="morgen" />
    </xsd:restriction>
  </xsd:simpleType>
</xsd:attribute>
```

4.4. Komplexe Typen

- Feste Reihenfolge der Subelemente
- Auswahl eines Subelements
- Beliebige Reihenfolge der Subelemente
- Gemischter Inhalt
- simple content vs. complex content

Feste Reihenfolge der Subelemente

■ **<xsd:sequence>**

- ist der Default bei complexType wenn nichts angegeben
- Reihenfolge des Auftretens muss beachtet werden
- wie in DTDs: (...,...)
- in DTD kein min/maxOccurs, nur Verschachtelungen aus *,+,?
- min/maxOccurs auch auf sequence selbst anwendbar

■ Beispiel:

```
<xsd:complexType name="Bestand">  
  <xsd:sequence>  
    <xsd:element name="Firma" type="xsd:string"/>  
    <xsd:element name="Stichtag" minOccurs="0"  
      type="xsd:date"/>  
    <xsd:element name="Artikel" maxOccurs="unbounded"  
      type="artTyp"/>  
  </xsd:sequence>  
</xsd:complexType>
```

Auswahl eines Subelementes

■ `<xsd:choice>`

- wie in DTD: (..|..|..)
- mächtiger als in DTDs, da mit min/maxOccurs verknüpfbar
- min/maxOccurs auch auf choice selbst anwendbar
- Verschachtelungen von choice, sequence möglich

■ Beispiel

```
<xsd:choice minOccurs="0" maxOccurs="2">
```

```
  <xsd:element name="titel" type="xsd:string" maxOccurs="2"/>
```

```
  <xsd:element name="nummer" type="xsd:long" />
```

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

- ## ■ Bemerkung: So etwas ist nicht leicht in DTD ausdrückbar!

Beliebige Reihenfolge der Subelemente

■ `<xsd:all>`

- Alle Elemente in beliebiger Ordnung
- `all` kann nicht verschachtelt werden.
- Auch Schachtelung mit Sequence und Choice ist verboten.
- Für jedes Element darin muss `maxOccurs=1` und `minOccurs=0` oder `=1` gelten.

■ Beispiel

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

```
  <xsd:all>
```

```
    <xsd:element ref="Hunde"/>
```

```
    <xsd:element ref="Katzen"/>
```

```
    <xsd:element ref="Hasen"/>
```

```
  </xsd:all>
```

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

Gemischter Inhalt

- Gemischter Inhalt: Attribut **mixed** kann beliebig mit sequence, choice, minOccurs,... verknüpft werden

- Beispiel-Schema:

```
<xsd:element name="veranstaltung" type="neuerTyp"/>
<xsd:complexType name="neuerTyp" mixed="true">
  <xsd:sequence>
    <xsd:element name="titel" type="xsd:string"/>
    <xsd:element name="nummer" type="xsd:long"
      minOccurs="unbounded"/>
  </xsd:sequence>
</xsd:complexType>
```

- Beispiel-Instanzdokument:

```
<veranstaltung>Dies ist die VL <titel>Semistrukturierte
Daten </titel> mit Nummer <nummer>181135</nummer> oder
<nummer>181136</nummer>, das weiß ich nicht genau
</veranstaltung>
```

Simple Content vs. Complex Content

- Simple Content
 - nur Text Inhalt
 - keine Subelemente möglich
 - Attribute möglich bei simpleContent, aber nicht in simpleType!
 - ist default bei simpleType
- Complex Content
 - Elemente oder gemischter Inhalt
 - Attribute möglich
 - ist default bei complexType (d.h. man kann innerhalb eines complexType-Elements auf das Subelement complexContent verzichten).
- Complex Type mit Simple Content?
 - ist sinnvoll: bei einem Element mit Text-Inhalt und Attribut(en)

Beispiel

```
<xsd:complexType name="titel">
  <xsd:simpleContent>
    <xsd:extension base="xsd:string">
      <xsd:attribute name="sprache" type="xsd:NMTOKEN" />
    </xsd:extension>
  </xsd:simpleContent>
</xsd:complexType>
```

Bemerkung:

- Elemente mit reinem Text-Inhalt: eigentlich simpler Typ
- Attribute können aber nur für "complexType" definiert werden!!
- Lösung: `complexType mit simpleContent`
- Benötigt Vererbung, um Elementinhalt als simpleType zu definieren.

4.5. Simple Typen

- Überblick
- Built-in Typen
- Beispiele

Überblick

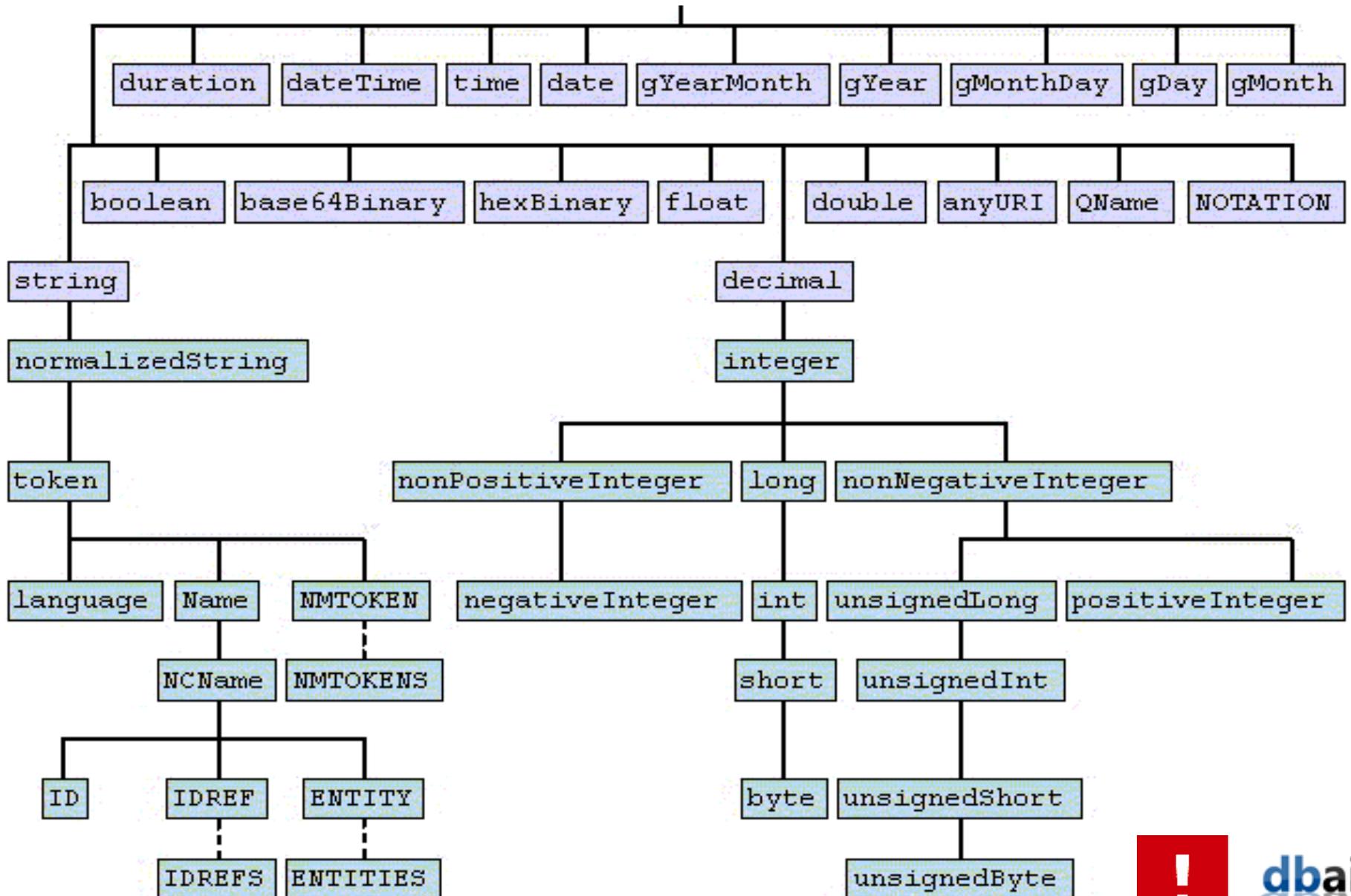
Mögliche Klassifikationen für simple Datentypen

- **Atomar vs. aggregiert**
 - atomare: bestehen aus unteilbaren Werten
 - aggregierte: listen-artige und Vereinigungstypen
- **Primitiv vs. abgeleitet**
 - primitiv: unabhängig von anderen Datentypen
 - abgeleitet: auf der Basis eines anderen Typs definiert
- **Vorgegeben vs. anwenderdefiniert**
 - vorgegeben: 44 built-in types in XML Schema
 - Der Anwender kann davon weitere Typen ableiten.

Built-In Datentypen

- primitive Datentypen, z.B.:
 - string
 - decimal, float, double, boolean
 - duration, time, dateTime, date,
 - etc.
- abgeleitete Typen, z.B.:
 - von decimal abgeleitet: integer, positiveInteger, negativeInteger, nonNegativeInteger, nonPositiveInteger, long, int, short, byte, ...
 - von string abgeleitet: normalizedString, token, Name,...
 - von token abgeleitet (für DTD Kompatibilität): ID, IDREF, IDREFS, ENTITY, ENTITIES, NMTOKEN, NMTOKENS

Hierarchie der Built-In Datentypen



Beispiele für zulässige Werte

boolean: true, false, 1, 0
float (single-precision, 32-bit) : -INF, -1E4, 12.78E-2, 12, INF, NaN
double (double-precision 64-bit): -INF, -1E4, 12.78E-2, 12, INF, NaN
language (gültige Werte für xml:lang lt. XML 1.0): en-GB, en-US, fr, ...
date: 2007-05-31
gMonth: --05--
gYear: 2007
gYearMonth: 2007-02
gDay: ---31
gMonthDay: --05-31
time: 13:20:00.000-05:00 (d.h. -5h Zeitverschiebung bzgl. UCT)
duration: P1Y2M3DT10H30M12.3S
(P = period, year, month, day, time, ...)

4.6. Vererbung

- Ableitung neuer Typen
- Restriktionen bei simplen Typen
- Listentypen
- Vereinigungstypen
- Abgeleitete built-in Typen
- Abgeleitete komplexe Typen
- Weitere Features von XML-Schema

Ableitung neuer Typen

- derived by restriction:
 - auf simple/komplexe Typen anwendbar
 - Instanz des eingeschränkten Typs B ist auch Instanz des Basistyps A.
- derived by extension:
 - neue Elemente/Attribute hinzufügen
 - Simpler Typ wird dadurch zu komplexem Typ
- derived by list:
 - Liste von Elementen eines simplen Typs
- derived by union:
 - Vereinigung von 2 oder mehr simplen Typen

Restriktionsmöglichkeiten bei simplen Typen

- heißen auch Eigenschaften bzw. **Facetten**
- `length`, `minLength`, `maxLength`
 - Beschränkung der Stringlänge
- `minInclusive`, `maxInclusive`, `minExclusive`, `maxExclusive`
 - Beschränkung des Wertebereichs bei Zahlen
- `Pattern`
 - Reguläre Ausdrücke wie in Perl
- `enumeration`
 - Auflistung aller erlaubten Werte
- `whitespace`
 - `preserve`, `replace`, `collapse`
- `totalDigits`, `fractionDigits`
 - Dezimalstellen insgesamt bzw. nach dem Komma

Aufzählungstypen

- Erlaubt: ein Wert aus einer vordefinierten Menge

```
<xsd:simpleType name="Termine">  
  <xsd:restriction base="xsd:string">  
    <xsd:enumeration value="gestern"/>  
    <xsd:enumeration value="heute"/>  
    <xsd:enumeration value="morgen"/>  
  </xsd:restriction>  
</xsd:simpleType>
```

- Bemerkung:
 - Patterns und Enumerations werden immer mit "oder" interpretiert wenn sie öfter vorkommen
 - alle anderen Einschränkungen mit "und"

Listentypen

- `<xsd:list itemType="..."/>`
- Listen können nur simple Typen enthalten
 - insbes.: Listen von Listen nicht unterstützt
- Listenelemente durch Whitespaces getrennt
- Beispiel

```
<xsd:simpleType name="Nachbarlaender">  
  <xsd:list itemType="xsd:NMTOKEN" />  
</xsd:simpleType>
```

- Instanz `"I CH FL D CZ SK H SLO"`

Vereinigungstypen

- Kombination von mehreren **simple** Typen

- entweder via **memberTypes**-Attribut

```
<xsd:simpleType name="Bsp-Union">  
  <xsd:union memberTypes="Typ1 Typ2"/>  
</xsd:simpleType>
```

- oder Typen direkt innerhalb des **union**-Elements definieren

```
<xsd:simpleType name="Size">  
  <xsd:union>  
    <xsd:simpleType>  
      <xsd:restriction base='xsd:integer'>...  
    </xsd:simpleType>  
    <xsd:simpleType>  
      <xsd:restriction base='xsd:string'>...  
    </xsd:simpleType>  
  </xsd:union>  
</xsd:simpleType>
```

Beispiele für abgeleitete built-in Typen

Einschränkung mittels min/maxInclusive Facette:

```
<xsd:simpleType name="short">  
  <xsd:restriction base="xsd:int">  
    <xsd:minInclusive value="-32768"/>  
    <xsd:maxInclusive value="32767"/>  
  </xsd:restriction>  
</xsd:simpleType>
```

```
<xsd:simpleType name="positiveInteger">  
  <xsd:restriction base="xsd:nonNegativeInteger">  
    <xsd:minInclusive value="1"/>  
  </xsd:restriction>  
</xsd:simpleType>
```

Abgeleitete komplexe Typen

■ Restriction:

- Einschränkungen des Wertebereichs, z.B.:
Komponententyp einschränken, optionale Komponenten weglassen, min/max-Occurs einschränken, etc.
- eingeschränkter Typ ist Teilmenge des Basistyps
- Alle Komponenten (Subelemente, Attribute), die enthalten sein sollen, müssen noch einmal explizit angegeben werden.

■ Extension:

- Hinzufügen von Elementen/Attributen
- Basistyp kann simpler oder komplexer Typ sein.
(simpler Typ wird dadurch zu komplexem Typ)
- Bei Erweiterung eines komplexen Typs: Man muss nur jene Komponenten angeben, die neu dazukommen.

Beispiel

■ Restriction

```
<xsd:complexType name="Bestand">
  <xsd:sequence>
    <xsd:element name="Firma" type="xsd:string"/>
    <xsd:element name="Stichtag" minOccurs="0" type="xsd:date"/>
    <xsd:element name="Artikel" maxOccurs="unbounded" type="artTyp"/>
  </xsd:sequence>
</xsd:complexType>
```

```
<xsd:complexType name="BestandNeu">
  <xsd:restriction base="Bestand">
    <xsd:sequence>
      <xsd:element name="Firma" type="xsd:string"/>
      <xsd:element name="Artikel" maxOccurs="100" type="artTyp"/>
    </xsd:sequence>
  </xsd:restriction>
</xsd:complexType>
```

Beispiel

■ Extension

```
<xsd:complexType name="Address">
  <xsd:sequence>
    <xsd:element name="name" type="xsd:string"/>
    <xsd:element name="street" type="xsd:string"/>
    <xsd:element name="city" type="xsd:string"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="US-Address">
  <xsd:extension base="Address">
    <xsd:sequence>
      <xsd:element name="state" type="US-State"/>
      <xsd:element name="zip" type="xsd:positiveInteger"/>
    </xsd:sequence>
  </xsd:extension>
</xsd:complexType>
```

Weitere Features von XML-Schema

- Ersetzungen im Instanz-Dokument
 - **Substitutionsgruppen**: Überall, wo Element A erwartet wird, darf auch Element B verwendet werden.
 - **Ableitung**: Überall, wo Typ A erwartet wird, darf auch von A abgeleiteter Typ verwendet werden.
- Attribute zur Steuerung der Vererbung/Substitution
 - **abstract**: Abstraktes Element wird nie im Instanz-Dokument verwendet
 - **block**: Substitutionen verbieten
 - **final**: keine weiteren Ableitungen von diesem Typ erlaubt
 - **fixed**: Facette darf nicht weiter verändert werden.

4.7. weitere XML-Schema Komponenten

- Element-Gruppen
- Attribut-Gruppen
- Integritätsbedingungen
- Dokumentation im Schema

Elementgruppen

■ Elementgruppe definieren

- müssen global deklariert werden: `xsd:group` mit Attribut "name"
- keine Attributdeklarationen erlaubt

```
<xsd:group name="namensgruppe">  
  <xsd:sequence>  
    <xsd:element name="Vorname" type="xsd:string"/>  
    <xsd:element name="Nachname" type="xsd:string"/>  
  </xsd:sequence>  
</xsd:group>
```

■ Verwendung einer Elementgruppe

- in `complexType` oder anderer Elementgruppe
- Referenzierung mittels `xsd:group` mit Attribut "ref"

```
<xsd:complexType name="Kunde">  
  <xsd:sequence>  
    <xsd:element name="Firma" type="xsd:string"/>  
    <xsd:group ref="namensgruppe"/>  
  </xsd:sequence>  
</xsd:complexType>
```

Attributgruppen

- Attributgruppen
 - müssen global deklariert werden: `xsd:attributeGroup` mit Attribut "name"

```
<xsd:attributeGroup name="namensgruppe">  
  <xsd:attribute name="Vorname" type="xsd:string"/>  
  <xsd:attribute name="Nachname" type="xsd:string"/>  
</xsd:attributeGroup>
```

- Verwendung einer Attributgruppe
 - in `complexType`, Elementdeklaration, Attributgruppe
 - Referenzierung mittels `xsd:attributeGroup` mit Attribut "ref"

```
<xsd:complexType name="Kunde">  
  <xsd:sequence>  
    <xsd:element name="KundenNummer" type="xsd:integer"/>  
    <xsd:element name="Firma" type="xsd:string"/>  
  </xsd:sequence>  
  <xsd:attributeGroup ref="namensgruppe"/>  
</xsd:complexType>
```

Integritätsbedingungen

- **unique**: Elementinhalt bzw. Attributwert muss eindeutig sein:
 - und zwar innerhalb einer Menge von Elementen (die mittels XPath-Ausdruck bestimmt werden)
 - vergleichbar mit UNIQUE-constraint in relationaler DB
- **key**: Elementinhalt bzw. Attributwert muss eindeutig und tatsächlich vorhanden sein:
 - ähnlich wie unique (aber Wert darf nicht ausgelassen werden)
 - vergleichbar mit Primärschlüssel in relationaler DB
- **keyref**: Verweis auf einen key-Wert:
 - XML-Parser überprüft referentielle Integrität
 - vergleichbar mit Fremdschlüssel in relationaler DB

Beispiel: key/keyref

```
<xsd:element name="Lehre" type="LehreTyp">
  <xsd:key name="veranstaltungsnummer">
    <xsd:selector xpath="veranstaltung" />
    <xsd:field xpath="nummer" />
  </xsd:key>
  <xsd:keyref name="referenzname" refer="veranstaltungsnummer">
    <xsd:selector xpath="termine/termin" />
    <xsd:field xpath="lvanummer" />
  </xsd:keyref>
</xsd:element>
```



```
<Lehre>
  <veranstaltung><name>...</name><nummer>13</nummer></veranstaltung>
  <veranstaltung><name>...</name><nummer>17</nummer></veranstaltung>
  <termine>
    <termin><datum>...<datum><lvanummer>17</lvanummer></termin>
    <termin><datum>...<datum><lvanummer>13</lvanummer></termin>
    <termin><datum>...<datum><lvanummer>17</lvanummer></termin>
  </termine>
</Lehre>
```

Dokumentation im Schema

- Es gibt dafür 3 spezielle Elemente:
 - `annotation`: top-level element (direkt unter `<schema>`) oder auch in Element-, Attribut-, ...Definitionen
 - Subelemente von `annotation`: (reine Text-Elemente)
 - ◆ `documentation`
 - ◆ `appInfo`: zusätzliche Information für Applikation

- Beispiel:

```
<xs:annotation>
```

```
  <xs:appInfo>SSD Unterlagen</xs:appInfo>
```

```
  <xs:documentation xml:lang="de">
```

```
    Diese Schema definiert ein SSD-Vorlesungsskriptum
```

```
  </xs:documentation>
```

```
</xs:annotation>
```

4.8. Namespaces

- Namespaces in XML-Schema
- Schema Definition ohne Target NS
- Schema Definition mit Target NS
- Schema Definitionen mit mehreren Target NS

Namespaces in XML Schema

- NS-Deklaration für **Schema-NS** im Root Element einer Schema Definition (üblicher Präfix "xsd" (oder "xs")) :

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
```

- NS-Deklaration im Instanzdokument (optional):

- `xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"`

- Deklaration des Namespaces in dem Elemente/Attribute des Instanzdokuments liegen: **Target-NS** (jedoch nur 1 Target NS pro XSD-Datei!):

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.example.com/contact"
  xmlns:co="http://www.example.com/contact">
```

```
  <xsd:element name="contact" type="co:ContactType"/>
```

```
  <xsd:complexType name="ContactType">...</xsd:complexType>
```

```
</xsd:schema>
```

Schema-Definition ohne Target Namespace

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="contact" type="ContactType"/>
  <xsd:complexType name="ContactType">
    <xsd:sequence>
      <xsd:element name="customer" type="USAddress"/>
      <xsd:element name="organization" type="USAddress"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="USAddress">
    <xsd:sequence> ... </xsd:sequence>
    <xsd:attribute name="country" type="xsd:NMTOKEN"
      fixed="US"/>
  </xsd:complexType>
</xsd:schema>
```



Referenzierung eines Schemas ohne Target-NS

Mittels Attribut `noNamespaceSchemaLocation`

```
<?xml version="1.0"?>
```

```
<contact
```

```
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
  xsi:noNamespaceSchemaLocation="contact.xsd">
```

```
  <customer country="US">
```

```
    <name>B.Gates</name>
```

```
    <street>123 Main Street</street>
```

```
    <city>Redmond</city>
```

```
    <state>WA</state>
```

```
    <zip>98052</zip>
```

```
  </customer>
```

```
  <organization country="US">
```

```
    <name>Microsoft Corporation</name>
```

```
    . . . .
```

```
  </organization>
```

```
</contact>
```

Schema-Definition mit Target Namespace

- Global (d.h. „Top-Level“) definierte Komponenten
 - d.h. direkte Subelemente von `<schema>`
 - sind automatisch im Target NS
 - müssen daher "qualifiziert" (d.h.: mit Präfix) referenziert werden.
- Lokale Komponenten
 - sind default-mäßig im leeren NS und
 - werden daher "unqualifiziert" (d.h.: ohne Präfix) verwendet
- Änderung dieses Default-Verhaltens
 - Global: mit den Attributen `elementFormDefault` bzw. `attributeFormDefault` des `<schema>` Elements
 - Lokal: mit dem Attribut `form` für Elemente/Attribute
 - Werte dieser Attribute: `qualified` oder `unqualified`

Schema-Definition mit Target Namespace

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.example.com/contact"
  xmlns:co="http://www.example.com/contact">
  <xsd:element name="contact" type="co:ContactType"/>
  <xsd:complexType name="ContactType">
    <xsd:sequence>
      <element name="customer" type="co:USAddress"/>
      <element name="organization" type="co:USAddress"/>
    </xsd:sequence>
  </xsd:complexType>

  <xsd:complexType name="USAddress">
    ...
  </xsd:complexType>
</xsd:schema>
```



Referenzierung eines Schemas mit Target-NS

Mittels Attribut `schemaLocation`

```
<?xml version="1.0"?>
```

```
<co:contact
```

```
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
  xmlns:co="http://www.example.com/contact"
```

```
  xsi:schemaLocation="http://www.example.com/contact  
    contact.xsd">
```

```
  <customer country="US">
```

```
    <name>B.Gates</name>
```

```
    <street>123 Main Street</street>
```

```
    <city>Redmond</city>
```

```
    ...
```

```
  </customer>
```

```
  ...
```

```
</co:contact>
```



Beispiel: Änderung des FormDefaults

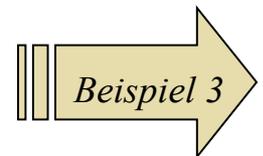
■ im Schema

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.example.com/contact"
xmlns:co="http://www.example.com/contact"
elementFormDefault="qualified"
attributeFormDefault="qualified">
<xsd:element name="contact" type="co:ContactType"/>
<xsd:complexType name="ContactType">
  <xsd:sequence>
    <xsd:element name="customer" type="co:USAddress"/>
    <xsd:element name="organization" type="co:USAddress"/>
  </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="USAddress">
  <xsd:element ... form="unqualified"/>
</xsd:complexType>
</xsd:schema>
```

Beispiel: Änderung des FormDefaults

- im Instanz-Dokument:

```
<co:contact
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:co="http://www.example.com/contact"
  xsi:schemaLocation="http://www.example.com/contact
    contact.xsd">
  <co:customer co:country="US">
    <name>B.Gates<name>
    <street>123 Main Street<street> ...
    <city>Redmond<city>
    <state>WA<state>
    <zip>98052<zip>
  </co:customer>
  ...
</co:contact>
```



Schema und mehrere Namespaces

- Problem: Pro XSD-Datei nur ein Target NS
- Lösung: Mehrere XSD-Dateien notwendig! Verknüpfung
 - mittels `<xsd:any namespace="NS">` :
 - ◆ an sich uneingeschränkter Inhalt; mit Attribut `processContent` steuerbar
 - besser: `<xsd:import>` und Elementreferenz verwenden:

```
<xsd:schema ... xmlns:co="http://www.example.com/contact"
  xmlns:org="www.ns.org"
  targetNamespace="http://www.example.com/contact"/>
<xsd:import namespace="www.ns.org"
  schemaLocation="org.xsd" />
<!-- TargetNS in org.xsd: www.ns.org -->
...
<xsd:element ref="org:organization"/>
```

