

# XML-Schema (SCH)

## **Lernziele**

- Sie erhalten einen Überblick über alle für XML-Schema relevanten Spezifikationen und Werkzeuge.
- Sie kennen die Vorteile von XML-Schema gegenüber DTDs.
- Sie wissen, wie Elemente und Attribute in XML-Schema deklariert werden.
- Sie kennen die eingebauten Datentypen von XML-Schema.
- Sie erfahren, wie Sie eigene Datentypen mit XML-Schema definieren.
- Sie wissen, wie Sie ihre Schemata dokumentieren können.
- Sie sind in der Lage, Inhaltsmodelle mit XML-Schema zu definieren.
- Sie wissen, wie Sie mit XML-Schema Namensräume bevölkern können.

# Überblick und Einführung

## Was sind Schemata?

Ein **Schema** ist ein Modell für die Beschreibung der Struktur von Information.

Der Ausdruck Schema stammt ursprünglich aus dem Bereich der Datenbanken, wo er für die Beschreibung der Struktur von Daten in relationalen Tabellen steht.

Das Schema einer validierbaren XML-Datei wird mit Hilfe der DTD beschrieben.

## Was ist XML-Schema?

**XML-Schema** ist eine XML-Anwendung, mit denen Schemata zur Beschreibung der Struktur von Information erstellt werden können.

Der Zweck eines XML-Schemas besteht wie bei DTDs darin, die zulässige Struktur von XML-Dokumenten zu beschreiben.

Vereinfacht ausgedrückt ist XML-Schema die verbesserte Version der DTD-Syntax.

## Warum XML-Schema?

- DTDs bieten nur sehr begrenzte Möglichkeiten für die Beschreibung und den Einsatz von Schemainformationen.  
Z.B.: Mit DTDs alleine können Sie weder erzwingen, dass der Inhalt eines `<datum>`-Elements ein Datum ist, noch dafür sorgen, dass ein Text in einem Eingabefeld mindestens zehn und höchstens zwanzig Zeichen enthält.
- DTDs erfordern das Lernen einer neuen Syntax zusätzlich zur XML-Syntax.

=> XML-Schema soll diese Schwächen von DTDs beheben. XML-Schema ist der designierte Nachfolger der DTDs

## DTD versus XML-Schema

	<b>DTD</b>	<b>XML Schema</b>
<i>Datentypisierung</i>	<p>schwach</p> <p>im wesentlichen Zeichenketten (CDATA), Name Token und IDs</p>	<p>stark</p> <ul style="list-style-type: none"> <li>• alle Datentypen, die von modernen Programmiersprachen unterstützt werden (Zeichenketten, Datum, Zahlen...)</li> <li>• eigene komplexe Datentypen können definiert werden</li> </ul>
<i>Inhaltsmodelle</i>	<p>schwach</p> <ul style="list-style-type: none"> <li>• eine einfache Sequenz oder Auswahlliste, die nicht bei gemischtem Inhalt zur Verfügung steht, kann nur keine, eine oder viele Vorkommnisse eines Elements spezifizieren</li> <li>• keine benannten Element- oder Attributgruppen</li> </ul>	<p>stark</p> <ul style="list-style-type: none"> <li>• detailliertere Inhaltsmodelle, die auch bei gemischtem Inhalt zur Verfügung stehen</li> <li>• Festlegung der exakten Anzahl der Vorkommnisse möglich</li> <li>• benannte Element- oder Attributgruppen</li> </ul>
<i>Vererbung</i>	nein	ja
<i>Namen</i>	nur globale Namen	globale und lokale Namen
<i>Syntax</i>	EBNF + Pseudo-XML	reines XML
<i>Unterstützung mehrerer Schemata für ein Dokument</i>	<p>nein</p> <p>eine DTD pro Dokument</p>	<p>ja</p> <p>so viele wie nötig, basierend auf XML-Namensräumen</p>
<i>Tools / Implementierung</i>	Tools müssen zwei verschiedene Syntaxarten un-	Tools müssen nur XML-Syntax unterstützen

	<b>DTD</b>	<b>XML Schema</b>
<i>Unterstützung</i>	unterstützen	Syntax unterstützen
<i>DOM-Unterstützung</i>	keine	kann angezeigt und manipuliert werden mittels DOM
<i>Erweiterbarkeit</i>	limitiert die XML 1.0 Empfehlung (und SGML) müssen geändert werden, um DTD-Syntax zu erweitern	unlimitiert basiert auf XML
<i>Legacy-beschränkungen</i>	ja viele nicht gerade ideale Aspekte von XML stammen aus der Anforderung, Rückwärtskompatibilität mit SGML zu wahren	nein obwohl einige Datentypen der DTDs behalten wurden
<i>Dynamische Schemata</i>	nein DTDs sind in praktisch allen Anwendungen lediglich lesbar.	ja Schemata können zur Laufzeit ausgewählt und modifiziert werden, z.B. als Ergebnis einer Benutzeraktion.

## XML-Schema-Spezifikationen

### ***XML Schema Part 0: Primer***

Eine lesenswerte, beispielunterstützte Einführung in den Standard. Sie ist nicht normativ, also kein Teil des eigentlichen Standards.

### ***XML Schema Part 1: Structures***

Beschreibt die Konstrukte, die für die Definition der Struktur von XML-Dokumenten verfügbar sind. Dieser Teil von XML-Schema erweitert die Grundfunktionalität von DTDs für diese Aufgabe.

### ***XML Schema Part 2: Datatypes***

Definiert ein Repertoire von Standard-Datentypen und Mechanismen, um eigene Datentypen zu definieren. Das hier festgelegte Typsystem ist eng auf viele Standards in der Internetwelt abgestimmt.

## Welche Tools unterstützen XML-Schema?

Einige der gängigen XML-Parser und Entwicklungswerkzeuge unterstützen bereits die Hauptteile der Spezifikationen.

<b>Tool</b>	<b>Entwickler</b>	<b>Beschreibung</b>
XSV	Henry S. Thompson, Richard Tobin	kommandozeilenorientiert
Xerces-J / Xerces-C++	Apache XML Project	XML-Parser mit API
XML Parser for Java v2	Oracle	XML-Parser mit API
MSXML 4	Microsoft	XML-Parser mit API
XML Spy	Altova	XML-Editor
XML Authority (integriert in TURBO XML)	TIBCO Extensibility	XML Editor

# Eine erstes Beispiel

## *Beispiel SCH-1: Ein erstes Beispiel*

---

Gegeben sei folgendes einfaches XML-Dokument:

### **Instanzdokument**

```
<?xml version="1.0"?>
<Kunde>
  <Vorname>Dieter</Vorname>
  <Zweitname>Thomas</Zweitname>
  <Nachname>Heck</Nachname>
</Kunde>
```

Ein Dokument, welches konform zu einem Schema ist, nennt sich **Instanzdokument** (*instance document*). Schauen wir uns das passende Schema zu obigem XML-Dokument an:

### **XML-Schema**

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="Kunde">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="Vorname" type="xs:string"/>
        <xs:element name="Zweitname" type="xs:string"/>
        <xs:element name="Nachname" type="xs:string"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Die DTD für das obige Instanzdokument sähe folgendermaßen aus:

### **DTD**

```
<!ELEMENT Kunde (Vorname, Zweitname, Nachname)>
<!ELEMENT Vorname (#PCDATA)>
<!ELEMENT Zweitname (#PCDATA)>
<!ELEMENT Nachname (#PCDATA)>
```

### **Erläuterung**

Da ein XML-Schema selbst ein XML-Dokument ist, beginnt es mit einer XML-Deklaration. Jedes XML-Schema hat das Wurzelement `<xs:schema>`. Alle Element- und Attribut-Deklarationen sowie die Datentypdefinitionen eines XML-Schema stehen innerhalb dieses Wurzelements. Hier müssen Sie auch den Standard-Namensraum für XML-Schema deklarieren. Das Präfix können Sie frei wählen, i.a. bietet sich jedoch `xs` an. Der URI ist nicht veränderbar und muss immer wie oben angegeben werden.

In der zweiten Zeile des obigen Beispiels beginnt die Deklaration des ersten Elements mittels des `<xs:element>`-Elements. Über das Attribut `name` wird der Name des deklarierten Elements festgelegt.

Das `<xs:complexType>`-Element sei erst mal zurückgestellt. Es ist - soviel bereits jetzt - immer notwendig, um Inhaltsmodelle für Elemente zu definieren. Das `<xs:sequence>`-Element ist - ähnlich wie das `<xs:element>`-Element - selbstbeschreibend. Es definiert eine Elementfolge und wird **Kompositor** (**compositor**) genannt. XML-Schema kennt noch andere solcher Kompositoren zur Beschreibung von Inhaltsmodellen.

Die Element-Deklarationen innerhalb des `<xs:sequence>`-Elements besitzen alle ein `type`-Attribut, dessen Wert `xs:string` ist. Über dieses Attribut können Elementen und Attributen in XML-Schema Datentypen zugewiesen werden.

---

### **Beachten Sie:**

**XML-Schema-Dateien haben immer die Endung `.xsd`**

# Datentypen: Einführung

## Einfache und komplexe Typen

Einfache Typen unterscheiden sich von komplexen Typen dadurch, dass sie sich auf Daten beziehen, die im Sinne von XML-Schema nicht weiter teilbar sind (*atomare Typen*).

Einfacher formuliert:

Hat ein Element Element-Inhalt oder Attribute, so ist sein Datentyp **komplex**, handelt es sich um Zeichendaten (PCDATA im DTD-Jargon), so ist sein Datentyp **einfach**.

XML stellt eine Reihe von einfachen eingebauten Datentypen zur Verfügung (z.B. `boolean`, `string`, `integer`, `decimal`). Mit diesen Datentypen können Sie festlegen, welche Attributwerte zulässig sind und welche Werte als Textinhalt von Elementen erlaubt sind.

Neben diesen eingebauten Datentypen bietet XML-Schema Ihnen die Möglichkeit, Ihre eigenen Datentypen abzuleiten.

Komplexe Typen benötigt man immer dann, wenn man Elemente deklarieren möchte, die Attribute besitzen oder andere Elemente beinhalten können. In obigem Beispiel haben wir dem Element `<kunde>` einen komplexen Typ zugewiesen, indem wir ein `<xs:complexType>`-Element in das `<xs:element>`-Element eingebunden haben. Wir haben dann über den `<xs:sequence>`-Kompositor das Inhaltsmodell für das Element `<kunde>` definiert.

Folgendes ist nicht erlaubt, auch wenn es zunächst naheliegend erscheint:

### **Beispiel SCH-2: Nicht erlaubte Element-Deklaration**

```
<xs:element name="Kunde">
  <xs:element name="Vorname" type="xs:string">
  <xs:element name="Zweitname" type="xs:string">
  <xs:element name="Nachname" type="xs:string">
</xs:element>
```

## Anonyme und benannte komplexe Typen

Erfolgt die Definition eines komplexen Typs für ein Element innerhalb des `<xs:element>`-Elements, so spricht man von einem **anonymen Typen**. Wollen Sie jedoch, dass mehrere Elemente die gleichen Kindelemente und Attribute tragen, so sollten Sie **benannte komplexe Typen** definieren und diese dann über das `type`-Attribut in der Element-Deklaration dem entsprechenden Element zuweisen.

Das obige Schema sähe unter Verwendung eines benannten komplexen Typs folgendermaßen aus:

### **Beispiel SCH-3: Benannter komplexer Typ**

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:complexType name="KundenTyp">
    <xs:sequence>
      <xs:element name="Vorname" type="xs:string"/>
      <xs:element name="Zweitname" type="xs:string"/>
      <xs:element name="Nachname" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
  <xs:element name="Kunde" type="KundenTyp">
</xs:schema>
```

### **Beachten Sie:**

Es ist unerheblich, ob der benannte komplexe Typ vor oder nach der Element-Deklaration definiert wird.

# XML-Schema-Komponenten

XML-Schema unterscheidet

- primäre Komponenten (*primary components*)
- sekundäre Komponenten (*secondary components*)

## Primäre Komponenten

Es gibt vier primäre Komponenten:

- Element-Deklarationen
- Definitionen einfacher Typen (*simple type definitions*)
- Definitionen komplexer Typen (*complex type definitions*)
- Attribut-Deklarationen

### **Beachten Sie:**

Elemente und Attribute werden **deklariert**, Typen **definiert**.

## Sekundäre Komponenten

- Modell-Gruppen-Definitionen (*model group definitions*)
- Attribut-Gruppen (*attribute groups*)
- Notation-Deklarationen
- Identity Constraints
- Annotationen (*annotations*)

# Primäre Komponenten und Annotationen

## Element-Deklarationen

Die einfachste mögliche Element-Deklaration in XML-Schema hat die Form:

```
<xs:element name="elementname"/>
```

### *Beispiel SCH-4: Einfachste Element-Deklaration in XML-Schema*

```
<xs:element name="geburtstag"/>
```

Das eigentliche Ziel einer Element-Deklaration ist die Verbindung eines Element-Namens mit einem Datentyp. Der Datentyp für ein Element wird über das Attribut `type` angegeben:

```
<xs:element name="elementname" type="xs:datentyp"/>
```

### *Beispiel SCH-5: Element-Deklaration mit expliziter Datentypangabe*

---

## **XML-Schema**

```
<xs:element name="geburtstag" type="xs:date"/>
```

Im obigen Beispiel wird als Datentyp für den Elementinhalt `date` festgelegt, d.h. der Inhalt muss ein Datum im Format `YYYY-MM-DD` sein.

## **Inстанzdokument**

```
<geburtstag>2002-02-13</geburtstag>
```

---

Wird das `type`-Attribut in der Element-Deklaration weggelassen, so ist das Element vom sogenannten **ur-type**. Es kann dann jede Mischung aus Ele-

menten, Attributen und Zeichendaten (Text) enthalten, die im Schema deklariert sind.

Wie bereits dargestellt können Elementen Datentypen auf zwei Arten zugewiesen werden:

- anonym, d.h. innerhalb des `<xs:element>`-Elements
- über Referenzierung, d.h. der Name eines Datentyps taucht als Wert des `type`-Attributs auf

### **Globale versus lokale Element-Deklarationen**

- **Globale Element-Deklarationen** sind Kinder des Wurzelements `<xs:schema>`.
- **Lokale Element-Deklarationen** sind keine direkten Kinder des Wurzelements `<xs:schema>`.

Global deklarierte Elemente können innerhalb eines komplexen Typs referenziert werden und gestatten auf diese Weise Wiederverwendung.

In folgendem Beispiel wird dasselbe Schema für zwei verschiedene Instanzdokumente verwendet. Das eine beinhaltet Angaben über Angestellte, das andere über Kunden einer Firma.

#### ***Beispiel SCH-6: Wiederverwendung von Element-Deklarationen***

---

#### ***Instanzdokument zur Aufnahme von Kundendaten***

```
<?xml version="1.0"?>
<Kunde kundenID="242552">
  <Name>
    <Vorname>Dieter</Vorname>
    <Zweitname>Thomas</Zweitname>
    <Nachname>Heck</Nachname>
  </Name>
  <Adresse>
    <Strasse>Bahnhofstr. 32</Strasse>
    <PLZ>20989</PLZ>
    <Stadt>Hamburg</Stadt>
    <Bundesland>Hamburg</Bundesland>
    <Land>Deutschland</Land>
  </Adresse>
</Kunde>
```

### ***Inстанздokument zur Aufnahme von Mitarbeiterdaten***

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

<Mitarbeiter mitarbeiterID="133">
  <Name>
    <Vorname>Dieter</Vorname>
    <Zweitname>Thomas</Zweitname>
    <Nachname>Heck</Nachname>
  </Name>
  <Adresse>
    <Strasse>Bahnhofstr. 32</Strasse>
    <PLZ>20989</PLZ>
    <Stadt>Hamburg</Stadt>
    <Bundesland>Hamburg</Bundesland>
    <Land>Deutschland</Land>
  </Adresse>
</Mitarbeiter>
```

### ***XML-Schema für Kundendaten und Mitarbeiterdaten***

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

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

  <xs:element name="Kunde">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="Name"/>
        <xs:element ref="Adresse"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:element name="Mitarbeiter">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="Name"/>
        <xs:element ref="Adresse"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:element name="Name">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="Vorname" type="xs:string"/>
        <xs:element name="Zweitname" type="xs:string"/>
        <xs:element name="Nachname" type="xs:string"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
```

```

    </xs:complexType>
  </xs:element>

  <xs:element name="Adresse">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="Strasse" type="xs:string"/>
        <xs:element name="PLZ" type="xs:string"/>
        <xs:element name="Stadt" type="xs:string"/>
        <xs:element name="Bundesland" type="xs:string"/>
        <xs:element name="Land" type="xs:string"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

</xs:schema>

```

### **Beachten Sie:**

Jedes global deklarierte Element kann als Wurzelement eines Instanzdokuments verwendet werden.

Vorteil dieser Regelung: Sie sind in der Lage, Strukturen zu definieren, die dazu benutzt werden können, Fragmente von Dokumenten zu validieren, ohne für jedes Fragment ein eigenes Schema definieren zu müssen. Außerdem ist es auf diese Weise möglich, *ein* Schema für mehrere Dokumententypen zu definieren.

Die einzige Möglichkeit, nur *ein* Wurzelement in einem Instanzdokument zuzulassen, ist, nur ein globales Element zu deklarieren und alle anderen Elemente "unter" dem globalen Element in komplexen Typen zu deklarieren.

Folgendes Dokument wäre also ebenfalls konform zu obigem Schema:

### **Beispiel SCH-7: Validierung von Dokumentenfragmenten**

```

<?xml version="1.0"?>

<Adresse>
  <Strasse>Bahnhofstr. 32</Strasse>
  <PLZ>20989</PLZ>
  <Stadt>Hamburg</Stadt>
  <Bundesland>Hamburg</Bundesland>
  <Land>Deutschland</Land>
</Adresse>

```

### **Beachten Sie:**

Eine Element-Deklaration, die ein `ref`-Attribut enthält, kann weder ein `name`-Attribut tragen, noch einen komplexen Typ enthalten.

### **Häufigkeitsindikatoren für Elemente**

Mittels der optionalen Attribute `minOccurs` und `maxOccurs` lässt sich festlegen, wie oft ein Element auftauchen darf. Die Werte der beiden Attribute können jede beliebige positive Ganzzahl annehmen.

In folgendem Schema ist der Zweitname optional:

#### ***Beispiel SCH-8: minOccurs und maxOccurs***

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="Kunde">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="Vorname" type="xs:string"/>
        <xs:element name="Zweitname" type="xs:string"
          minOccurs="0" maxOccurs="1"/>
        <xs:element name="Nachname" type="xs:string"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

In obigem Beispiel haben wir ein optionales Element definiert. Es muss mindestens 0 mal vorkommen (es kann also auch weggelassen werden) und kann maximal ein mal vorkommen. Lässt man beide Attribute weg, so gilt der Standardwert 1 für beide und das Element muss genau einmal vorkommen.

Die DTD-Häufigkeitsindikatoren lassen sich folgendermaßen in XML-Schema darstellen:

<b>Häufigkeits-indikator</b>	<b>minOccurs-Wert</b>	<b>maxOccurs-Wert</b>	<b>Bedeutung</b>
[keiner]	1	1	genau einmal
?	0	1	kein oder einmal
*	0	unbounded	keinmal o. öfters
+	1	unbounded	einmal o. öfters

### **Beachten Sie:**

- Lassen Sie `maxOccurs` weg, setzen aber `minOccurs`, so nimmt der XML-Schema-Prozessor den Wert von `minOccurs` auch für `maxOccurs` an.
- `minOccurs` und `maxOccurs` können nicht in globalen, sondern nur in lokalen Element-Deklarationen verwendet werden.

Häufigkeitsindikatoren können aber bei lokalen Deklarationen gesetzt werden, die globale Deklarationen referenzieren:

#### **Beispiel SCH-9: Referenzierung globaler Element-Dekl. mit Häufigkeitsindikator**

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

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

  <xs:element name="Kunde">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="Vorname"
                    minOccurs="0"
                    maxOccurs="1"/>
        <xs:element ref="Zweitname"
                    minOccurs="0"
                    maxOccurs="unbounded"/>
        <xs:element ref="Nachname"
                    minOccurs="1"
                    maxOccurs="1"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:element name="Vorname" type="xs:string"/>
  <xs:element name="Zweitname" type="xs:string"/>
  <xs:element name="Nachname" type="xs:string"/>

</xs:schema>
```

### **Standardwerte und feste Werte für Elementinhalt (*value constraints*)**

Sie können für Elemente, die nur Text enthalten, (unveränderbare) Standardwerte vorgeben. Dies geschieht über die Attribute `fixed` bzw. `default` einer Element-Deklaration:

#### **Beispiel SCH-10: Einstellung eines Standardwertes für ein Element**

```
<xs:element name="Status" type="xs:string"
            default="in Arbeit"/>
```

Ist das Element im Instanzdokument leer, so setzt der validierende Parser den Wert des `default`-Attributs standardmäßig als Elementinhalt.

**Beispiel SCH-11: Festschreiben eines Elementinhalts**

```
<xs:element name="Einstufung" type="xs:string"
            fixed="vertraulich"/>
```

Benutzen Sie das `fixed`-Attribut, dann muss das Element entweder leer sein (in diesem Fall setzt der XML-Parser den Wert des `fixed`-Attributs als Elementinhalt ein) oder der Elementinhalt muss gleich dem Wert des `fixed`-Attributs sein.

## Attribut-Deklarationen

Eine Attribut-Deklaration hat formale Ähnlichkeit mit einer Element-Deklaration und sieht in einfachster Form wie folgt aus:

```
<xs:attribute name="attributname"/>
```

**Beispiel SCH-12: Attribut-Deklaration in XML-Schema**

```
<xs:attribute name="kundenID"/>
```

Ein Attribut besitzt immer nur einfache Typen, da es weder Elemente noch andere Attribute enthält.

**Beachten Sie:**

Ist kein `type`-Attribut angegeben, dann wird als Standardwert der `ur-type` bzw. die Version für einfache Typen `anySimpleType` angenommen. Das bedeutet, dass jedes in XML gültige Zeichen als Wert auftauchen darf.

**Beispiel SCH-13: Typisierung für Attribute in XML-Schema**

```
<xs:attribute name="kundenID" type="xs:integer"/>
```

Die Deklaration eines Attributs erfolgt immer innerhalb eines komplexen Typs, da Elemente einfachen Typs keine Attribute besitzen dürfen. Hat ein Element also ein Attribut, so ist es zwangsläufig von komplexem Typ.

Hat ein Element sowohl Element-Inhalt als auch Attribute, so werden Attribute nach der Definition des Element-Inhalts in der Definition des komplexen Typs für das Element angehängt:

**Beispiel SCH-14: Attribut-Deklaration und Element-Inhalt**

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

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

  <xs:element name="Kunde">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="Vorname" type="xs:string"/>
        <xs:element name="Zweitname" type="xs:string"/>
        <xs:element name="Nachname" type="xs:string"/>
      </xs:sequence>
      <xs:attribute name="kundenID" type="xs:integer"/>
    </xs:complexType>
  </xs:element>

</xs:schema>
```

Hat ein Element nur Zeichendaten als Inhalt, so ist die Attributzuweisung etwas komplexer und wird ein wenig später behandelt.

**Globale und lokale Attribut-Deklarationen**

Wie bei den Element-Deklarationen unterscheidet man auch bei Attribut-Deklarationen zwischen globalen und lokalen Deklarationen.

Eine globale Attributdeklaration können Sie auf die gleiche Weise wie eine Element-Deklaration referenzieren:

**Beispiel SCH-15: Globale Attribut-Deklaration**

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

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

  <xs:attribute name="kundenID" type="xs:integer"/>

  <xs:element name="Kunde">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="Vorname" type="xs:string"/>
        <xs:element name="Zweitname" type="xs:string"/>
        <xs:element name="Nachname" type="xs:string"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

</xs:schema>
```

```

    <xs:attribute ref="kundenID"/>
  </xs:complexType>
</xs:element>

</xs:schema>

```

## Attribut-Vorgaben

Über das Attribut `use` in einer Attribut-Deklaration können sie festlegen, ob ein **Attribut obligatorisch** oder **optional** ist. Sie weisen diesem Attribut einfach den Wert `required` oder `optional` zu. Dies entspricht den Schlüsselwörtern `#REQUIRED` und `#IMPLIED` bei Attributlisten-Deklarationen in einer DTD:

### *Beispiel SCH-16: Obligatorische Attribute deklarieren*

## DTD

```
<!ATTLIST Adresse art CDATA #REQUIRED>
```

## XML-Schema

```

<?xml version="1.0"?>

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

  <xs:element name="Adressliste">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="Adresse"
          minOccurs="1"
          maxOccurs="unbounded" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:element name="Adresse">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="Name" type="xs:string" />
        <xs:element name="Strasse" type="xs:string" />
        <xs:element name="PLZ" type="xs:string" />
        <xs:element name="Stadt" type="xs:string" />
      </xs:sequence>
      <xs:attribute name="art"

```

```
        type="xs:string"
        use="required"/>
    </xs:complexType>
</xs:element>

</xs:schema>
```

### ***Inстанздokument***

```
<?xml version="1.0" encoding="iso-8859-1"?>

<Adressliste>
  <Adresse art="privat">
    <Name>Peter Kunert</Name>
    <Strasse>Bahnhofstr. 32</Strasse>
  <Adresse art="geschäftlich">
    <Name>Dieter Mann</Name>
    <Strasse>Weimarer Str. 4</Strasse>
    <PLZ>52786</PLZ>
    <Stadt>Köln</Stadt>
  </Adresse>
  <Adresse art="privat">
    <Name>Horst Liebermann</Name>
    <Strasse>Tegeler Weg 4</Strasse>
    <PLZ>16543</PLZ>
    <Stadt>Berling</Stadt>
  </Adresse>
</Adressliste>
```

---

Das `art`-Attribut für die `<Adresse>`-Elemente könnten Sie auch als optionales Attribut deklarieren:

### Beispiel SCH-17: Optionale Attribute deklarieren

---

#### DTD

```
<!ATTLIST preis bezahlt CDATA #IMPLIED>
```

#### XML-Schema

```
<xs:attribute name="art"  
              type="xs:string"  
              use="optional"/>
```

---

#### Beachten Sie:

- Standardeinstellung ist `optional`.
- Das `use`-Attribut können Sie nicht in globalen Attribut-Deklarationen verwenden.

Es ist ebenfalls möglich, bestimmte Attribute aus einem XML-Dokument auszuschließen:

#### Beispiel SCH-18: Unzulässige Attribute in XML-Schema kennzeichnen

```
<xs:attribute name="art" use="prohibited"/>
```

#### Standardwerte und feste Werte bei Attributen (*value constraints*)

Sie können Attributen **feste Werte** (entspricht `#FIXED` in der DTD) oder **Standardwerte** (entspricht Attribut-Vorgaben mit Wertangabe in der DTD) zuweisen.

## Beispiel SCH-19: Standardwerte vorgeben

---

### DTD

```
<!ATTLIST preis währung CDATA #IMPLIED "€">
```

### XML-Schema

```
<?xml version="1.0" encoding="iso-8859-1"?>

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

  <xs:attribute name="kundenID" type="xs:integer"/>

  <xs:element name="Konto">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="Kontoart" type="xs:string"/>
        <xs:element name="Kontostand" type="xs:string"/>
      </xs:sequence>
      <xs:attribute name="währung" type="xs:string"
        use="optional" default="€"/>
    </xs:complexType>
  </xs:element>

  <xs:element name="Kunde">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="Vorname" type="xs:string"/>
        <xs:element name="Zweitname" type="xs:string"/>
        <xs:element name="Nachname" type="xs:string"/>
        <xs:element ref="Konto"/>
      </xs:sequence>
      <xs:attribute ref="kundenID" use="required"/>
    </xs:complexType>
  </xs:element>

</xs:schema>
```

### Inстанздokument

```
<?xml version="1.0" encoding="iso-8859-1"?>

<Kunde kundenID="3245">
  <Vorname>Dieter</Vorname>
  <Zweitname>Thomas</Zweitname>
  <Nachname>Heck</Nachname>
  <Konto>
```

```
<Kontoart>Girokonto</Kontoart>
<Kontostand>6547,98</Kontostand>
</Konto>
</Kunde>
```

---

Feste Werte werden folgendermaßen festgelegt:

**Beispiel SCH-20: Feste Werte vorgeben**

---

**DTD**

```
<!ATTLIST preis währung CDATA #FIXED "€">
```

**XML-Schema**

```
<xs:attribute name="währung" type="xs:string"
use="optional" fixed="€"/>
```

---

**Beachten Sie:**

- Setzen Sie einen Standardwert für ein Attribut, so muss der Wert des use-Attributs optional sein.
- Sie können fixed-Attribut und default-Attribut nicht in derselben Attribut-Deklaration verwenden.

## Annotationen

XML-Schema bietet zwei Arten von Annotations-Elementen an, die beide innerhalb eines Elements namens `<xs:annotation>` auftauchen können:

- `<xs:documentation>` - Mit diesem Element können Sie Kommentare zu Ihren Deklarationen und Definitionen hinzufügen

- `<xs:appinfo>` - Mit diesem Element können Sie Informationen für weiterverarbeitende Anwendungen zur Verfügung stellen.

Selbstverständlich können Sie auch ganz normale XML-Kommentare verwenden (`<!-- Kommentar -->`).

Die Verwendung der Annotations-Elemente von XML-Schema hat allerdings folgende Vorteile:

- Sie können strukturierte Kommentare innerhalb eines `<xs:documentation>`-Elements hinzufügen (z.B. XHTML-Markup).
- Sie können die Schema-Dokumentation automatisch über ein XSLT-Stylesheet erstellen.
- XML-Parser können XML-Kommentare ignorieren. Die `<xs:documentation>`-Elemente stehen aber jeder Anwendung zur Verfügung. Falls die weiterverarbeitende Anwendung z.B. ein XML-Editor ist, kann dieser über die Auswertung des Inhalts der `<xs:documentation>`-Elemente den Autoren Informationen über die Verwendung des Markups zur Verfügung stellen.

Das `<xs:annotation>`-Element kann zu Beginn fast jeden XML-Schema-Konstrukts auftauchen. Die Annotation gehört jeweils zu dem Konstrukt, an dessen Beginn es auftaucht.

Copyright- und Autoreninformationen können so z.B. direkt unter dem `<xs:schema>`-Element gesetzt werden, während Informationen zu einzelnen Elementen direkt unter das jeweilige `<xs:element>`-Element gehören.

**Beispiel SCH-21: Dokumentation eines XML-Schema mit `<xs:documentation>`**

```
<?xml version="1.0" encoding="iso-8859-1"?>

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

  <xs:annotation>
    <xs:documentation>
      Schema für Kundennamen-Infos
      Benutzt im Seminar "XML-Technologien" der GES
      Copyright Franz-Josef Herpers
    </xs:documentation>
  </xs:annotation>

  <xs:element name="Kunde">
    <xs:annotation>
      <xs:documentation>
        Der Zweitname ist optional, sollte aber benutzt
        werden, wenn der Kunde einen Zweitnamen hat, um
```

```
        zwischen Kunden mit gleichem Namen unterscheiden
        zu können.
    </xs:documentation>
</xs:annotation>
<xs:complexType>
    <xs:sequence>
        <xs:element name="Vorname" type="xs:string"/>
        <xs:element name="Zweitname" type="xs:string"
            minOccurs="0" maxOccurs="1"/>
        <xs:element name="Nachname" type="xs:string"/>
    </xs:sequence>
</xs:complexType>
</xs:element>

</xs:schema>
```

Für das `<xs:appinfo>`-Element gelten dieselben Regeln wie für `<xs:documentation>`.

# Validierung von Instanzdokumenten

## **TOOL-TIPP:**

Die zunächst einfachste und billigste Möglichkeit, Instanzdokumente gegen ein XML-Schema zu validieren, ist der Schema-Prozessor **xsv**. Er wird ständig weiterentwickelt von Henry Thompson, einem Co-Autor der XML-Schema-Empfehlung. Es ist ein kommandozeilenorientiertes Tool und für die Windows-Plattform gibt es eine sich selbst installierende *exe*-Datei. Sie können ihre Dokumente aber auch online über ein Web-Formular validieren lassen.

## **URL:**

<http://www.ltg.ed.ac.uk/~ht/xsv-status.html> - Win32-Version  
<http://www.w3.org/2001/03/webdata/xsv> - Online-Version

## Verknüpfung eines Schema mit einem XML-Dokument

Im Wurzelement des XML-Dokuments geben Sie an, wo der XML-Schema-Prozessor das XML-Schema findet. Dies geschieht über das Attribut `noNamespaceSchemaLocation` bzw. `schemaLocation`, je nachdem, ob Sie ihr Schema an einen Namensraum binden oder nicht.

### **Beispiel SCH-22: Verknüpfung eines XML-Schemas mit einem Instanzdokument**

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

<rechnung
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="rechnungsSchema.xsd">

  ...

</rechnung>
```

Bei der "Ortsangabe" handelt es sich - wie immer bei XML - um einen URI. Das XML-Schema kann dabei sowohl auf einem entfernten Web-Server liegen als auch im Dateisystem des Rechners, auf dem sich auch die XML-Datei befindet.

Die Dateierdung für XML-Schema-Dateien lautet `.xsd`.

***Beachten Sie:***

Der Prozessor ist nicht verpflichtet, das angegebene Schema zu benutzen.

# Datentypen im Detail

Im Unterschied zu XML unterstützt XML-Schema alle gängigen Datentypen und einige mehr.

## Einfache Datentypen versus komplexe Datentypen

Elemente von **einfachem Datentyp** in XML-Schema haben keine Kinder und besitzen auch keine Attribute, können aber eingeschränkt sein auf z.B. Zahlen oder Datumsangaben o.ä.. Attribut-Werte sind immer einfache Typen.

Elemente von **komplexem Datentyp** in XML-Schema können andere Elemente beinhalten und Attribute tragen. Sie können in ähnlicher Weise eingeschränkt werden wie Elemente einfachen Typs.

## Eigenschaften von XML-Schema-Datentypen

Alle XML-Schema-Datentypen bestehen aus drei Teilen:

- Wertebereich (*value space*)
- lexikalischer Bereich (*lexical space*)
- Facetten (*facets*)

Der **Wertebereich** eines Datentyps ist die Menge aller möglichen Werte dieses Datentyps. Z.B. erlaubt ein `integer` alle positiven und negativen Ganzzahlen, aber keine Dezimalzahlen mit Nachkommastellen.

Der **lexikalische Bereich** eines Datentyps ist eine Zeichenkette, die den Wert des Datentyps repräsentiert. Zeichenketten haben per Definition nur eine lexikalische Repräsentation. Zahlen allerdings haben zahlreiche äquivalente lexikalische Repräsentationen, z.B. 100, 100.0 und 1.00e2 für den Zahlenwert 100.

Eine **Facette** ist eine definierende Eigenschaft eines Datentyps. Sie unterscheidet den Datentyp von anderen. Facetten beinhalten Eigenschaften wie die Länge einer Zeichenkette, die Anzahl von Punkte in einer Liste, den Bereich von Minimum- bis zu Maximum-Werten einer Zahl usw.

Man unterscheidet:

- fundamentale Facetten (*fundamental facets*)
- einschränkende Facetten (*constraining facets*)

Die **fundamentalen Facetten** sind jedem Datentyp inhärent und definieren die Charakteristika der Wertebereiche von Datentypen:

- Gleichheit (*equality*)
- Ordnung (*order*)
- Grenzen (*bounds*)
- Kardinalität (*cardinality*)
- numerisch/nicht-nummerisch

Die **einschränkenden Facetten** limitieren die erlaubten Werte des Wertebereichs eines Datentyps. Einschränkende Facetten können benutzt werden, um neue Datentypen von bestehenden durch Beschränkung des Wertebereichs abzuleiten. So können Sie zum Beispiel vom Datentyp *integer* einen neuen Datentyp ableiten, der nur einen Wertebereich von -50 bis +50 zulässt.

## Primitive versus abgeleitete Datentypen

XML-Schema unterscheidet:

- primitive Datentypen (*primitive datatypes*)
- abgeleitete Datentypen (*derived datatypes*)

**Primitive Datentypen** sind die atomaren Einheiten in XML-Schema. Sie sind von keinem anderen Datentyp abgeleitet und nicht teilbar. Sie werden ausschließlich vom W3C in der Spezifikation von XML-Schema festgelegt. Ein solcher Datentyp in XML-Schema ist z.B. eine Zeichenkette (*string*). Einige Programmiersprachen setzen Zeichenketten wiederum aus Zeichen (*characters*) zusammen, in XML-Schema ist eine Zeichenkette allerdings ein primitiver Datentyp und somit nicht weiter in kleinere Einheiten zerlegbar.

Ein **abgeleiteter Datentyp** ist von einem sogenannten **Basistyp** abgeleitet. Auch von abgeleiteten Datentypen können weitere Datentypen abgeleitet werden.

### Beispiel SCH-23: Abgeleiteter Datentyp

```
<xs:simpleType name="PLZ">
  <xs:restriction base="xs:string">
    <xs:length value="5"/>
  </xs:restriction>
</xs:simpleType>
```

Datentypen können in verschiedener Weise abgeleitet werden:

- durch Einschränkung (*by restriction*)
- über eine Liste (*by list*)
- über eine Vereinigung (*by union*)
- durch Erweiterung (nur komplexe Typen) (*by extension*)

## Eingebaute versus benutzerdefinierte Typen

XML-Schema stellt eine Reihe **eingebauter Datentypen (*built-in datatypes*)** zur Verfügung. Diese eingebauten Datentypen sind sowohl primitive als auch abgeleitete Datentypen. Sie wurden vom W3C für so universal gehalten, dass Sie ohnehin von jedem Schema-Entwickler neu erfunden worden wären.

Der Hauptvorteil der Datentypen in XML-Schema ist jedoch, dass Sie ihre eigenen **benutzerdefinierten Datentypen (*user-defined datatypes*)** auf der Basis der eingebauten Datentypen kreieren können. Benutzerdefinierte Datentypen sind also immer abgeleitete Datentypen.

## Arten von Datentypen

- atomare Datentypen (*atomic types*)
- Listen-Typen (*list types*)
- Vereinigungs-Typen (*union types*)

### Atomare Datentypen

Atomare Datentypen sind unteilbar, d.h. sie haben einen Wert, der nicht weiter zerlegt werden kann im Rahmen von XML-Schema. Atomare Typen können sowohl primitiv als auch abgeleitet sein.

### **Beachten Sie:**

Ein atomarer Datentyp ist nicht immer ein primitiver Datentyp!

#### **Beispiel SCH-24 Atomare Datentypen**

```
<atom>Dieser String ist eine schöner String und er kann  
nicht weiter in Buchstaben-Atome zerlegt werden</atom>  
  
<atom>445678</atom>  
<!-- Ein integer ist in XML-Schema ein abgeleiteter Da-  
tentyp -->
```

### **Listen-Typen**

Neben den atomaren Typen, deren Werte nur jeweils eine Einheit umfassen, kennt XML-Schema auch sogenannte **Listen-Typen**. Sie können ihre Listen-Typen von einfachen Typen ableiten. Eingebaute Listen-Typen sind z.B. NMTOKENS, ENTITIES und IDREFS. Die Einzelwerte einer Liste werden durch Leerzeichen getrennt. Im Unterschied zum atomaren Typ `string`, gelten die einzelnen durch Leerzeichen getrennten Zeichendaten dann als getrennte Werte.

#### **Beispiel SCH-25: Ableitung eines Listen-Typs**

---

### **XML-Schema**

```
<xs:simpleType name="Bundesländer">  
  <xs:list itemType="xs:string"/>  
</xs:simpleType>
```

Eine XML-Instanz sähe folgendermaßen aus:

```
<Bundesländer>BW B HH RP NRW</Bundesländer>
```

Problematisch wäre allerdings:

```
<Bundesländer>  
  Hansestadt Hamburg Hansestadt Bremen  
</Bundesländer>
```

---

### **Beachten Sie:**

- Listen-Typen sind nur von einfachen Typen ableitbar, nicht von Listen-Typen.
- Listen-Typen sind immer abgeleitete Typen

### **Vereinigungs-Typen**

Mit Vereinigungs-Typen können Sie Elementen eine Auswahl an Werten zuweisen, die aus einer Vereinigung der Werte verschiedener atomarer oder Listen-Typen besteht. Vereinigungs-Typen sind - wie Listen-Typen - immer abgeleitete Datentypen und setzen sich immer aus mindestens zwei alternativen Datentypen zusammen. Die Basistypen eines Vereinigungs-Typs werden auch **Mitglieder-Typen (*member types*)** genannt.

Die Hauptanwendung eines Vereinigungs-Typs besteht darin, mehrere äquivalente Repräsentationen derselben Daten zur Verfügung zu stellen. So können Sie z.B. zwei verschiedene Formen der Darstellung von Schuhgrößen über einen Vereinigungs-Typ zulassen:

#### ***Beispiel SCH-26: Definition von Vereinigungstypen***

---

### **XML-Schema**

```
<?xml version="1.0" encoding="iso-8859-1"?>

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

  <xs:element name="Kunden">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="Kunde"
                    minOccurs="1"
                    maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:element name="Kunde">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="Name" type="xs:string"/>
        <xs:element name="Schuhgrößen"
                    type="Schuhgrößen"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

```

    </xs:complexType>
  </xs:element>

  <xs:simpleType name="Größenname">
    <xs:restriction base="xs:string">
      <xs:enumeration value="S"/>
      <xs:enumeration value="M"/>
      <xs:enumeration value="L"/>
      <xs:enumeration value="XL"/>
    </xs:restriction>
  </xs:simpleType>

  <xs:simpleType name="Größennamen">
    <xs:list itemType="Größenname"/>
  </xs:simpleType>

  <xs:simpleType name="Größennummern">
    <xs:list itemType="xs:decimal"/>
  </xs:simpleType>

  <xs:simpleType name="Schuhgrößen">
    <xs:union memberTypes="Größennamen Größennummern"/>
  </xs:simpleType>

</xs:schema>

```

Ein Element namens `Schuhgrößen` vom Datentyp `union.Schuhgröße` könnte also folgende verschiedene Formen annehmen:

```

<Schuhgrößen>8 8.5 9 9.5 10 10.5 11 12 13</Schuhgrößen>
<Schuhgrößen>S M L XL</Schuhgrößen>

```

Ein vollständiges Instanzdokument sieht folgendermaßen aus:

### ***Instanzdokument***

```

<Kunden>
  <Kunde>
    <Name>Schuhladen Müller</Name>
    <Schuhgrößen>S L M</Schuhgrößen>
  </Kunde>
  <Kunde>
    <Name>Schuhboutique Friedrich</Name>
    <Schuhgrößen>8.5 9 9.5 10</Schuhgrößen>
  </Kunde>
  <Kunde>
    <Name>Schuhkiste Hermann</Name>
    <Schuhgrößen>9 10 11</Schuhgrößen>

```

```

</Kunde>
<Kunde>
  <Name>Schuhladen Gisela Strunz</Name>
  <Schuhgrößen>S M L</Schuhgrößen>
</Kunde>
</Kunden>

```

## Die eingebauten Datentypen

### Namensraum für eingebaute Datentypen

Um die XML-Schema Datentypen auch in anderen Schemasprachen einsetzen zu können, gibt es neben dem Namensraum für die gesamte XML-Schema Spezifikation einen, der nur die Datentypen abdeckt und den Struktur-Teil ausblendet:

<http://www.w3.org/2001/XMLSchema-datatypes>

### Die eingebauten primitiven Datentypen

Datentyp	Beschreibung	Beispiel
<b>String-Datentypen</b>		
string	jede in XML gültige Zeichenkette	Der Tag ist schön
anyURI	eine URI (relativ oder absolut)	http://www.server.de
QName	ein Qualified Name	xs:element
Notation	repräsentiert den NOTATION-Typ aus XML 1.0.	
<b>Binäre Datentypen</b>		
boolean	repräsentiert binäre Logik (wahr oder falsch)	true, false, 1, 0
base64Binary	Base64-codierte binäre Daten	101001010101
hexBinary	hex-codierte binäre Daten	111011011101
<b>Nummerische Datentypen</b>		
decimal	beliebige Dezimalzahl	
float	reelle Zahlen (32 Bit)	-INF, -1E4, 4.5E-2, 37, INF, NaN

Datentyp	Beschreibung	Beispiel
double	reelle Zahlen (64 Bit)	-INF, -1E4, 4.5E-2, 37, INF, NaN
<b>Datums-/Zeit-Datentypen</b>		
duration	<p>Zeitspanne im Format</p> <p>PnYnMnDTnH:nM:nS</p> <p><i>wobei:</i></p> <p>P = Bezeichner, der immer angegeben werden muss</p> <p>nY = Anzahl Jahre</p> <p>nM = Anzahl Monate</p> <p>nD = Anzahl Tage</p> <p>T = Trenner Datum-Zeit</p> <p>nH = Anzahl Stunden</p> <p>nM = Anzahl Minuten</p> <p>nS = Anzahl Sekunden</p>	<p>P1Y0M1DT20:25:30</p> <p><i>Bedeutung:</i></p> <p>1 Jahr und einen Tag, 20 Stunden, 25 Minuten und 30 Sekunden.</p>
dateTime	<p>ein bestimmter Zeitpunkt im Format:</p> <p>CCYY-MM-DDThh:mm:ss</p> <p><i>wobei:</i></p> <p>CC = Jahrhundert</p> <p>YY = Jahr</p> <p>MM = Monat</p> <p>DD = Tag</p> <p>T = Trenner Datum-Zeit</p> <p>hh = Stunden</p> <p>mm = Minuten</p> <p>ss = Sekunden</p> <p><b>Beachten Sie:</b> Angabe von Jahr 0000 ist verboten</p>	<p>2001-04-16T15:23:15</p> <p><i>Bedeutung:</i></p> <p>16. April 2001, 15 Uhr 23 Minuten und 15 Sekunden</p>
date	<p>Datum nach gregorianischem Kalender</p> <p>Format: CCYY-MM-DD</p>	<p>2001-04-16</p> <p><i>Bedeutung:</i></p> <p>16. April 2001</p>
time	<p>bestimmte Uhrzeit die täglich wiederkehrt</p> <p>Format: HH:MM:SS</p>	<p>14:12:30</p> <p><i>Bedeutung:</i></p> <p>14 Uhr 12 Minuten und 30 Sekunden</p>

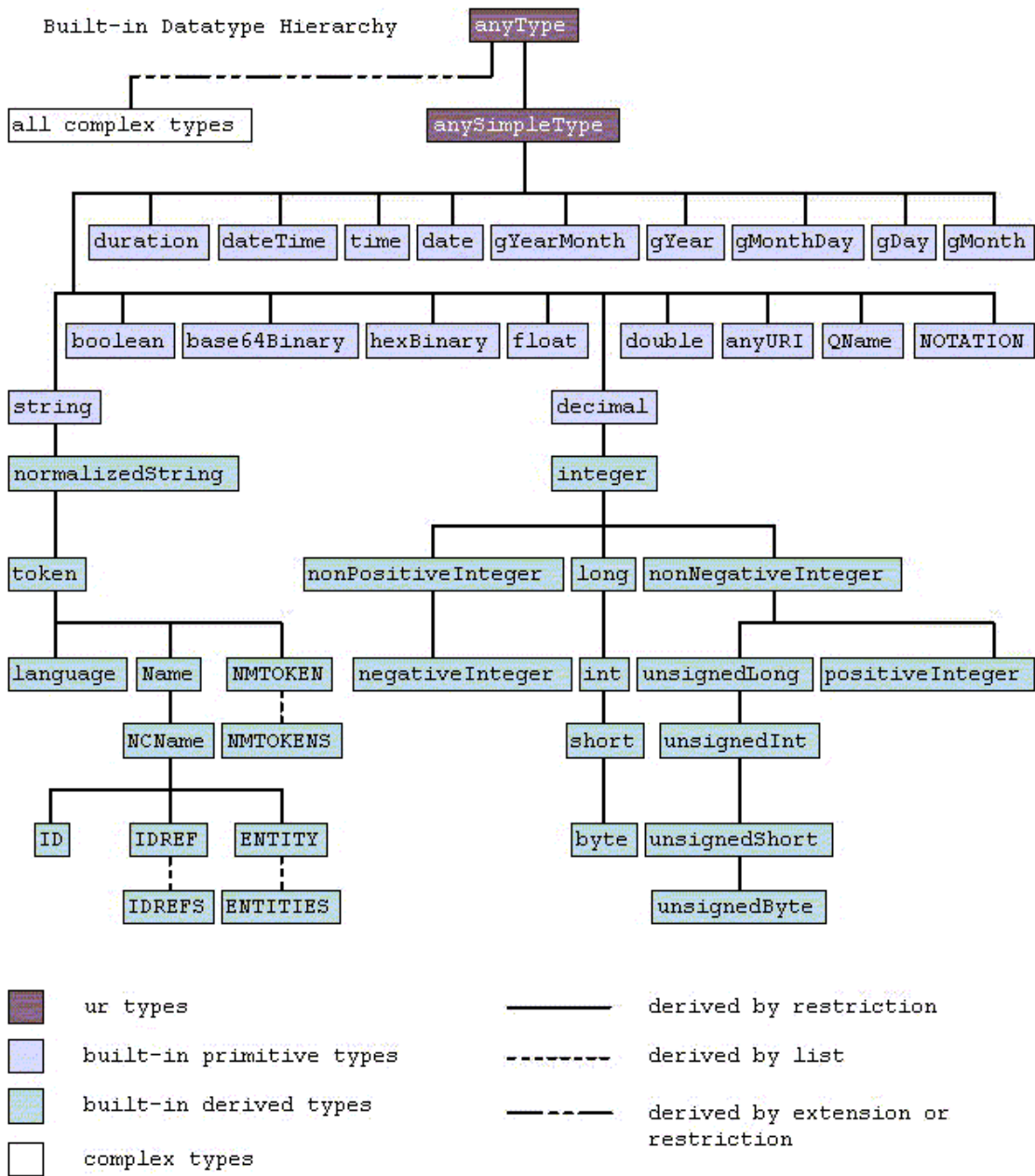
Datentyp	Beschreibung	Beispiel
gYearMonth	ein Monat Format: CCYY-MM	1999-02 <i>Bedeutung:</i> Februar 1999
gYear	ein bestimmtes Jahr Format: CCYY	1986 <i>Bedeutung:</i> Das Jahr 1986
gMonthDay	ein bestimmter Tag eines bestimmten Monats, ungeachtet des Jahres Format: -MM-DD	-16-06 <i>Bedeutung:</i> 16. April (ideal für Geburtstage, Feiertage, und andere jährlich wiederkehrende Ereignisse)
gMonth	ein bestimmter Monat Format: -MM	-12 <i>Bedeutung:</i> Der Monat Dezember
gDay	ein bestimmter Tag eines Monats Format: -DD	-16 <i>Bedeutung:</i> der 16. Tag eines Monats (ideal für monatl. wiederkehrende Ereignisse)

### Die eingebauten abgeleiteten Datentypen

Datentyp	Beschreibung	Beispiel
<b>von string abgeleitet</b>		
normalizedString	in bezug auf Leerraum ( <i>whitespace</i> ) normalisierter String, d.h. ein String der keine Zeilenumbrüche ( <i>carriage return</i> ), Zeilenvorschübe ( <i>line feed</i> ) und Tabs (#x9) mehr enthält	Ein normalisierter String
token	String-Tokens, die in bezug auf Leerraum normalisiert sind und keinen führenden und nachfolgenden Leerstellen mehr enthalten, sowie nicht mehr als zwei Leerstellen hintereinander	Eins Zwei Drei Vier
language	Sprachbezeichner nach RFC 1766	en-GB, en-US, de
Name	XML-Namen	Name, autor, titel
NCName	"non-colonized" Namen aus XML (ein Qualified Name ohne Präfix und Doppelpunkt)	adresse

Datentyp	Beschreibung	Beispiel
ID	Attribut-Typ ID aus XML 1.0	
IDREF	Attribut-Typ IDREF aus XML 1.0	
IDREFS	Attribut-Typ IDREFS aus XML 1.0	
NMTOKEN	Attribut-Typ NMTOKEN aus XML 1.0	klein, mittel, groß
NMTOKENS	Attribut-Typ NMTOKENS aus XML 1.0	klein mittel groß
ENTITY	Attribut-Typ ENTITY aus XML 1.0	
ENTITIES	Attribut-Typ ENTITIES aus XML 1.0	
<b>von decimal abgeleitet</b>		
integer	Standarddatentyp integer	-4, 0, 2, 7
negativeInteger	negative integer-Werte	-4, -1
positiveInteger	Ein integer größer 1	1, 24, 345343
nonNegativeInteger	Ein positiver integer-Wert, einschließlich 0	0, 1, 42
nonPositiveInteger	nicht positive integer-Werte (0 eingeschlossen)	-4, -1, 0
byte	Ein integer zwischen -128 und 127	-127, -42, 0, 54, 125
short	Ein integer zwischen -32768 und 32767	-31353, -43, 345, 31347
int	Ein integer zwischen -2147483648 und 21474836487	-24781982, 24781924
long	ein integer zwischen -9223372036854775898 und 9223372036854775898	-26756454, 781676712556
unsignedByte	Ein nonNegativeInteger zwischen 0 und 255	0, 46, 247
unsignedShort	Ein nonNegativeInteger zwischen 0 und 65535	78, 64328
unsignedInt	Ein nonNegativeInteger zwischen 0 und 4294967295	46, 4255774, 2342823723
unsignedLong	Ein nonNegativeInteger zwischen 0 und 18446744973709551615	0, 356, 238753829383

Die genaue Ableitungshierarchie der eingebauten Datentypen verdeutlicht folgende Abbildung:



**Abbildung SCH-1: Hierarchie der eingebauten Datentypen in XML-Schema**

# Inhaltsmodelle mit XML-Schema

## Kompositoren

Inhaltsmodelle lassen sich in XML-Schema über komplexe Typen definieren. Die Unterscheidung zwischen anonymen komplexen Typen und benannten komplexen Typen haben Sie schon kennen gelernt.

Innerhalb von komplexen Typen können Sie drei **Kompositoren** verwenden, die mehrere Element-Deklaration zu einer **unbenannten Modellgruppe (*unnamed model group*)** zusammenfassen. Die Kompositoren werden übrigens in XML-Schema mit dem etwas umständlichen Begriff ***model group schema components*** benannt.

Die drei erwähnten Kompositoren sind:

- `<xs:sequence>`
- `<xs:choice>`
- `<xs:all>`

### `<xs:sequence>`

Den `<xs:sequence>`-Kompositor kennen Sie bereits. Er zeigt an, dass alle Elemente, die in ihm deklariert werden, in der Reihenfolge im Instanzdokument auftauchen müssen, in der Sie deklariert wurden.

### `<xs:choice>`

Der `<xs:choice>`-Kompositor zeigt an, dass jedes der Elemente, die in ihm deklariert wurden, im Instanzdokument erscheinen kann. Allerdings immer nur eines von allen deklarierten.

#### **Beispiel SCH-27: `<xs:choice>`-Kompositor**

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="Kunde">
    <xs:complexType>
      <xs:choice>
        <xs:element name="Vorname" type="xs:string"/>
        <xs:element name="Nachname" type="xs:string"/>
      </xs:choice>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

```
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Im obigen Beispiel wird eine Gruppe definiert, die die Elemente <Vorname> und <Nachname> enthält, von denen nur eines als Kind des <Kunden>-Elements im Instanzdokument auftauchen darf.

### **<xs:all>**

Der <xs:all>-Kompositor erlaubt, dass die in ihm deklarierten Elemente in jeder beliebigen Reihenfolge auftauchen dürfen. Sie dürfen jedoch nur einmal oder keimnal auftauchen.

#### ***Beispiel SCH-28: <xs:all>-Kompositor***

---

### **XML-Schema**

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="Kunden">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="Kunde"
          minOccurs="1"
          maxOccurs="unbounded" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:element name="Kunde">
    <xs:complexType>
      <xs:all>
        <xs:element name="Vorname" type="xs:string"
          minOccurs="0" maxOccurs="1" />
        <xs:element name="Nachname" type="xs:string"
          minOccurs="0" maxOccurs="1" />
      </xs:all>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Das <Kunden>-Element im Instanzdokument kann das Element <Vorname> und <Nachname> in jeglicher Reihenfolge enthalten und beide Elemente sind optional (wegen der gesetzten minOccurs und maxOccurs-Attribute).

### ***Instanzdokument***

```
<?xml version="1.0"?>
<Kunden
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="../xsd/sch028.xsd">
  <Kunde >
    <Vorname>Dieter</Vorname>
  </Kunde>
  <Nachname>Peter</Nachname>
</Kunde>
  <Kunde >
    <Vorname>Goll</Vorname>
    <Nachname>Ursula</Nachname>
  </Kunde>
  <Kunde>
    <Nachname>Monika</Nachname>
    <Vorname>Schaarschmidt</Vorname>
  </Kunde>
</Kunden>
```

---

### ***Beachten Sie***

Sie können den Wert des minOccurs oder maxOccurs-Attributs eines Elements innerhalb einer all-Gruppe nicht größer als 1 setzen.

### **Kombination von Kompositoren**

Der <xs:sequence>- und <xs:choice>-Kompositor lassen sich auch kombinieren. In folgendem Beispiel lassen wir dem Autor eines Instanzdokuments die Wahl bei der Zusammenstellung eines Eises. Das erste Kind des <Eis>-Elements ist entweder ein <Erdbeer>- oder <Schokolade>-Element. Das zweite Kindelement ist entweder ein <Waffel>-Element oder <Becher>-Element.

#### ***Beispiel SCH-29: Kombination von Kompositoren***

```
<xs:element name="Eis">
  <xs:complexType>
    <xs:sequence>
```

```

<xs:choice>
  <xs:element name="Erdbeer" type="xs:string"/>
  <xs:element name="Schokolade" type="xs:string"/>
</xs:choice>
<xs:choice>
  <xs:element name="Waffel" type="xs:string"/>
  <xs:element name="Becher" type="xs:string"/>
</xs:choice>
</xs:sequence>
</xs:complexType>
</xs:element>

```

### **Beachten Sie:**

- Ein komplexer Typ kann nur einen `<xs:all>`-Kompositor enthalten.
- Das `<xs:all>`-Element muss als erstes Kind des `<xs:complexType>`-Elements folgen
- Der `<xs:all>`-Kompositor kann weder mit dem `<xs:choice>`- noch dem `<xs:sequence>`-Kompositor kombiniert werden.

### **Wiederholung von Kompositoren**

`<xs:choice>`- und `<xs:sequence>`-Elemente können Häufigkeitsindikatoren in Form von `minOccurs` und `maxOccurs`-Attributen tragen (`<xs:all>`-Elemente nicht!). Dadurch werden die Kompositoren wiederholbar.

#### ***Beispiel SCH-30: Wiederholbare Kompositoren***

---

### **XML-Schema**

```

<?xml version="1.0"/>

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

  <xs:element name="Kunden">
    <xs:complexType>
      <xs:sequence minOccurs="1" maxOccurs="unbounded">
        <xs:element name="Vorname" type="xs:string"/>
        <xs:element name="Zweitname"
          type="xs:string"
          minOccurs="0"
          maxOccurs="unbounded"/>
        <xs:element name="Nachname" type="xs:string"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

```

```
    </xs:complexType>
  </xs:element>

</xs:schema>
```

Ein zu obigem Schema konformes Instanzdokument könnte folgendermaßen aussehen:

### **Instanzdokument**

```
<?xml version="1.0" encoding="iso-8859-1"?>

<Kunden>
  <Vorname>Dieter</Vorname>
  <Zweitname>Thomas</Zweitname>
  <Nachname>Heck</Nachname>
  <Vorname>Peter</Vorname>
  <Nachname>Lüttgen</Nachname>
  <Vorname>Rainer</Vorname>
  <Zweitname>Maria</Zweitname>
  <Nachname>Rilke</Nachname>
  <Vorname>Frank</Vorname>
  <Nachname>Grüner</Nachname>
</Kunden>
```

---

## Benannte Modell-Gruppen

Die Gruppen, die durch Kompositoren gebildet werden, sind strenggenommen unbenannte **Modell-Gruppen (*unnamed model groups*)**. Sie können solchen Gruppen jedoch einen Namen geben, sie global definieren und dann in verschiedenen komplexen Typen referenzieren. Es handelt sich hier um das gleiche Prinzip der Wiederverwendung, wie Sie es schon von benannten komplexen Typen her kennen. Das Prinzip ist auch aus DTDs bekannt, wo Parameter-Entities diesen Part übernehmen.

Eine Definition einer **Modell-Gruppe (*model group definition*)** ist innerhalb eines `<xs:group>`-Elements eingeschlossen und wird global definiert.

Das Namensbeispiel dieses Kapitels lässt sich mit Modell-Gruppe folgendermaßen darstellen:

## Beispiel SCH-31: Benannte Modell-Gruppe

---

### XML-Schema

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

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

  <xs:element name="Kunde">
    <xs:complexType>
      <xs:group ref="NamensGruppe"/>
    </xs:complexType>
  </xs:element>

  <xs:group name="NamensGruppe">
    <xs:sequence>
      <xs:element name="Vorname" type="xs:string"/>
      <xs:element name="Zweitname" type="xs:string"/>
      <xs:element name="Nachname" type="xs:string"/>
    </xs:sequence>
  </xs:group>

</xs:schema>
```

### Inстанздokument

```
<?xml version="1.0"?>
<Kunde>
  <Vorname>Dieter</Vorname>
  <Zweitname>Thomas</Zweitname>
  <Nachname>Heck</Nachname>
</Kunde>
```

---

Wie Kompositoren sind auch Modell-Gruppen wiederholbar über Hinzufügen von minOccurs- und maxOccurs-Attributen zum <xs:group>-Element.

## Beispiel SCH-32: Wiederholbare benannte Modell-Gruppen

---

### XML-Schema

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

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

  <xs:element name="Kunden">
    <xs:complexType>
      <xs:group ref="NamensGruppe"
        minOccurs="1"
        maxOccurs="unbounded"/>
    </xs:complexType>
  </xs:element>

  <xs:group name="NamensGruppe">
    <xs:sequence>
      <xs:element name="Vorname" type="xs:string"/>
      <xs:element name="Zweitname"
        type="xs:string"
        minOccurs="0"
        maxOccurs="1"/>
      <xs:element name="Nachname" type="xs:string"/>
    </xs:sequence>
  </xs:group>

</xs:schema>
```

### Inстанздokument

```
<?xml version="1.0"?>
<Kunden>
  <Vorname>Dieter</Vorname>
  <Zweitname>Thomas</Zweitname>
  <Nachname>Heck</Nachname>
  <Vorname>Peter</Vorname>
  <Nachname>Konzelmann</Nachname>
  <Vorname>Rainer</Vorname>
  <Zweitname>Maria</Zweitname>
  <Nachname>Rilke</Nachname>
</Kunden>
```

**Beachten Sie:**

Sie können die Häufigkeitsindikatoren nicht direkt bei der globalen Definition der Modell-Gruppe angeben.

## Attribut-Gruppen

XML-Schema bietet auch für Attribute die Möglichkeit, benannte Gruppen zu definieren. Diese definierten **Attribut-Gruppen (*attribute groups*)** können dann Elementen zugeordnet werden.

Attribut-Gruppen werden mittels des Elements `<xs:attributeGroup>` global definiert.

Folgendes Beispiel verdeutlicht die Verwendung einer Attribut-Gruppe in einem XML-Schema.

**Beispiel SCH-33: Verwendung Attribut-Gruppen**

---

**Inстанздokument**

```
<?xml version="1.0" encoding="iso-8859-1"?>
<Kontakte>
  <Kunden>
    <Kunde kundenID="c143"
          hinzugefügt="2001-05-27"
          letzteÄnderung="2001-05-29"
          autor="e0012">
      <Vorname>Dieter</Vorname>
      <Zweitname>Thomas</Zweitname>
      <Nachname>Heck</Nachname>
    </Kunde>
    <!-- Weitere Kunden -->
  </Kunden>
  <Angestellte>
    <Mitarbeiter mitarbeiterID="e0012"
                 hinzugefügt="2000-06-12"
                 letzteÄnderung="2001-03-23"
                 autor="e0002">
      <Vorname>Rainer</Vorname>
      <Zweitname>Maria</Zweitname>
    </Mitarbeiter>
  </Angestellte>
</Kontakte>
```

```

    <Nachname>Rilke</Nachname>
  </Mitarbeiter>
  <!-- Weitere Mitarbeiter -->
</Angestellte>

<Lieferanten>
  <Lieferant lieferantenID="10001"
    hinzugefügt="2001-04-05"
    letzteÄnderung="2001-04-05"
    autor="e0023">
    <Vorname>Thomas</Vorname>
    <Zweitname>S.</Zweitname>
    <Nachname>Sebestyen</Nachname>
  </Lieferant>
  <!-- Weitere Lieferanten -->
</Lieferanten>

</Kontakte>

```

Im Instanzdokument werden Kundendaten, Mitarbeiterdaten und Lieferantendaten gehalten. Sowohl das <Kunden>-Element als auch die Elemente <Mitarbeiter> und <Lieferant> tragen als Attribute eine id, und drei Attribute zur Kennzeichnung, wann der Eintrag hinzugefügt wurde (hinzugefügt), wann er zuletzt geändert wurde (letzteÄnderung) und von wem (autor).

### **XML-Schema**

```

<?xml version="1.0" encoding="iso-8859-1"?>

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

  <xs:attributeGroup name="ErstellungsdatenGruppe">
    <xs:attribute name="hinzugefügt"
      type="xs:date"
      use="required"/>
    <xs:attribute name="letzteÄnderung"
      type="xs:date"
      use="required"/>
    <xs:attribute name="autor"
      type="xs:string"
      use="required"/>
  </xs:attributeGroup>

  <xs:group name="NamensGruppe">
    <xs:sequence>
      <xs:element name="Vorname" type="xs:string"/>
      <xs:element name="Zweitname"
        type="xs:string"

```

```

                minOccurs="0"
                maxOccurs="1"/>
        <xs:element name="Nachname" type="xs:string"/>
    </xs:sequence>
</xs:group>

<xs:element name="Kontakte">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="Kunden"/>
            <xs:element ref="Angestellte"/>
            <xs:element ref="Lieferanten"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>

<xs:element name="Kunden">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="Kunde" minOccurs="1"
                maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>

<xs:element name="Angestellte">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="Mitarbeiter" minOccurs="1"
                maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>

<xs:element name="Lieferanten">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="Lieferant" minOccurs="1"
                maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>

<xs:element name="Kunde">
    <xs:complexType>
        <xs:group ref="NamensGruppe"/>
        <xs:attribute name="kundenID"
            type="xs:string"
            use="required"/>
        <xs:attributeGroup ref="ErstellungsdatenGruppe"/>

```

```

    </xs:complexType>
  </xs:element>

  <xs:element name="Mitarbeiter">
    <xs:complexType>
      <xs:group ref="NamensGruppe"/>
      <xs:attribute name="mitarbeiterID"
        type="xs:string"
        use="required"/>
      <xs:attributeGroup ref="ErstellungsdatenGruppe"/>
    </xs:complexType>
  </xs:element>

  <xs:element name="Lieferant">
    <xs:complexType>
      <xs:group ref="NamensGruppe"/>
      <xs:attribute name="lieferantenID"
        type="xs:string"
        use="required"/>
      <xs:attributeGroup ref="ErstellungsdatenGruppe"/>
    </xs:complexType>
  </xs:element>

</xs:schema>

```

Die Attribut-Gruppe `ErstellungsdatenGruppe` definiert die gemeinsamen Attribute der Elemente `<Kunde>`, `<Mitarbeiter>` und `<Lieferant>`.

Die Element-Deklarationen für die Elemente `<Kunde>`, `<Mitarbeiter>` und `<Lieferant>` enthalten alle in ihrem `<xs:complexType>`-Element eine Referenz auf eine Attribut-Gruppe, die die gemeinsamen Attribute bündelt.

---

**Beachten Sie:**

Die Referenz auf die Attribut-Gruppe steht an derselben Stelle, an der auch eine Attribut-Deklaration steht, nämlich direkt vor dem End-Tag des `<xs:complexType>`-Elements.

## Definition von Inhaltsmodellen

Die aus der DTD bekannten Inhaltsmodelle lassen sich in XML folgendermaßen darstellen:

### **Elemente mit (nur) Zeichendaten als Inhalt**

Wollen Sie als Elementinhalt lediglich Text, nicht jedoch andere Elemente zulassen, so weisen Sie dem Element einen einfachen Typ zu. Diese Vorgehensweise entspricht der Element-Deklaration mit dem Schlüsselwort #PCDATA in einer DTD.

### **Element-Inhalt**

Wie sie Element-Inhalt in XML-Schema festlegen, wissen Sie bereits. Fassen wir noch einmal die drei verschiedenen Möglichkeiten zusammen, wie sie ein Inhaltsmodell definieren können, das nur Elemente beinhaltet:

- anonymer komplexer Typ
- benannter komplexer Typ
- benannte Modell-Gruppe

### **Gemischter Inhalt**

Während Sie mit DTDs keine Kontrolle über die Reihenfolge der Elemente in gemischten Inhaltsmodellen haben, ist dies bei XML-Schema kein Problem: Um dem XML-Schema-Prozessor anzuzeigen, dass Sie ein gemischtes Inhaltsmodell wünschen, geben Sie einfach das `mixed`-Attribut mit dem Wert `true` zu dem komplexen Typ an, der sich auf das zu definierende Element bezieht.

#### ***Beispiel SCH-34: Gemischter Elementinhalt***

---

### ***Inстанzdokument***

```
<brief>
  <begrüßung>Sehr geehrter Herr Peter
Müller</begrüßung> Ihre Bestellung von
<anzahl>2</anzahl> <produkt>LCD-Monitoren</produkt> ist
am <bestelldatum>2001-11-05</bestelldatum> bei uns
eingegangen.
</brief>
```

## DTD

```
<!ELEMENT brief (#PCDATA | begrüßung | anzahl | produkt
| bestelldatum)*>

<!ELEMENT begrüßung (#PCDATA)>
<!ELEMENT anzahl (#PCDATA)>
<!ELEMENT produkt (#PCDATA)>
<!ELEMENT bestelldatum (#PCDATA)>
```

## XML-Schema

```
<?xml version="1.0" encoding="iso-8859-1"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:element name="brief">
    <xs:complexType mixed="true">
      <xs:sequence>
        <xs:element name="begrüßung" type="xs:string"/>
        <xs:element name="anzahl"
          type="xs:positiveInteger"/>
        <xs:element name="produkt" type="xs:string"/>
        <xs:element name="bestelldatum" type="xs:date"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

</xs:schema>
```

---

## Leere Elemente

Um ein leeres Element in XML-Schema zu definieren, benötigen Sie ebenfalls einen komplexen Typ, der nur Elemente als Inhalt erlaubt; *aber*: Sie deklarieren dann keine Elemente (und referenzieren auch keine global deklarierten) innerhalb des `<xs:complexType>`-Elements, sondern lassen das `<xs:complexType>`-Element einfach leer.

### **Beispiel SCH-35: Leeres Element**

---

#### **XML-Schema**

```
<?xml version="1.0" encoding="iso-8859-1"?>

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

  <xs:element name="bild">
    <xs:complexType/>
  </xs:element>

</xs:schema>
```

#### **DTD**

```
<!ELEMENT bild EMPTY>
```

#### **Inстанздokument**

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

---

Attribute können Sie dann in der Definition des komplexen Typs hinzufügen:

### **Beispiel SCH-36: Leeres Element mit Attributen**

---

#### **XML-Schema**

```
<?xml version="1.0" encoding="iso-8859-1"?>

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

  <xs:element name="produkt">
    <xs:complexType>
      <xs:attribute name="produktID"
                    use="required"
                    type="xs:string"/>
      <xs:attribute name="name"
                    use="required"
                    type="xs:string"/>
    </xs:complexType>
  </xs:element>

</xs:schema>
```

```

        type="xs:string"/>
    </xs:complexType>
</xs:element>

</xs:schema>

```

### ***Inстанздokument***

```
<produkt name="Leberwurst" produktID="7659"/>
```

### **Beliebiger Inhalt**

Über das Schlüsselwort `ANY` können Sie in einer DTD für den Inhalt eines Elements jedes in der DTD deklarierte Element zulassen.

In XML-Schema nutzen Sie dazu das `<xs:any>`-Element, das auch als **Wildcard** bezeichnet wird.

Folgendes Beispiel ist äquivalent zum Gebrauch des `ANY`-Schlüsselworts in der DTD.

#### ***Das Beispiel SCH-37: <xs:any> und ANY***

### ***DTD***

```

<!ELEMENT Kunde (Vorname, Zweitname, Nachname, Ergän-
zung)>
  <!ELEMENT Vorname (#PCDATA)>
  <!ELEMENT Zweitname (#PCDATA)>
  <!ELEMENT Nachname (#PCDATA)>
  <!ELEMENT Ergänzung ANY>

```

### ***XML-Schema***

```

<?xml version="1.0" encoding="iso-8859-1"?>

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

  <xs:element name="Vorname" type="xs:string"/>
  <xs:element name="Zweitname" type="xs:string"/>
  <xs:element name="Nachname" type="xs:string"/>
  <xs:element name="Ergänzung">

```

```

<xs:complexType mixed="true">
  <xs:sequence>
    <xs:any minOccurs="1" maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>
</xs:element>

<xs:element name="Kunde">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="Vorname"/>
      <xs:element ref="Zweitname"/>
      <xs:element ref="Nachname"/>
      <xs:element ref="Ergänzung"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

</xs:schema>

```

### ***Inстанздokument***

```

<?xml version="1.0" encoding="iso-8859-1"?>

<Kunde>
  <Vorname>Dieter</Vorname>
  <Zweitname>Thomas</Zweitname>
  <Nachname>Heck</Nachname>
  <Ergänzung>
    Ein noch schnellerer Ansager als
    <Vorname>Ilja</Vorname>
    <Nachname>Richter</Nachname>
  </Ergänzung>
</Kunde>

```

Das <Ergänzung>-Element darf gemischten Inhalt enthalten, wobei jedes Element, das innerhalb der DTD bzw. dem Schema deklariert wurde, verwendet werden darf.

---

### ***Beachten Sie:***

In einem Element, das mittels <xs:any> deklariert wird, dürfen nur Elemente verwendet werden, die global deklariert wurden.

Mit XML-Schema haben sie für den beliebigen Inhalt von Elementen allerdings wesentlich mehr Gestaltungsmöglichkeiten. So können Sie z.B. nur Element-Inhalt zulassen, indem Sie einfach `mixed="true"` bei der Definition des komplexen Typs weglassen.

Außerdem kann das `<xs:any>`-Element ein Attribut namens `processContents` besitzen, über das Sie dem XML-Schema-Prozessor anzeigen, was er mit dem Inhalt des Elements im Instanzdokument genau machen soll. Das `processContents`-Attribut kann folgende Werte annehmen:

Wert	Bedeutung
skip	zeigt an, dass der Prozessor den Inhalt des Elements nicht validieren soll
strict	zeigt an, dass der Prozessor den Inhalt des Elements validieren soll
lax	zeigt an, dass der Prozessor den Inhalt des Elements validieren soll, wenn er kann

**Beachten Sie:**

`lax` ist der Vorgabewert des `processContents`-Attributs

Im folgenden XML-Schema wird das `processContents`-Attribut mit dem Wert `skip` verwendet. Das `<Nachname>`-Element ist nun lokal deklariert und deshalb darf es bei strikter Validierung nicht im Element `<Ergänzung>` auftauchen.

**Beispiel SCH-38: skip beim processContents-Attribut des <xs:any>-Elements**

```
<?xml version="1.0" encoding="iso-8859-1"?>

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

  <xs:element name="Vorname" type="xs:string"/>
  <xs:element name="Zweitname" type="xs:string"/>

  <xs:element name="Ergänzung">
    <xs:complexType mixed="true">
      <xs:sequence>
        <xs:any minOccurs="1" maxOccurs="unbounded"
          processContents="skip"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
```

```

<xs:element name="Kunde">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="Vorname"/>
      <xs:element ref="Zweitname"/>
      <xs:element name="Nachname" type="xs:string"/>
      <xs:element ref="Ergänzung"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
</xs:schema>

```

Zusätzlich können Sie über das `namespace`-Attribut des `<xs:any>`-Elements einen Namensraum angeben, aus dem das im zugehörigen Element verwendete Markup stammen muss. Auf diese Weise können Sie z.B. sicherstellen, dass lediglich Elemente aus dem XHTML-Namensraum Verwendung finden. Selbstverständlich muss der XHTML-Prozessor dann auch in der Lage sein, das entsprechende Schema zu finden, in dem die Elemente des Namensraums deklariert sind. Setzen Sie den Wert des `processContents`-Attributs auf `lax`, so validiert der XML-Schema-Prozessor den Inhalt des Elements nur, wenn er ein zum Namensraum passendes Schema findet.

#### **Beispiel SCH-39: Namensraumangabe im `<xs:any>`-Element**

### **XML-Schema**

```

<?xml version="1.0" encoding="iso-8859-1"?>

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

  <xs:element name="Vorname" type="xs:string"/>
  <xs:element name="Zweitname" type="xs:string"/>

  <xs:element name="Ergänzung">
    <xs:complexType mixed="true">
      <xs:sequence>
        <xs:any minOccurs="1" maxOccurs="unbounded"
          namespace="http://www.w3.org/1999/xhtml"
          processContents="lax"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:element name="Kunde">
    <xs:complexType>

```

```

<xs:sequence>
  <xs:element ref="Vorname" />
  <xs:element ref="Zweitname" />
  <xs:element name="Nachname" type="xs:string" />
  <xs:element ref="Ergänzung" />
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:schema>

```

### **Instanzdokument**

```

<?xml version="1.0" encoding="iso-8859-1"?>
<Kunde>
  <Vorname>Dieter</Vorname>
  <Zweitname>Thomas</Zweitname>
  <Nachname>Heck</Nachname>
  <Ergänzung>
    <p xmlns="http://www.w3.org/1999/xhtml">
      Ein noch schnellerer Ansager als
      <b>Ilja Richter</b>
    </p>
  </Ergänzung>
</Kunde>

```

### **Beachten Sie:**

Sie können auch mehrere Namensräume durch ein Leerzeichen getrennt angeben.

Neben einer expliziten Namensraumangabe kann der Wert des namespace-Attributs folgende Werte annehmen:

Wert	Bedeutung
##any	Standardeinstellung. Das XML kann aus jedem Namensraum stammen
##local	Bezieht sich auf XML, das nicht mit einem Namensraum qualifiziert ist.

#other	Bezieht sich auf XML, das nicht aus demselben Zielnamensraum stammt, wie das Schema-Dokument
##targetNamespace	Bezieht sich auf XML, das aus demselben Zielnamensraum stammt, wie das Schema-Dokument.

Möchten Sie auch beliebige Attribute verwenden, dann müssen Sie zusätzlich das `<anyAttribute>`-Element verwenden.

Auch mit dem `<xs:anyAttribute>`-Element können Sie ein namespace-Attribut verwenden und damit z.B. alle Attribute aus dem XHTML-Namensraum zulassen.

**Beispiel SCH-40: `<xs:anyAttribute>`**

```

<xs:element name="Ergänzung">
  <xs:complexType mixed="true">
    <xs:sequence>
      <xs:any minOccurs="1" maxOccurs="unbounded"
        namespace="http://www.w3.org/1999/xhtml"
        processContents="lax"/>
    </xs:sequence>
    <xs:anyAttribute
      namespace="http://www.w3.org/1999/xhtml"/>
  </xs:complexType>
</xs:element>

```

**Nil-Werte**

XML-Schema stellt einen Mechanismus zur Verfügung, der es Ihnen ermöglicht, Elemente eines bestimmten Datentyps leer zu lassen, weil es z.B. in diesem Datensatz keinen Wert für dieses Element gibt. Im Normalfall würde ein XML-Schema-Prozessor einen Fehler melden, falls z.B. das Element `<Anzahl>`, welches als `integer` deklariert ist, in einem Instanzdokument plötzlich leer bliebe (ein leerer String ist kein gültiger `integer`-Wert).

Kennzeichnen Sie das Element jedoch über das Attribut `nillable` im XML-Schema, so können Sie im Instanzdokument durch Hinzufügung eines `nil`-Attributs mit dem Wert `true` dem XML-Schema-Prozessor anzeigen, dass es sich nicht um ein fälschlicherweise leeres Element handelt, sondern um so etwas wie den `NULL`-Wert, den Sie vielleicht von Datenbanken kennen.

## Beispiel SCH-41: Deklaration eines Elements als *nillable*

---

### XML-Schema

```
<?xml version="1.0" encoding="iso-8859-1"?>

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

  <xs:element name="Kunden">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="Kunde" minOccurs="1"
          maxOccurs="unbounded" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:element name="Vorname" type="xs:string"/>
  <xs:element name="Zweitname" type="xs:string"/>
  <xs:element name="Nachname" type="xs:string"/>
  <xs:element name="AnzahlKinder" type="xs:integer"
    nillable="true"/>

  <xs:element name="Kunde">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="Vorname"/>
        <xs:element ref="Zweitname"/>
        <xs:element ref="Nachname"/>
        <xs:element ref="AnzahlKinder"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

</xs:schema>
```

### Instanzdokument

```
<?xml version="1.0" encoding="iso-8859-1"?>

<Kunden
  xsi:noNamespaceSchemaLocation="../xsd/sch039.xsd">
  <Kunde>
    <Vorname>Dieter</Vorname>
    <Zweitname>Thomas</Zweitname>
    <Nachname>Heck</Nachname>
    <AnzahlKinder>5</AnzahlKinder>
  </Kunde>
```

```

<Kunde>
  <Vorname>Peter</Vorname>
  <Zweitname>Richard</Zweitname>
  <Nachname>König</Nachname>
  <AnzahlKinder>1</AnzahlKinder>
</Kunde>
<Kunde>
  <Vorname>Rainer</Vorname>
  <Zweitname>Maria</Zweitname>
  <Nachname>Fassbinder</Nachname>
  <AnzahlKinder xsi:nil="true"/>
</Kunde>
<Kunde>
  <Vorname>Sven</Vorname>
  <Zweitname>Martin</Zweitname>
  <Nachname>Heger</Nachname>
  <AnzahlKinder xsi:nil="true"></AnzahlKinder>
</Kunde>
</Kunden>

```

Dieses Feature ist v.a. beim Export aus Datenbanken nützlich.

**Beachten Sie:**

Das `xsi`-Präfix gehört zum XML-Schema-Namensraum für Instanzdokumente.

## Ein Beispiel

Betrachten wir ein etwas komplizierteres Beispiel. Folgendes XML-Dokument stellt eine Sammlung von Kontaktadressen dar:

**Beispiel SCH-42: Komplexeres Beispiel**

**Instanzdokument**

```

<?xml version="1.0" encoding="iso-8859-1"?>

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

```

```

xsi:noNamespaceSchemaLocation="Kontaktliste.xsd">
<Kontakt hinzugefügt="2001-03-15"
      letzteÄnderung="2001-05-31">
  <Name>
    <Vorname>Dieter</Vorname>
    <Zweitname>Thomas</Zweitname>
    <Nachname>Heck</Nachname>
  </Name>
  <Privatadresse>
    <Strasse>Kleine Hamburger Str. 7</Strasse>
    <PLZ>14567</PLZ>
    <Stadt>Berlin</Stadt>
    <Bundesland>Berlin</Bundesland>
    <Land>Deutschland</Land>
  </Privatadresse>
  <Geschäftsadresse>
    <Strasse>Elberfelder Str. 54</Strasse>
    <PLZ>56748</PLZ>
    <Stadt>Köln</Stadt>
    <Bundesland>NRW</ Bundesland >
    <Land>Deutschland</Land>
  </Geschäftsadresse>
  <E-Kontakt>
    <Email>dth@hitparade.org</Email>
    <TelPrivat>0306543672</TelPrivat>
    <TelGeschäftlich>0221872684</TelGeschäftlich>
    <Fax> 0221276387 </Fax>
  </E-Kontakt>
</Kontakt>

<!-- Weitere Kontakte -->

</Kontaktliste>

```

Achten Sie beim zugehörigen XML-Schema vor allem auf folgende Dinge:

- Es ist eine benannte Modell-Gruppe für die Adressdetails definiert, damit diese wiederverwendet werden können.
- Das Schema besitzt nur ein global deklariertes Element, damit das Wurzelement eines Instanzdokument eindeutig bestimmt ist.
- Es sind einige Elemente deklariert, die im Beispiel-Instanzdokument nicht auftauchen (z.B. <Fax>, <Pager> im <E-Kontakt>-Element).

### **XML-Schema**

```
<?xml version="1.0" encoding="iso-8859-1"?>
```

```

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

  <xs:element name="Kontaktliste">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="Kontakt"
          type="KontaktDetailsTyp"
          minOccurs="0"
          maxOccurs="unbounded" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:complexType name="KontaktDetailsTyp">
    <xs:sequence>
      <xs:element name="Name">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="Vorname"
              type="xs:string" />
            <xs:element name="Zweitname"
              type="xs:string"
              minOccurs="0"
              maxOccurs="1" />
            <xs:element name="Nachname"
              type="xs:string" />
          </xs:sequence>
        </xs:complexType>
      </xs:element>

      <xs:element name="Privatadresse"
        minOccurs="0"
        maxOccurs="1">
        <xs:complexType>
          <xs:group ref="AdressenGruppe" />
        </xs:complexType>
      </xs:element>

      <xs:element name="Geschäftsadresse"
        minOccurs="0"
        maxOccurs="1">
        <xs:complexType>
          <xs:group ref="AdressenGruppe" />
        </xs:complexType>
      </xs:element>

      <xs:element name="E-Kontakt">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="Email"

```

```

        type="xs:string"
        minOccurs="0"
        maxOccurs="1"/>
<xs:element name="TelPrivat"
    type="xs:integer"
    nillable="true"
    minOccurs="0"
    maxOccurs="1"/>
<xs:element name="TelGeschäftlich"
    type="xs:integer"
    nillable="true"
    minOccurs="0"
    maxOccurs="1"/>
<xs:element name="Fax"
    type="xs:integer"
    minOccurs="0"
    maxOccurs="1"/>
<xs:element name="Pager"
    type="xs:integer"
    minOccurs="0"
    maxOccurs="1"/>
    </xs:sequence>
</xs:complexType>
</xs:element>

</xs:sequence>
<xs:attribute name="hinzugefügt"
    type="xs:date"
    use="required"/>
<xs:attribute name="letzteÄnderung" type="xs:date"/>
</xs:complexType>

<xs:group name="AdressenGruppe">
    <xs:sequence>
        <xs:element name="Strasse" type="xs:string"/>
        <xs:element name="PLZ" type="xs:string"/>
        <xs:element name="Stadt" type="xs:string"/>
        <xs:element name="Bundesland" type="xs:string"/>
        <xs:element name="Land" type="xs:string"/>
    </xs:sequence>
</xs:group>

</xs:schema>

```

# Ableitung von neuen Datentypen

## Fundamentale Facetten

Fundamentale Facetten sind dem Datentyp inhärent und definieren die Charakteristika der Wertebereiche von Datentypen:

- **Gleichheit (*equality*)** - verschiedene Werte können verglichen werden und es lässt sich bestimmen, ob Sie gleich sind ( $A=B$  oder  $B=A$ ) oder nicht ( $A \neq B$  oder  $A \neq B$ ). Diese Facette trifft auf alle Datentypen in XML-Schema zu.
- **Ordnung (*order*)** - Zahlen und andere Datentypen können geordnete Werte haben, die einen Größer-als ( $A < B$ ) oder Kleiner-als-Vergleich ( $A < B$ ) erlauben. Diese Facette trifft auf alle numerischen Datentypen und Datumsangaben zu.
- **Grenzen (*bounds*)** - geordnete Datentypen können Minimal- und Maximalwerte haben (z.B. `int`, `short`, `positiveInteger`).
- **Kardinalität (*cardinality*)** - stellt die erlaubte Anzahl von Werten im Wertebereich des Datentyps dar (trifft auf alle Datentypen zu).
- **numerisch/nicht-numerisch** - alle Datentypen lassen sich in diese Dichotomie einordnen.

## Einschränkende Facetten

Sie können von den in XML-Schema eingebauten Datentypen ihre eigenen individuellen Datentypen ableiten. Dies geschieht über sogenannte **einschränkende Facetten**.

Die einschränkende Facetten in XML-Schema sind:

<b>Facette</b>	<b>Bedeutung</b>
length	beschränkt die Länge eines einfachen Typs durch Angabe der Anzahl von Einheiten, die er enthalten darf. Diese Einheiten sind Buchstaben, Ziffern, Listenpunkte usw., je nachdem um welchen Typ es sich handelt.
minLength	minimale Längenangabe für den Wert des Typs
maxLength	maximale Längenangabe für den Wert des Typs
pattern	der Wert des Typs muss mit einem angegebenen Muster (regulären Ausdruck) übereinstimmen
enumeration	der Wertebereich des Typs wird auf den/die in einer Aufzählung angegebenen Werte beschränkt
maxInclusive	legt eine inklusive Obergrenze für den Wertebereich des Typs fest
maxExclusive	legt eine exklusive Obergrenze für den Wertebereich des Typs fest
minInclusive	legt eine inklusive Untergrenze für den Wertebereich des Typs fest
minExclusive	legt eine exklusive Untergrenze für den Wertebereich des Typs fest
whiteSpace	beschränkt den Wertebereich des Typs gemäß den Regeln für Whitespace-Normalisierung in XML 1.0
totalDigits	legt die maximale Stellenanzahl eines Dezimalwerts fest
fractionDigits	legt die maximale Anzahl Stellen fest, die hinter dem Dezimalzeichen eines Dezimalwerts erscheinen dürfen

## Einige Beispiele für Facetten

### length, minlength, maxlength

Die Einheit für die Längenbeschränkung bei den Facetten `length`, `minlength` und `maxLength` richten sich nach dem Basistyp:

- Zeichen für Basistyp `string`
- Listeneinträge bei Listen-Typen

#### **Beispiel SCH-43: Längenbeschränkungen von einfachen Typen**

```
<xs:simpleType name="PLZ">
  <xs:restriction base="xs:string">
    <xs:length value="5"/>
  </xs:restriction>
</xs:simpleType>
```

Der folgende selbst definierte, einfache Typ beschränkt den Inhalt von Elementen des Typs `ktoNrTyp` auf ein Muster mit fünf Ziffern, einem optionalen Bindestrich und vier weiteren Ziffern:

#### **Beispiel SCH-44: pattern-Facette**

---

### **XML-Schema**

```
<xs:element name="ktoNr"/>
<xs:simpleType>
  <xs:restriction base="xs:string">
    <xs:pattern value="\d{5}(-\d{4})?"/>
  </xs:restriction>
</xs:simpleType>
</xs:element>
```

Beispiele für Elemente, welche diese Element-Deklaration erfüllen, sähen folgendermaßen aus:

## ***Inстанздokument***

```
<ktoNr>12456-6543</ktoNr>  
<ktoNr>65980</ktoNr>
```

---

## **enumeration**

Die Facette `enumeration`, ist die richtige Wahl, wenn Sie eine genaue Aufzählung der zulässigen Werte angeben möchten:

### ***Beispiel SCH-45: enumeration-Facette***

```
<xs:element name="Kleidergröße">  
  <xs:simpleType>  
    <xs:restriction base="xs:string">  
      <xs:enumeration value="XL"/>  
      <xs:enumeration value="L"/>  
      <xs:enumeration value="M"/>  
      <xs:enumeration value="S"/>  
    </xs:restriction>  
  </xs:simpleType>  
</xs:element>
```

## **minExclusive, maxExclusive, minInclusive, maxInclusive**

Die Facetten `minExclusive`, `maxExclusive`, `minInclusive` und `maxInclusive` legen den niedrigsten bzw. höchsten zulässigen Wert für ein Element fest. Sie könne Mindest- und Maximalbegrenzungen nach Bedarf kombinieren. Im folgenden Beispiel muss das Anfangsdatum dem 22.05.2001 entsprechen oder später liegen.

### ***Beispiel SCH-46: minInclusive-Facette***

```
<xs:element name="Anfangsdatum">  
  <xs:simpleType>  
    <xs:restriction base="xs:date">  
      <xs:minInclusive value="2001-05-22"/>  
    </xs:restriction>  
  </xs:simpleType>  
</xs:element>
```

## totalDigits, fractionDigits

Mittels der Facetten `totalDigits` und `fractionDigits` können Sie vorschreiben, wie viele Stellen rechts und links vom Dezimalzeichen zulässig sind.

### **Beispiel SCH-47: Facetten `totalDigits` und `fractionDigits`**

```
<xs:simpleType name='Körpertemperatur_Typ'>
  <xs:restriction base='xs:decimal'>
    <xs:totalDigits value='4' />
    <xs:fractionDigits value='1' />
    <xs:minInclusive value='36.4' />
    <xs:maxInclusive value='40.5' />
  </xs:restriction>
</xs:simpleType>
```

### **Beachten Sie:**

Nicht alle Facetten sind auf alle Datentypen anwendbar. Die XML-Schema Empfehlung enthält genaue Tabellen, welche Facetten für welchen Datentyp infrage kommen.

## Vererbung bei Datentypen

Die Möglichkeit in XML-Schema eigene Datentypen abzuleiten, entspricht dem objektorientierten Konzept der Vererbung.

**Einfache Typen** können auf drei verschiedene, bereits bekannte, Arten abgeleitet werden:

- durch Einschränkung (*by restriction*)
- über eine Liste (*by list*)
- über eine Vereinigung (*by union*)

**Komplexe Typen** lassen sich auf zwei Arten ableiten:

- durch Erweiterung (*by extension*)
- durch Einschränkung eines anderen komplexen Typs oder des Ur-Typs (*ur-type*)

## Ableitung einfacher Typen

Die Ableitungsarten für einfache Typen haben Sie bereits alle kennen gelernt. Folgendes Beispiel fasst die drei Ableitungsmöglichkeiten noch einmal zusammen:

### *Beispiel SCH-48: Ableitungsarten einfacher Typen auf einen Blick*

---

#### **XML-Schema**

```
<?xml version="1.0" encoding="iso-8859-1"?>

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

  <xs:element name="Kunden">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="Kunde" minOccurs="1"
          maxOccurs="unbounded" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:element name="Kunde">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="Name" type="xs:string" />
        <xs:element name="Schuhgrößen"
          type="Vereinigung.Schuhgrößen" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:simpleType name="Größenname">
    <xs:restriction base="xs:string">
      <xs:enumeration value="S" />
      <xs:enumeration value="M" />
      <xs:enumeration value="L" />
      <xs:enumeration value="XL" />
    </xs:restriction>
  </xs:simpleType>

  <xs:simpleType name="Größennummer">
    <xs:restriction base="xs:decimal">
      <xs:minInclusive value="3.0" />
      <xs:maxInclusive value="15.0" />
      <xs:pattern value="[0-9]{1,2}([.][0-9])?" />
    </xs:restriction>
  </xs:simpleType>
</xs:schema>
```

```

    </xs:restriction>
</xs:simpleType>

<xs:simpleType name="Liste.Größennamen">
  <xs:list itemType="Größensname"/>
</xs:simpleType>

<xs:simpleType name="Liste.Größennummern">
  <xs:list itemType="Größensnummer"/>
</xs:simpleType>

<xs:simpleType name="Vereinigung.Schuhgrößen">
  <xs:union
    memberTypes="Liste.Größennamen Liste.Größennummern"/>
</xs:simpleType>

</xs:schema>

```

### **Inстанздokument**

```

<?xml version="1.0" encoding="iso-8859-1"?>

<Kunden>
  <Kunde>
    <Name>Schuladen Müller</Name>
    <Schuhgrößen>S L M</Schuhgrößen>
  </Kunde>
  <Kunde>
    <Name>Schuhboutique Friedrich</Name>
    <Schuhgrößen>8.5 9 9.5 10</Schuhgrößen>
  </Kunde>
  <Kunde>
    <Name>Schuhkiste Hermann</Name>
    <Schuhgrößen>9 10 11</Schuhgrößen>
  </Kunde>
  <Kunde>
    <Name>Schuladen Gisela Strunz</Name>
    <Schuhgrößen>S M L</Schuhgrößen>
  </Kunde>
</Kunden>

```

## Ableitung komplexer Typen

Komplexe Typen lassen sich ableiten von:

- einfachen Typen
- komplexen Typen

Eine **Ableitung von einfachen Typen** ist z.B. notwendig, um einem Element von einfachem Datentyp ein Attribut (per Erweiterung hinzuzufügen). Sie müssen dann bei der Definition des komplexen Typs allerdings signalisieren, dass dieser komplexe Typ nur Zeichendaten als Elementinhalt zulässt (also von einem einfachen Typ abgeleitet ist).

### *Beispiel SCH-49: Ableitung eines komplexen Typs von einem einfachen Typ*

---

#### **XML-Schema**

```
<?xml version="1.0" encoding="iso-8859-1"?>

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

  <xs:element name="Produkte">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="Produkt"
                    minOccurs="1"
                    maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:element name="Produkt">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="Name" type="xs:string"/>
        <xs:element ref="Preis"/>
        <xs:element ref="InternationalerPreis"
                    minOccurs="0"
                    maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:element name="Preis" type="Preis_Typ"/>

  <xs:simpleType name="Preis_Typ">
```

```

    <xs:restriction base="xs:decimal">
      <xs:fractionDigits value="2"/>
    </xs:restriction>
  </xs:simpleType>

  <xs:element name="InternationalerPreis">
    <xs:complexType>
      <xs:simpleContent>
        <xs:extension base="xs:anyType">
          <xs:attribute name="währung"
            type="xs:string"/>
        </xs:extension>
      </xs:simpleContent>
    </xs:complexType>
  </xs:element>

</xs:schema>

```

Sie benutzen das `<xs:complexType>`-Element, um einen komplexen Typ für das Element `<InternationalerPreis>` zu definieren. Um anzuzeigen, dass das Inhaltsmodell des neuen Typs nur Zeichendaten enthalten wird, verwenden Sie das Element `<xs:simpleContent>`. Schließlich leiten Sie den neuen komplexen Typ ab, indem Sie den einfachen Typ `Preis_Typ` erweitern. Die Erweiterung besteht in der Hinzufügung eines Attributs namens `währung`.

## **Inстанздokument**

```
<?xml version="1.0" encoding="iso-8859-1"?>

<Produkte>
  <Produkt>
    <Name>Auto</Name>
    <Preis>500000</Preis>
    <InternationalerPreis wahrung="US-Dollar">
      250000
    </InternationalerPreis>
    <InternationalerPreis wahrung="Baht">
      250000
    </InternationalerPreis>
  </Produkt>
  <Produkt>
    <Name>Leberwurst</Name>
    <Preis>20.45</Preis>
    <InternationalerPreis wahrung="US-Dollar">
      10.30
    </InternationalerPreis>
  </Produkt>
</Produkte>
```

---

Eine **Ableitung von komplexen Typen** erfordert die Verwendung des Elements `<xs:complexContent>`, um anzuzeigen, dass der Elementinhalt entweder gemischt ist oder nur aus weiteren Elementen besteht (der komplexe Typ also von einem anderen komplexen Typ abgeleitet ist).

**Beispiel SCH-50: Ableitung eines komplexen Typs von einem komplexen Typ**

---

## **XML-Schema**

```
<?xml version="1.0" encoding="iso-8859-1"?>

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

  <xs:element name="Adressen">
    <xs:complexType>
      <xs:sequence minOccurs="1" maxOccurs="unbounded">
        <xs:choice>
          <xs:element name="Adresse"
```

```

        type="Adresse_Typ" />
        <xs:element name="InternationaleAdresse"
            type="InternationaleAdresse_Typ" />
    </xs:choice>
</xs:sequence>
</xs:complexType>
</xs:element>

<xs:complexType name="Adresse_Typ">
    <xs:sequence>
        <xs:element name="Name" type="xs:string" />
        <xs:element name="Strasse" type="xs:string" />
        <xs:element name="PLZ" type="xs:string" />
        <xs:element name="Stadt" type="xs:string" />
    </xs:sequence>
</xs:complexType>

<xs:complexType name="InternationaleAdresse_Typ">
    <xs:complexContent>
        <xs:extension base="Adresse_Typ">
            <xs:sequence>
                <xs:element name="Land" type="xs:string" />
            </xs:sequence>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>

</xs:schema>

```

## ***Instanzdokument***

```
<?xml version="1.0" encoding="iso-8859-1"?>

<Adressen>
  <Adresse>
    <Name>Dieter Thomas Heck</Name>
    <Strasse>Hamburger Str. 45</Strasse>
    <PLZ>20356</PLZ>
    <Stadt>Hamburg</Stadt>
  </Adresse>
  <InternationaleAdresse>
    <Name>Michael Jackson</Name>
    <Strasse>Dream Boulevard 24</Strasse>
    <PLZ>12345</PLZ>
    <Stadt>Los Angeles</Stadt>
    <Land>USA</Land>
  </InternationaleAdresse>
</Adressen>
```

---

Die **Ableitung durch Erweiterung und Einschränkung** wurde in obigen Beispielen ja bereits demonstriert. Hier noch mal ein Beispiel, das beide Ableitungsarten verwendet.

### ***Beispiel SCH-51: Ableitung durch Einschränkung und Erweiterung***

## ***XML-Schema***

```
<?xml version="1.0" encoding="iso-8859-1"?>

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

  <xs:element name="Adressen">
    <xs:complexType>
      <xs:sequence minOccurs="1" maxOccurs="unbounded">
        <xs:choice>
          <xs:element name="Adresse"
            type="Adresse_Typ"/>
          <xs:element name="InternationaleAdresse"
            type="InternationaleAdresse_Typ"/>
          <xs:element name="AdresseOhneBundesland"
            type="AdresseOhneBundesland_Typ"/>
        </xs:choice>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

```

        </xs:choice>
    </xs:sequence>
</xs:complexType>
</xs:element>

<xs:complexType name="Adresse_Typ">
    <xs:sequence>
        <xs:element name="Name" type="xs:string"/>
        <xs:element name="Strasse" type="xs:string"/>
        <xs:element name="PLZ" type="xs:string"/>
        <xs:element name="Stadt" type="xs:string"/>
        <xs:element name="Bundesland" type="xs:string"/>
    </xs:sequence>
</xs:complexType>

<xs:complexType name="InternationaleAdresse_Typ">
    <xs:complexContent>
        <xs:extension base="AdresseOhneBundesland_Typ">
            <xs:sequence>
                <xs:element name="Land" type="xs:string"/>
            </xs:sequence>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>

<xs:complexType name="AdresseOhneBundesland_Typ">
    <xs:complexContent>
        <xs:restriction base="Adresse_Typ">
            <xs:sequence>
                <xs:element name="Name" type="xs:string"/>
                <xs:element name="Strasse" type="xs:string"/>
                <xs:element name="PLZ" type="xs:string"/>
                <xs:element name="Stadt" type="xs:string"/>
            </xs:sequence>
        </xs:restriction>
    </xs:complexContent>
</xs:complexType>

</xs:schema>

```

### ***Instanzdokument***

```

<?xml version="1.0" encoding="iso-8859-1"?>

<Adressen>
  <Adresse>
    <Name>Dieter Thomas Heck</Name>
    <Strasse>Hamburger Str. 45</Strasse>
    <PLZ>20356</PLZ>
  </Adresse>
</Adressen>

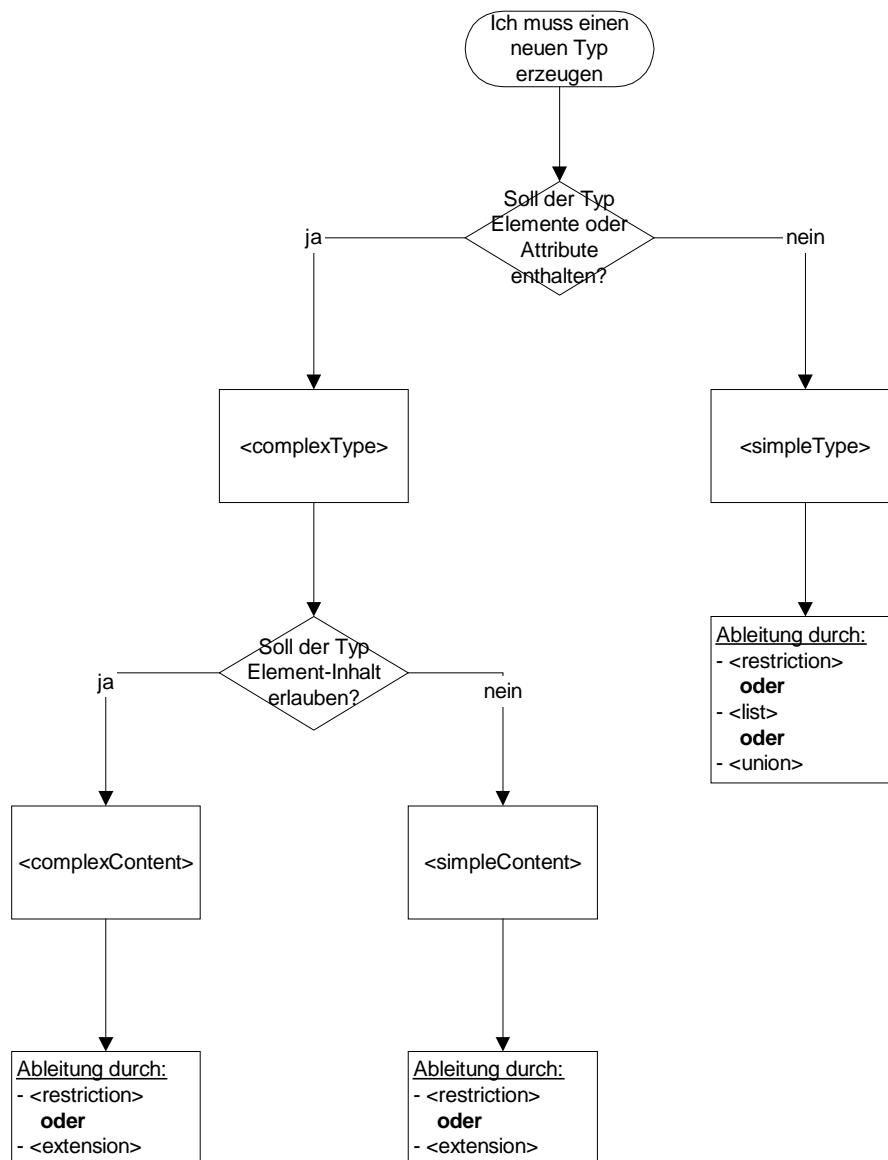
```

```
<Stadt>Hamburg</Stadt>
  <Bundesland>Hamburg</Bundesland>
</Adresse>
<InternationaleAdresse>
  <Name>Michael Jackson</Name>
  <Strasse>Dream Boulevard 24</Strasse>
  <PLZ>12345</PLZ>
  <Stadt>Los Angeles</Stadt>
  <Land>USA</Land>
</InternationaleAdresse>
<AdresseOhneBundesland>
  <Name>Peter Lüttgen</Name>
  <Strasse>Hohenzollernstr. 1</Strasse>
  <PLZ>51674</PLZ>
  <Stadt>Düren</Stadt>
</AdresseOhneBundesland>
</Adressen>
```

---

## Entscheidungsbaum für XML-Schema-Typen

Folgender Entscheidungsbaum ist eine gute Orientierung für die Typenhierarchie von XML-Schema.



**Abbildung SCH-2: Entscheidungsbaum für Typhierarchie in XML-Schema**

Anhand dieser Typhierarchie läßt sich nun auch die etwas eigentümliche Deklaration von leeren Elementen gut verdeutlichen.

### **Beispiel SCH-52: Deklaration leerer Elemente**

---

Die ohne den Kontext der Typhierarchie etwas schwer nachvollziehbare Art, ein leeres Element zu deklarieren, ist die folgende:

```
<?xml version="1.0" encoding="iso-8859-1"?>

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

  <xs:element name="bild">
    <xs:complexType/>
  </xs:element>

</xs:schema>
```

Diese Variante ist allerdings nur eine Abkürzung für die folgende Schreibweise:

```
<?xml version="1.0" encoding="iso-8859-1"?>

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

  <xs:element name="bild">
    <xs:complexType>
      <xs:complexContent>
        <xs:restriction base="xs:anyType">
        </xs:restriction>
      </xs:complexContent>
    </xs:complexType>
  </xs:element>

</xs:schema>
```

Im Kontext der Typhierarchie wird dies nun verständlich: Wir müssen einen komplexen Typen für die Deklaration eines leeren Elements verwenden - selbst, wenn das leere Element keine Attribute enthalten soll -, da ein Element einfachen Typs auch Zeichendaten enthalten kann, ein leeres Element aber nicht. Kindelemente, also Element-Inhalt, sind bei leeren Elementen auch nicht erlaubt, aber dies ist kein Problem, da ein Element von komplexem Typ Element-Inhalt erlaubt, aber nicht fordert. Somit lassen wir bei der Typdefiniti-

on die Definition eines Inhaltsmodells einfach weg und lassen stattdessen keinen Inhalt zu. Dies wird ausgedrückt durch eine Einschränkung des *ur-typs* (`anyType`). Da das `<xs:restriction>`-Element leer bleibt, ist die Einschränkung absolut, d.h. es ist kein Inhalt erlaubt, das Element also somit als leer deklariert.

---

# XML-Schema und Namensräume

## Verwendung von Namensräumen in XML-Schema

Wie wir Sie bereits gesehen haben, dienen Namensräume in XML-Schema der Unterscheidung zwischen XML-Schema- und eigenen Elementen, Attributen und Typen.

XML-Schema kennt drei Namensräume:

- <http://www.w3.org/2001/XMLSchema>  
Der Namensraum für XML-Schema (muss immer im Wurzelement `<x:schema>` eines XML-Schemas angegeben werden)
- <http://www.w3.org/2001/XMLSchema-datatypes>  
enthält eine Kopie der eingebauten XML-Schema-Datentypen; dient der Verwendung in alternativen Schema-Applikationen, die die XML-Schema-Datentypen verwenden möchten
- <http://www.w3.org/2001/XMLSchema-instance>  
Für XML-Schema-Komponenten in Instanzdokumenten (z.B. `xsi:nil`)

XML-Schema verwendet Namensräume aber auch, um Instanzdokumente gegen das Markup eines bestimmten Namensraums zu validieren bzw. ein Dokument zu validieren, das Markup aus verschiedenen Namensräumen enthält.

## Namensräume bevölkern

Im Unterschied zu DTDs können Sie mit XML-Schema nicht nur die Struktur von Dokumenten festlegen, sondern gleichzeitig auch einen Namensraum erstellen (bevölkern, wie es im Fachjargon genannt wird) und somit auch XML-Instanzdokumente validieren, die Markup aus diesem Namensraum benutzen.

Dies geschieht über die Angabe eines sogenannten **Zielnamensraums** für ein bestimmtes XML-Schema. Diesen Zielnamensraum geben Sie im Wurzelement des XML-Schema über das Attribut `targetNamespace` an.

### **XML-Schema**

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

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns="http://www.meinServer.de/kunde"
  targetNamespace="http://www.meinServer.de/kunde">

  <xs:element name="Vorname" type="xs:string"/>
  <xs:element name="Zweitname" type="xs:string"/>
  <xs:element name="Nachname" type="xs:string"/>

  <xs:element name="Kunde">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="Vorname"/>
        <xs:element ref="Zweitname"/>
        <xs:element ref="Nachname"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

</xs:schema>
```

#### **Beachten Sie:**

Verweise auf Elemente die einem Zielnamensraum zugeordnet werden müssen über das entsprechende Präfix Informationen über den Namensraum erhalten, mit dem diese Komponenten verknüpft sind. In obigem Beispiel ist ein Standard-Namensraum für diese Verweise deklariert.

Alle globalen Elemente, die nun unterhalb von `<xs:schema>` deklariert werden gehören zum Zielnamensraum.

## **Inстанздokument**

```
<?xml version="1.0"?>
<kunde:Kunde
  xmlns:kunde="http://www.meinServer.de/kunde"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.meinServer.de/kunde
    ../xsd/sch053.xsd">
  <kunde:Vorname>Dieter</kunde:Vorname>
  <kunde:Zweitname>Thomas</kunde:Zweitname>
  <kunde:Nachname>Heck</kunde:Nachname>
</kunde:Kunde>
```

---

### **Beachten Sie:**

Der Zielnamensraum gilt nur für global deklarierte Komponenten des XML-Schema.

Würden Sie alle Kindelemente von Kunden innerhalb des komplexen Typs von Kunde definieren, sähe ein konformes Instanzdokument wie in folgendem Beispiel aus:

---

### **Beispiel SCH-54: Bevölkerung eines Namensraums bei nur einem globalen Element**

---

## **XML-Schema**

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

<xs:schema
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.meinServer.de/kunde">

  <xs:element name="Kunde">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="Vorname" type="xs:string"/>
        <xs:element name="Zweitname" type="xs:string"/>
        <xs:element name="Nachname" type="xs:string"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

</xs:schema>
```

## **Inстанздokument**

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

<kunde:Kunde
  xmlns:kunde="http://www.meinServer.de/kunde"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.meinServer.de/kunde
    ../xsd/sch054.xsd">
  <Vorname>Dieter</Vorname>
  <Zweitname>Thomas</Zweitname>
  <Nachname>Heck</Nachname>
</kunde:Kunde>
```

---

Über die Verknüpfung von Elementen mit einem Namensraum lassen sich diese Elemente auch in anderen Schemadokumenten wiederverwenden, ohne dass es zu Namenskonflikten und Mehrdeutigkeiten kommen kann.

## **Lokal deklarierten Komponenten in den Zielnamensraum miteinbeziehen**

### **Alle lokalen Komponenten in den Zielnamensraum einbeziehen**

Standardmäßig werden nur global deklarierte Komponenten mit dem Zielnamensraum verknüpft. Wollen Sie auch lokal deklarierte Elemente miteinbeziehen, so verwenden sie das Attribut `elementFormDefault` mit dem Wert `qualified` im `<xs:schema>`-Wurzelement. Für Attribute gibt es die Entsprechung `attributeFormDefault`. Standardwert der beiden Attribute ist `unqualified`.

### **XML-Schema**

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

<xs:schema
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.meinServer.de/kunde"
  elementFormDefault="qualified"
  attributeFormDefault="qualified">

  <xs:element name="Kunde">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="Vorname" type="xs:string"/>
        <xs:element name="Zweitname" type="xs:string"/>
        <xs:element name="Nachname" type="xs:string"/>
      </xs:sequence>
      <xs:attribute name="kundenID" type="xs:integer"/>
    </xs:complexType>
  </xs:element>

</xs:schema>
```

### **Inстанздokument**

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

<kunde:Kunde
  xmlns:kunde="http://www.meinServer.de/kunde"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.meinServer.de/kunde
    ../xsd/sch055.xsd"
  kunde:kundenID="2345">
  <kunde:Vorname>Dieter</kunde:Vorname>
  <kunde:Zweitname>Thomas</kunde:Zweitname>
  <kunde:Nachname>Heck</kunde:Nachname>
</kunde:Kunde>
```

## **Bestimmte lokale Komponenten in den Zielnamensraum einbeziehen**

Möchten Sie nicht alle, sondern nur bestimmte lokal deklarierte Elemente oder Attribute in den Zielnamensraum miteinbeziehen, so können Sie innerhalb der jeweiligen Element- bzw. Attribut-Deklaration das Attribut `form` mit dem Wert `qualified` verwenden. Umgekehrt können Sie auch ein lokal deklariertes Element über die Verwendung dieses Attributs mit dem Wert `unqualified` aus dem Zielnamensraum ausschließen. Dies macht natürlich nur Sinn, wenn Sie im `<xs:schema>`-Element das Attribut `elementFormDefault="qualified"` verwendet haben.

### ***Beispiel SCH-56: Bestimmte lokale Elemente in den Zielnamensraum einbeziehen***

---

## **XML-Schema**

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

<xs:schema
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.meinServer.de/kunde"
  elementFormDefault="unqualified"
  attributeFormDefault="unqualified">

  <xs:element name="Kunde">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="Vorname"
          type="xs:string"
          form="qualified"/>
```

## **Inстанздokument**

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

<kunde:Kunde
  xmlns:kunde="http://www.meinServer.de/kunde"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.meinServer.de/kunde
    ../xsd/sch056.xsd">
```

```
kundenID="2345">  
<kunde:Vorname>Dieter</kunde:Vorname>  
<Zweitname>Thomas</Zweitname>  
<Nachname>Heck</Nachname>  
</kunde:Kunde>
```

---