

# XSD - XML Schema Definition

## Definieren von XML-Dokumenten

Michael Dienert

Walther-Rathenau-Gewerbeschule  
Freiburg

15. September 2016

# Inhalt

Was sind XSD Dateien und warum soll man das lernen?

## Was sind XSD Dateien

- XSD ist die Abkürzung von *XML Schema Definition*.
- XSD ist selbst wieder eine XML-Anwendung.
- D.h. XSD-Dokumente müssen immer *wohlgeformte* XML-Dokumente sein.
- XSD beschreibt die **Struktur** eines XML-Dokuments.
- Mit XSD kann man die Struktur **genauer** beschreiben als mit der DTD.

## Was sind XSD Dateien

- XSD ist die Abkürzung von *XML Schema Definition*.
- XSD ist selbst wieder eine XML-Anwendung.
- D.h. XSD-Dokumente müssen immer *wohlgeformte* XML-Dokumente sein.
- XSD beschreibt die **Struktur** eines XML-Dokuments.
- Mit XSD kann man die Struktur **genauer** beschreiben als mit der DTD.

## Was sind XSD Dateien

- XSD ist die Abkürzung von *XML Schema Definition*.
- XSD ist selbst wieder eine XML-Anwendung.
- D.h. XSD-Dokumente müssen immer *wohlgeformte* XML-Dokumente sein.
- XSD beschreibt die **Struktur** eines XML-Dokuments.
- Mit XSD kann man die Struktur **genauer** beschreiben als mit der DTD.

## Was sind XSD Dateien

- XSD ist die Abkürzung von *XML Schema Definition*.
- XSD ist selbst wieder eine XML-Anwendung.
- D.h. XSD-Dokumente müssen immer *wohlgeformte XML-Dokumente* sein.
- XSD beschreibt die **Struktur** eines XML-Dokuments.
- Mit XSD kann man die Struktur **genauer** beschreiben als mit der DTD.

## Was sind XSD Dateien

- XSD ist die Abkürzung von *XML Schema Definition*.
- XSD ist selbst wieder eine XML-Anwendung.
- D.h. XSD-Dokumente müssen immer *wohlgeformte* XML-Dokumente sein.
- XSD beschreibt die **Struktur** eines XML-Dokuments.
- Mit XSD kann man die Struktur **genauer** beschreiben als mit der DTD.

## Was sind XSD Dateien

- XSD ist die Abkürzung von *XML Schema Definition*.
- XSD ist selbst wieder eine XML-Anwendung.
- D.h. XSD-Dokumente müssen immer *wohlgeformte* XML-Dokumente sein.
- XSD beschreibt die **Struktur** eines XML-Dokuments.
- Mit XSD kann man die Struktur **genauer** beschreiben als mit der DTD.

## Was sind XSD Dateien

Einige Erweiterungen gegenüber DTD:

- Genaue Angaben **wie oft** ein Element vorkommen darf.
- Datentypen
- XSD erlaubt auch eigene, komplexe Datentypen zu definieren → Objekte.
- Datentypen können von bestehenden Datentypen abgeleitet werden → Vererbung.

## Was sind XSD Dateien

### Einige Erweiterungen gegenüber DTD:

- Genaue Angaben **wie oft** ein Element vorkommen darf.
- Datentypen
- XSD erlaubt auch eigene, komplexe Datentypen zu definieren → Objekte.
- Datentypen können von bestehenden Datentypen abgeleitet werden → Vererbung.

## Was sind XSD Dateien

Einige Erweiterungen gegenüber DTD:

- Genaue Angaben **wie oft** ein Element vorkommen darf.
- Datentypen
- XSD erlaubt auch eigene, komplexe Datentypen zu definieren → Objekte.
- Datentypen können von bestehenden Datentypen abgeleitet werden → Vererbung.

## Was sind XSD Dateien

Einige Erweiterungen gegenüber DTD:

- Genaue Angaben **wie oft** ein Element vorkommen darf.
- Datentypen
- XSD erlaubt auch eigene, komplexe Datentypen zu definieren → Objekte.
- Datentypen können von bestehenden Datentypen abgeleitet werden → Vererbung.

## Was sind XSD Dateien

Einige Erweiterungen gegenüber DTD:

- Genaue Angaben **wie oft** ein Element vorkommen darf.
- Datentypen
- XSD erlaubt auch eigene, komplexe Datentypen zu definieren → Objekte.
- Datentypen können von bestehenden Datentypen abgeleitet werden → Vererbung.

## Was sind XSD Dateien

Einige Erweiterungen gegenüber DTD:

- Genaue Angaben **wie oft** ein Element vorkommen darf.
- Datentypen
- XSD erlaubt auch eigene, komplexe Datentypen zu definieren → Objekte.
- Datentypen können von bestehenden Datentypen abgeleitet werden → Vererbung.

## Wozu lernen wir XSD ?

- XSD ist **der** weltweite Standard, mit dem XML-Dateien definiert werden.
- Java-XML-Bindings erzeugen aus einer XSD-Datei eine Klasse, deren Objekte exakt die zugehörige XML-Datei abbilden.
- Man kann die Java-Objekte mit allen Attributen in die XML-Datei sichern (nur die Attribute!).
- Man kann aus der XML-Datei wieder die Java-Objekte erzeugen.
- Man kann sogar aus einer Java-Klasse eine XSD-Datei erzeugen!

## Wozu lernen wir XSD ?

- XSD ist **der** weltweite Standard, mit dem XML-Dateien definiert werden.
- Java-XML-Bindings erzeugen aus einer XSD-Datei eine Klasse, deren Objekte exakt die zugehörige XML-Datei abbilden.
- Man kann die Java-Objekte mit allen Attributen in die XML-Datei sichern (nur die Attribute!).
- Man kann aus der XML-Datei wieder die Java-Objekte erzeugen.
- Man kann sogar aus einer Java-Klasse eine XSD-Datei erzeugen!

## Wozu lernen wir XSD ?

- XSD ist **der** weltweite Standard, mit dem XML-Dateien definiert werden.
- Java-XML-Bindings erzeugen aus einer XSD-Datei eine Klasse, deren Objekte exakt die zugehörige XML-Datei abbilden.
- Man kann die Java-Objekte mit allen Attributen in die XML-Datei sichern (nur die Attribute!).
- Man kann aus der XML-Datei wieder die Java-Objekte erzeugen.
- Man kann sogar aus einer Java-Klasse eine XSD-Datei erzeugen!

## Wozu lernen wir XSD ?

- XSD ist **der** weltweite Standard, mit dem XML-Dateien definiert werden.
- Java-XML-Bindings erzeugen aus einer XSD-Datei eine Klasse, deren Objekte exakt die zugehörige XML-Datei abbilden.
- Man kann die Java-Objekte mit allen Attributen in die XML-Datei sichern (nur die Attribute!).
- Man kann aus der XML-Datei wieder die Java-Objekte erzeugen.
- Man kann sogar aus einer Java-Klasse eine XSD-Datei erzeugen!

## Wozu lernen wir XSD ?

- XSD ist **der** weltweite Standard, mit dem XML-Dateien definiert werden.
- Java-XML-Bindings erzeugen aus einer XSD-Datei eine Klasse, deren Objekte exakt die zugehörige XML-Datei abbilden.
- Man kann die Java-Objekte mit allen Attributen in die XML-Datei sichern (nur die Attribute!).
- Man kann aus der XML-Datei wieder die Java-Objekte erzeugen.
- Man kann sogar aus einer Java-Klasse eine XSD-Datei erzeugen!

## Wozu lernen wir XSD ?

- XSD ist **der** weltweite Standard, mit dem XML-Dateien definiert werden.
- Java-XML-Bindings erzeugen aus einer XSD-Datei eine Klasse, deren Objekte exakt die zugehörige XML-Datei abbilden.
- Man kann die Java-Objekte mit allen Attributen in die XML-Datei sichern (nur die Attribute!).
- Man kann aus der XML-Datei wieder die Java-Objekte erzeugen.
- Man kann sogar aus einer Java-Klasse eine XSD-Datei erzeugen!

## Erstes Beispiel

```
1 <?xml version="1.0"?>
2 <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
3 <xs:element name="kontakte">
4 <xs:complexType>
5 <xs:sequence>
6 <xs:element name="Person" maxOccurs="unbounded">
7 <xs:complexType>
8 <xs:sequence>
9 <xs:element name="name" type="xs:string" />
10 <xs:element name="adresse" type="xs:string"
11 minOccurs="0" />
12 </xs:sequence>
13 <xs:attribute name="id" type="xs:long" use="required
14 " />
15 </xs:complexType>
16 </xs:element>
17 </xs:sequence>
18 </xs:complexType>
19 </xs:element>
20 </xs:schema>
```

## Erstes Beispiel

```
1 | <?xml version="1.0"?>
2 | <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
3 | <xs:element name="kontakte">
4 |   <xs:complexType>
5 |     <xs:sequence>
6 |       <xs:element name="Person" maxOccurs="unbounded">
7 |         <xs:complexType>
8 |           <xs:sequence>
9 |             <xs:element name="name" type="xs:string" />
10 |            <xs:element name="adresse" type="xs:string"
11 |              minOccurs="0" />
12 |           </xs:sequence>
13 |           <xs:attribute name="id" type="xs:long" use="required
14 |             " />
15 |         </xs:complexType>
16 |       </xs:element>
17 |     </xs:sequence>
18 |   </xs:complexType>
19 | </xs:element>
20 | </xs:schema>
```

## Erstes Beispiel

```
1 | <?xml version="1.0"?>
2 | <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
3 | <xs:element name="kontakte">
4 |   <xs:complexType>
5 |     <xs:sequence>
6 |       <xs:element name="Person" maxOccurs="unbounded">
7 |         <xs:complexType>
8 |           <xs:sequence>
9 |             <xs:element name="name" type="xs:string" />
10 |            <xs:element name="adresse" type="xs:string"
11 |              minOccurs="0" />
12 |           </xs:sequence>
13 |           <xs:attribute name="id" type="xs:long" use="required
14 |             " />
15 |         </xs:complexType>
16 |       </xs:element>
17 |     </xs:sequence>
18 |   </xs:complexType>
19 | </xs:element>
20 | </xs:schema>
```

## Erstes Beispiel

```
1 <?xml version="1.0"?>
2 <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
3 <xs:element name="kontakte">
4 <xs:complexType>
5 <xs:sequence>
6 <xs:element name="Person" maxOccurs="unbounded">
7 <xs:complexType>
8 <xs:sequence>
9 <xs:element name="name" type="xs:string" />
10 <xs:element name="adresse" type="xs:string"
11 minOccurs="0" />
12 </xs:sequence>
13 <xs:attribute name="id" type="xs:long" use="required"
14 " />
15 </xs:complexType>
16 </xs:element>
17 </xs:sequence>
18 </xs:complexType>
19 </xs:element>
20 </xs:schema>
```

## Erstes Beispiel

```
1 <?xml version="1.0"?>
2 <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
3 <xs:element name="kontakte">
4 <xs:complexType>
5 <xs:sequence>
6 <xs:element name="Person" maxOccurs="unbounded">
7 <xs:complexType>
8 <xs:sequence>
9 <xs:element name="name" type="xs:string" />
10 <xs:element name="adresse" type="xs:string"
11 minOccurs="0" />
12 </xs:sequence>
13 <xs:attribute name="id" type="xs:long" use="required"
14 " />
15 </xs:complexType>
16 </xs:element>
17 </xs:sequence>
18 </xs:complexType>
19 </xs:element>
20 </xs:schema>
```

## Erstes Beispiel

```
1 <?xml version="1.0"?>
2 <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
3 <xs:element name="kontakte">
4 <xs:complexType>
5 <xs:sequence>
6 <xs:element name="Person" maxOccurs="unbounded">
7 <xs:complexType>
8 <xs:sequence>
9 <xs:element name="name" type="xs:string" />
10 <xs:element name="adresse" type="xs:string"
11 minOccurs="0" />
12 </xs:sequence>
13 <xs:attribute name="id" type="xs:long" use="required"
14 " />
15 </xs:complexType>
16 </xs:element>
17 </xs:sequence>
18 </xs:complexType>
19 </xs:element>
20 </xs:schema>
```

## Erstes Beispiel

```
1 <?xml version="1.0"?>
2 <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
3 <xs:element name="kontakte">
4 <xs:complexType>
5 <xs:sequence>
6 <xs:element name="Person" maxOccurs="unbounded">
7 <xs:complexType>
8 <xs:sequence>
9 <xs:element name="name" type="xs:string" />
10 <xs:element name="adresse" type="xs:string"
11 minOccurs="0" />
12 </xs:sequence>
13 <xs:attribute name="id" type="xs:long" use="required"
14 " />
15 </xs:complexType>
16 </xs:element>
17 </xs:sequence>
18 </xs:complexType>
19 </xs:element>
20 </xs:schema>
```

## Erstes Beispiel

```
1 <?xml version="1.0"?>
2 <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
3 <xs:element name="kontakte">
4 <xs:complexType>
5 <xs:sequence>
6 <xs:element name="Person" maxOccurs="unbounded">
7 <xs:complexType>
8 <xs:sequence>
9 <xs:element name="name" type="xs:string" />
10 <xs:element name="adresse" type="xs:string"
11 minOccurs="0" />
12 </xs:sequence>
13 <xs:attribute name="id" type="xs:long" use="required"
14 " />
15 </xs:complexType>
16 </xs:element>
17 </xs:sequence>
18 </xs:complexType>
19 </xs:element>
20 </xs:schema>
```

## Erstes Beispiel

```
1 <?xml version="1.0"?>
2 <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
3 <xs:element name="kontakte">
4 <xs:complexType>
5 <xs:sequence>
6 <xs:element name="Person" maxOccurs="unbounded">
7 <xs:complexType>
8 <xs:sequence>
9 <xs:element name="name" type="xs:string" />
10 <xs:element name="adresse" type="xs:string"
11 minOccurs="0" />
12 </xs:sequence>
13 <xs:attribute name="id" type="xs:long" use="required"
14 " />
15 </xs:complexType>
16 </xs:element>
17 </xs:sequence>
18 </xs:complexType>
19 </xs:element>
20 </xs:schema>
```

## Erstes Beispiel

```
1 <?xml version="1.0"?>
2 <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
3 <xs:element name="kontakte">
4 <xs:complexType>
5 <xs:sequence>
6 <xs:element name="Person" maxOccurs="unbounded">
7 <xs:complexType>
8 <xs:sequence>
9 <xs:element name="name" type="xs:string" />
10 <xs:element name="adresse" type="xs:string"
    minOccurs="0" />
11 </xs:sequence>
12 <xs:attribute name="id" type="xs:long" use="required"
    "/>
13 </xs:complexType>
14 </xs:element>
15 </xs:sequence>
16 </xs:complexType>
17 </xs:element>
18 </xs:schema>
```

## Erstes Beispiel

```
1 <?xml version="1.0"?>
2 <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
3 <xs:element name="kontakte">
4 <xs:complexType>
5 <xs:sequence>
6 <xs:element name="Person" maxOccurs="unbounded">
7 <xs:complexType>
8 <xs:sequence>
9 <xs:element name="name" type="xs:string" />
10 <xs:element name="adresse" type="xs:string"
    minOccurs="0" />
11 </xs:sequence>
12 <xs:attribute name="id" type="xs:long" use="required"
    "/>
13 </xs:complexType>
14 </xs:element>
15 </xs:sequence>
16 </xs:complexType>
17 </xs:element>
18 </xs:schema>
```

## Erstes Beispiel

```
1 <?xml version="1.0"?>
2 <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
3 <xs:element name="kontakte">
4 <xs:complexType>
5 <xs:sequence>
6 <xs:element name="Person" maxOccurs="unbounded">
7 <xs:complexType>
8 <xs:sequence>
9 <xs:element name="name" type="xs:string" />
10 <xs:element name="adresse" type="xs:string"
11 minOccurs="0" />
12 </xs:sequence>
13 <xs:attribute name="id" type="xs:long" use="required"
14 " />
15 </xs:complexType>
16 </xs:element>
17 </xs:sequence>
18 </xs:complexType>
19 </xs:element>
20 </xs:schema>
```

## Erstes Beispiel

```
1 <?xml version="1.0"?>
2 <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
3 <xs:element name="kontakte">
4 <xs:complexType>
5 <xs:sequence>
6 <xs:element name="Person" maxOccurs="unbounded">
7 <xs:complexType>
8 <xs:sequence>
9 <xs:element name="name" type="xs:string" />
10 <xs:element name="adresse" type="xs:string"
11 minOccurs="0" />
12 </xs:sequence>
13 <xs:attribute name="id" type="xs:long" use="required
14 " />
15 </xs:complexType>
16 </xs:element>
17 </xs:sequence>
18 </xs:complexType>
19 </xs:element>
20 </xs:schema>
```

## Erstes Beispiel

```
1 <?xml version="1.0"?>
2 <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
3 <xs:element name="kontakte">
4 <xs:complexType>
5 <xs:sequence>
6 <xs:element name="Person" maxOccurs="unbounded">
7 <xs:complexType>
8 <xs:sequence>
9 <xs:element name="name" type="xs:string" />
10 <xs:element name="adresse" type="xs:string"
11 minOccurs="0" />
12 </xs:sequence>
13 <xs:attribute name="id" type="xs:long" use="required
14 " />
15 </xs:complexType>
16 </xs:element>
17 </xs:sequence>
18 </xs:complexType>
19 </xs:element>
20 </xs:schema>
```

## Erstes Beispiel

```
1 <?xml version="1.0"?>
2 <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
3 <xs:element name="kontakte">
4 <xs:complexType>
5 <xs:sequence>
6 <xs:element name="Person" maxOccurs="unbounded">
7 <xs:complexType>
8 <xs:sequence>
9 <xs:element name="name" type="xs:string" />
10 <xs:element name="adresse" type="xs:string"
11 minOccurs="0" />
12 </xs:sequence>
13 <xs:attribute name="id" type="xs:long" use="required
14 " />
15 </xs:complexType>
16 </xs:element>
17 </xs:sequence>
18 </xs:complexType>
19 </xs:element>
20 </xs:schema>
```

## Erstes Beispiel

```
1 <?xml version="1.0"?>
2 <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
3 <xs:element name="kontakte">
4 <xs:complexType>
5 <xs:sequence>
6 <xs:element name="Person" maxOccurs="unbounded">
7 <xs:complexType>
8 <xs:sequence>
9 <xs:element name="name" type="xs:string" />
10 <xs:element name="adresse" type="xs:string"
11 minOccurs="0" />
12 </xs:sequence>
13 <xs:attribute name="id" type="xs:long" use="required
14 " />
15 </xs:complexType>
16 </xs:element>
17 </xs:sequence>
18 </xs:complexType>
19 </xs:element>
20 </xs:schema>
```

## Erstes Beispiel

```
1 <?xml version="1.0"?>
2 <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
3 <xs:element name="kontakte">
4 <xs:complexType>
5 <xs:sequence>
6 <xs:element name="Person" maxOccurs="unbounded">
7 <xs:complexType>
8 <xs:sequence>
9 <xs:element name="name" type="xs:string" />
10 <xs:element name="adresse" type="xs:string"
11 minOccurs="0" />
12 </xs:sequence>
13 <xs:attribute name="id" type="xs:long" use="required
14 " />
15 </xs:complexType>
16 </xs:element>
17 </xs:sequence>
18 </xs:complexType>
19 </xs:element>
20 </xs:schema>
```

## Erstes Beispiel

```
1 <?xml version="1.0"?>
2 <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
3 <xs:element name="kontakte">
4 <xs:complexType>
5 <xs:sequence>
6 <xs:element name="Person" maxOccurs="unbounded">
7 <xs:complexType>
8 <xs:sequence>
9 <xs:element name="name" type="xs:string" />
10 <xs:element name="adresse" type="xs:string"
11 minOccurs="0" />
12 </xs:sequence>
13 <xs:attribute name="id" type="xs:long" use="required
14 " />
15 </xs:complexType>
16 </xs:element>
17 </xs:sequence>
18 </xs:complexType>
19 </xs:element>
20 </xs:schema>
```

## Erstes Beispiel

```
1 <?xml version="1.0"?>
2 <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
3 <xs:element name="kontakte">
4 <xs:complexType>
5 <xs:sequence>
6 <xs:element name="Person" maxOccurs="unbounded">
7 <xs:complexType>
8 <xs:sequence>
9 <xs:element name="name" type="xs:string" />
10 <xs:element name="adresse" type="xs:string"
11 minOccurs="0" />
12 </xs:sequence>
13 <xs:attribute name="id" type="xs:long" use="required
14 " />
15 </xs:complexType>
16 </xs:element>
17 </xs:sequence>
18 </xs:complexType>
19 </xs:element>
20 </xs:schema>
```

## Erstes Beispiel

```
1 <?xml version="1.0"?>
2 <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
3 <xs:element name="kontakte">
4 <xs:complexType>
5 <xs:sequence>
6 <xs:element name="Person" maxOccurs="unbounded">
7 <xs:complexType>
8 <xs:sequence>
9 <xs:element name="name" type="xs:string" />
10 <xs:element name="adresse" type="xs:string"
11 minOccurs="0" />
12 </xs:sequence>
13 <xs:attribute name="id" type="xs:long" use="required
14 " />
15 </xs:complexType>
16 </xs:element>
17 </xs:sequence>
18 </xs:complexType>
19 </xs:element>
20 </xs:schema>
```

## Details einer XSD-Datei

- `xs:element` Element mit Daten
- Attribute von `xs:element`:

`name` Elementname

`type` Datentyp: siehe nächste Folie

`minOccurs` Mindestanzahl; `minOccurs` weglassen:  
Anzahl=1

`maxOccurs` Maximalanzahl; `maxOccurs` weglassen:  
Anzahl=1; `maxOccurs=unbounded` =  
beliebig viele

## Details einer XSD-Datei

- **xs:element** Element mit Daten

- Attribute von `xs:element`:

`name` Elementname

`type` Datentyp: siehe nächste Folie

`minOccurs` Mindestanzahl; `minOccurs` weglassen:  
Anzahl=1

`maxOccurs` Maximalanzahl; `maxOccurs` weglassen:  
Anzahl=1; `maxOccurs=unbounded` =  
beliebig viele

## Details einer XSD-Datei

- **xs:element** Element mit Daten
- Attribute von `xs:element`:

`name` Elementname

`type` Datentyp: siehe nächste Folie

`minOccurs` Mindestanzahl; `minOccurs` weglassen:  
Anzahl=1

`maxOccurs` Maximalanzahl; `maxOccurs` weglassen:  
Anzahl=1; `maxOccurs=unbounded` =  
beliebig viele

## Details einer XSD-Datei

- **xs:element** Element mit Daten
- Attribute von `xs:element`:

<code>name</code>	Elementname
<code>type</code>	Datentyp: siehe nächste Folie
<code>minOccurs</code>	Mindestanzahl; <code>minOccurs</code> weglassen: Anzahl=1
<code>maxOccurs</code>	Maximalanzahl; <code>maxOccurs</code> weglassen: Anzahl=1; <code>maxOccurs=unbounded</code> = beliebig viele

## Details einer XSD-Datei

- **xs:element** Element mit Daten
- Attribute von `xs:element`:

`name` Elementname

`type` Datentyp: siehe nächste Folie

`minOccurs` Mindestanzahl; `minOccurs` weglassen:  
Anzahl=1

`maxOccurs` Maximalanzahl; `maxOccurs` weglassen:  
Anzahl=1; `maxOccurs=unbounded` =  
beliebig viele

## Details einer XSD-Datei

- **xs:element** Element mit Daten
- Attribute von `xs:element`:

`name` Elementname

`type` Datentyp: siehe nächste Folie

`minOccurs` Mindestanzahl; `minOccurs` weglassen:  
Anzahl=1

`maxOccurs` Maximalanzahl; `maxOccurs` weglassen:  
Anzahl=1; `maxOccurs=unbounded` =  
beliebig viele

## Details einer XSD-Datei

- **xs:element** Element mit Daten
- Attribute von `xs:element`:

**name** Elementname

**type** Datentyp: siehe nächste Folie

**minOccurs** Mindestanzahl; `minOccurs` weglassen:  
Anzahl=1

**maxOccurs** Maximalanzahl; `maxOccurs` weglassen:  
Anzahl=1; `maxOccurs=unbounded` =  
beliebig viele

## Details einer XSD-Datei

### XSD-Standard-Datentypen - Simple Types

xs:string

xs:decimal

xs:integer

xs:float

xs:boolean

xs:date

xs:time

xs:anyType

## Details einer XSD-Datei

### **XSD-Standard-Datentypen - Simple Types**

xs:string

xs:decimal

xs:integer

xs:float

xs:boolean

xs:date

xs:time

xs:anyType

## Details einer XSD-Datei

### **XSD-Standard-Datentypen - Simple Types**

`xs:string`

`xs:decimal`

`xs:integer`

`xs:float`

`xs:boolean`

`xs:date`

`xs:time`

`xs:anyType`

## Details einer XSD-Datei

### **XSD-Standard-Datentypen - Simple Types**

xs:string

xs:decimal

xs:integer

xs:float

xs:boolean

xs:date

xs:time

xs:anyType

## Details einer XSD-Datei

### **XSD-Standard-Datentypen - Simple Types**

xs:string

xs:decimal

xs:integer

xs:float

xs:boolean

xs:date

xs:time

xs:anyType

## Details einer XSD-Datei

### **XSD-Standard-Datentypen - Simple Types**

xs:string

xs:decimal

xs:integer

xs:float

xs:boolean

xs:date

xs:time

xs:anyType

## Details einer XSD-Datei

### **XSD-Standard-Datentypen - Simple Types**

xs:string

xs:decimal

xs:integer

xs:float

xs:boolean

xs:date

xs:time

xs:anyType

## Details einer XSD-Datei

### **XSD-Standard-Datentypen - Simple Types**

xs:string

xs:decimal

xs:integer

xs:float

xs:boolean

xs:date

xs:time

xs:anyType

## Details einer XSD-Datei

### **XSD-Standard-Datentypen - Simple Types**

xs:string

xs:decimal

xs:integer

xs:float

xs:boolean

xs:date

xs:time

xs:anyType

## Details einer XSD-Datei

### **XSD-Standard-Datentypen - Simple Types**

xs:string

xs:decimal

xs:integer

xs:float

xs:boolean

xs:date

xs:time

xs:anyType

## Beispiele für einfache Elemente

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

Beispiel:

```
<name>alfred</name>
```

---

```
<xs:element name="adresse" type="xs:string" />
```

Beispiel:

```
<adresse>friedrichstrasse 51, 79098 freiburg</adresse>
```

---

```
<xs:element name="id" type="xs:long" use="required" />
```

Beispiel:

```
<id>1234</id>
```

## Beispiele für einfache Elemente

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

Beispiel:

```
<name>alfred</name>
```

---

```
<xs:element name="adresse" type="xs:string" />
```

Beispiel:

```
<adresse>friedrichstrasse 51, 79098 freiburg</adresse>
```

---

```
<xs:element name="id" type="xs:long" use="required" />
```

Beispiel:

```
<id>1234</id>
```

## Beispiele für einfache Elemente

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

Beispiel:

```
<name>alfred</name>
```

---

```
<xs:element name="adresse" type="xs:string" />
```

Beispiel:

```
<adresse>friedrichstrasse 51, 79098 freiburg</adresse>
```

---

```
<xs:element name="id" type="xs:long" use="required" />
```

Beispiel:

```
<id>1234</id>
```

## Beispiele für einfache Elemente

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

Beispiel:

```
<name>alfred</name>
```

---

```
<xs:element name="adresse" type="xs:string" />
```

Beispiel:

```
<adresse>friedrichstrasse 51, 79098 freiburg</adresse>
```

---

```
<xs:element name="id" type="xs:long" use="required" />
```

Beispiel:

```
<id>1234</id>
```

## Beispiele für einfache Elemente

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

Beispiel:

```
<name>alfred</name>
```

---

```
<xs:element name="adresse" type="xs:string" />
```

Beispiel:

```
<adresse>friedrichstrasse 51, 79098 freiburg</adresse>
```

---

```
<xs:element name="id" type="xs:long" use="required" />
```

Beispiel:

```
<id>1234</id>
```

## Beispiele für einfache Elemente

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

Beispiel:

```
<name>alfred</name>
```

---

```
<xs:element name="adresse" type="xs:string" />
```

Beispiel:

```
<adresse>friedrichstrasse 51, 79098 freiburg</adresse>
```

---

```
<xs:element name="id" type="xs:long" use="required" />
```

Beispiel:

```
<id>1234</id>
```

## Beispiele für einfache Elemente

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

Beispiel:

```
<name>alfred</name>
```

---

```
<xs:element name="adresse" type="xs:string" />
```

Beispiel:

```
<adresse>friedrichstrasse 51, 79098 freiburg</adresse>
```

---

```
<xs:element name="id" type="xs:long" use="required" />
```

Beispiel:

```
<id>1234</id>
```

## Beispiele für einfache Elemente

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

Beispiel:

```
<name>alfred</name>
```

---

```
<xs:element name="adresse" type="xs:string" />
```

Beispiel:

```
<adresse>friedrichstrasse 51, 79098 freiburg</adresse>
```

---

```
<xs:element name="id" type="xs:long" use="required" />
```

Beispiel:

```
<id>1234</id>
```

## Beispiele für einfache Elemente

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

Beispiel:

```
<name>alfred</name>
```

---

```
<xs:element name="adresse" type="xs:string" />
```

Beispiel:

```
<adresse>friedrichstrasse 51, 79098 freiburg</adresse>
```

---

```
<xs:element name="id" type="xs:long" use="required" />
```

Beispiel:

```
<id>1234</id>
```

## Beispiele für einfache Elemente

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

Beispiel:

```
<name>alfred</name>
```

---

```
<xs:element name="adresse" type="xs:string" />
```

Beispiel:

```
<adresse>friedrichstrasse 51, 79098 freiburg</adresse>
```

---

```
<xs:element name="id" type="xs:long" use="required" />
```

Beispiel:

```
<id>1234</id>
```

## Komplexe Elemente

- Elemente, die *Kindselemente* oder *Attribute* oder beides enthalten werden mit `xs:complexType` definiert
- Ist die Reihenfolge der Kinder fest, werden diese zusätzlich in ein `xs:sequence` - Element eingebettet.
- Ist die Reihenfolge beliebig, nimmt man stattdessen das Element `xs:choice`
- Selbstverständlich darf man auch innerhalb eines `xs:sequence`-Elements `xs:choice`-Elemente einbetten und umgekehrt.
- man kann so feste und beliebige Reihenfolge mischen.

## Komplexe Elemente

- Elemente, die *Kindselemente* oder *Attribute* oder beides enthalten werden mit `xs:complexType` definiert
- Ist die Reihenfolge der Kinder fest, werden diese zusätzlich in ein `xs:sequence` - Element eingebettet.
- Ist die Reihenfolge beliebig, nimmt man stattdessen das Element `xs:choice`
- Selbstverständlich darf man auch innerhalb eines `xs:sequence`-Elements `xs:choice`-Elemente einbetten und umgekehrt.
- man kann so feste und beliebige Reihenfolge mischen.

## Komplexe Elemente

- Elemente, die *Kindselemente* oder *Attribute* oder beides enthalten werden mit `xs:complexType` definiert
- Ist die Reihenfolge der Kinder fest, werden diese zusätzlich in ein `xs:sequence` - Element eingebettet.
- Ist die Reihenfolge beliebig, nimmt man stattdessen das Element `xs:choice`
- Selbstverständlich darf man auch innerhalb eines `xs:sequence`-Elements `xs:choice`-Elemente einbetten und umgekehrt.
- man kann so feste und beliebige Reihenfolge mischen.

## Komplexe Elemente

- Elemente, die *Kindselemente* oder *Attribute* oder beides enthalten werden mit `xs:complexType` definiert
- Ist die Reihenfolge der Kinder fest, werden diese zusätzlich in ein `xs:sequence` - Element eingebettet.
- Ist die Reihenfolge beliebig, nimmt man stattdessen das Element `xs:choice`
- Selbstverständlich darf man auch innerhalb eines `xs:sequence`-Elements `xs:choice`-Elemente einbetten und umgekehrt.
- man kann so feste und beliebige Reihenfolge mischen.

## Komplexe Elemente

- Elemente, die *Kindselemente* oder *Attribute* oder beides enthalten werden mit `xs:complexType` definiert
- Ist die Reihenfolge der Kinder fest, werden diese zusätzlich in ein `xs:sequence` - Element eingebettet.
- Ist die Reihenfolge beliebig, nimmt man stattdessen das Element `xs:choice`
- Selbstverständlich darf man auch innerhalb eines `xs:sequence`-Elements `xs:choice`-Elemente einbetten und umgekehrt.
- man kann so feste und beliebige Reihenfolge mischen.

## Komplexe Elemente

- Elemente, die *Kindselemente* oder *Attribute* oder beides enthalten werden mit `xs:complexType` definiert
- Ist die Reihenfolge der Kinder fest, werden diese zusätzlich in ein `xs:sequence` - Element eingebettet.
- Ist die Reihenfolge beliebig, nimmt man stattdessen das Element `xs:choice`
- Selbstverständlich darf man auch innerhalb eines `xs:sequence`-Elements `xs:choice`-Elemente einbetten und umgekehrt.
- man kann so feste und beliebige Reihenfolge mischen.

## Ein Beispiel für xs:complexType

```
1 <xs:element name="Person" maxOccurs="unbounded">
2   <xs:complexType>
3     <xs:sequence>
4       <xs:element name="name" type="xs:string" />
5       <xs:element name="adresse" type="xs:string" minOccurs=
        "0" />
6     </xs:sequence>
7   </xs:complexType>
8 </xs:element>
```

Beispiel eines validen Elements:

```
<Person>
  <name>neuman, alfred e.</name>
  <adresse>locaRoca 666, 1234 dancelingen</adresse>
</Person>
```

## Ein Beispiel für xs:complexType

```
1 | <xs:element name="Person" maxOccurs="unbounded">
2 |   <xs:complexType>
3 |     <xs:sequence>
4 |       <xs:element name="name" type="xs:string" />
5 |       <xs:element name="adresse" type="xs:string" minOccurs=
      "0" />
6 |     </xs:sequence>
7 |   </xs:complexType>
8 | </xs:element>
```

Beispiel eines validen Elements:

```
<Person>
  <name>neuman, alfred e.</name>
  <adresse>locaRoca 666, 1234 dancelingen</adresse>
</Person>
```

## Ein Beispiel für xs:complexType

```
1 <xs:element name="Person" maxOccurs="unbounded">
2   <xs:complexType>
3     <xs:sequence>
4       <xs:element name="name" type="xs:string" />
5       <xs:element name="adresse" type="xs:string" minOccurs=
        "0" />
6     </xs:sequence>
7   </xs:complexType>
8 </xs:element>
```

Beispiel eines validen Elements:

```
<Person>
  <name>neuman, alfred e.</name>
  <adresse>locaRoca 666, 1234 dancelingen</adresse>
</Person>
```

## Ein Beispiel für xs:complexType

```
1 <xs:element name="Person" maxOccurs="unbounded">
2   <xs:complexType>
3     <xs:sequence>
4       <xs:element name="name" type="xs:string" />
5       <xs:element name="adresse" type="xs:string" minOccurs=
        "0" />
6     </xs:sequence>
7   </xs:complexType>
8 </xs:element>
```

Beispiel eines validen Elements:

```
<Person>
  <name>neuman, alfred e.</name>
  <adresse>locaRoca 666, 1234 dancelingen</adresse>
</Person>
```

## Ein Beispiel für xs:complexType

```
1 <xs:element name="Person" maxOccurs="unbounded">
2   <xs:complexType>
3     <xs:sequence>
4       <xs:element name="name" type="xs:string" />
5       <xs:element name="adresse" type="xs:string" minOccurs=
6         "0" />
7     </xs:sequence>
8   </xs:complexType>
</xs:element>
```

Beispiel eines validen Elements:

```
<Person>
  <name>neuman, alfred e.</name>
  <adresse>locaRoca 666, 1234 dancelingen</adresse>
</Person>
```

## Ein Beispiel für xs:complexType

```
1 <xs:element name="Person" maxOccurs="unbounded">
2   <xs:complexType>
3     <xs:sequence>
4       <xs:element name="name" type="xs:string" />
5       <xs:element name="adresse" type="xs:string" minOccurs=
6         "0" />
7     </xs:sequence>
8   </xs:complexType>
</xs:element>
```

Beispiel eines validen Elements:

```
<Person>
  <name>neuman, alfred e.</name>
  <adresse>locaRoca 666, 1234 dancelingen</adresse>
</Person>
```

## Ein Beispiel für xs:complexType

```
1 <xs:element name="Person" maxOccurs="unbounded">
2   <xs:complexType>
3     <xs:sequence>
4       <xs:element name="name" type="xs:string" />
5       <xs:element name="adresse" type="xs:string" minOccurs=
        "0" />
6     </xs:sequence>
7   </xs:complexType>
8 </xs:element>
```

Beispiel eines validen Elements:

```
<Person>
  <name>neuman, alfred e.</name>
  <adresse>locaRoca 666, 1234 dancelingen</adresse>
</Person>
```

## Ein Beispiel für xs:complexType

```
1 <xs:element name="Person" maxOccurs="unbounded">
2   <xs:complexType>
3     <xs:sequence>
4       <xs:element name="name" type="xs:string" />
5       <xs:element name="adresse" type="xs:string" minOccurs=
        "0" />
6     </xs:sequence>
7   </xs:complexType>
8 </xs:element>
```

Beispiel eines validen Elements:

```
<Person>
  <name>neuman, alfred e.</name>
  <adresse>locaRoca 666, 1234 dancelingen</adresse>
</Person>
```

## Ein Beispiel für xs:complexType

```
1 <xs:element name="Person" maxOccurs="unbounded">
2   <xs:complexType>
3     <xs:sequence>
4       <xs:element name="name" type="xs:string" />
5       <xs:element name="adresse" type="xs:string" minOccurs=
6         "0" />
7     </xs:sequence>
8   </xs:complexType>
</xs:element>
```

Beispiel eines validen Elements:

```
<Person>
  <name>neuman, alfred e.</name>
  <adresse>locaRoca 666, 1234 dancelingen</adresse>
</Person>
```

## Ein Beispiel für xs:complexType

```
1 <xs:element name="Person" maxOccurs="unbounded">
2   <xs:complexType>
3     <xs:sequence>
4       <xs:element name="name" type="xs:string" />
5       <xs:element name="adresse" type="xs:string" minOccurs=
6         "0" />
7     </xs:sequence>
8   </xs:complexType>
</xs:element>
```

Beispiel eines validen Elements:

```
<Person>
  <name>neuman, alfred e.</name>
  <adresse>locaRoca 666, 1234 dancelingen</adresse>
</Person>
```

## Ein Beispiel für xs:complexType

```
1 <xs:element name="Person" maxOccurs="unbounded">
2   <xs:complexType>
3     <xs:sequence>
4       <xs:element name="name" type="xs:string" />
5       <xs:element name="adresse" type="xs:string" minOccurs=
6         "0" />
7     </xs:sequence>
8   </xs:complexType>
</xs:element>
```

### Beispiel eines validen Elements:

```
<Person>
  <name>neuman, alfred e.</name>
  <adresse>locaRoca 666, 1234 dancelingen</adresse>
</Person>
```

## Attribute

- Das Element `xs:attribute` legt die *Attribute* eines Elements fest.
- Es kann nur innerhalb einer `xs:complexType` - Umgebung verwendet werden.
- `xs:attribute` hat folgende Attribute:
  - `use` kann den Wert "required" oder "optional" haben.
  - `type` legt den Datentyp fest. Mögliche Basistypen sind die selben wie bei den Elementen.
- Enthält `xs:complexType` ein `xs:sequence` - Element, **muss** das Element `xs:attribute` nach dem `xs:sequence` - Element stehen!

## Attribute

- Das Element `xs:attribute` legt die *Attribute* eines Elements fest.
- Es kann nur innerhalb einer `xs:complexType` - Umgebung verwendet werden.
- `xs:attribute` hat folgende Attribute:
  - `use` kann den Wert "required" oder "optional" haben.
  - `type` legt den Datentyp fest. Mögliche Basistypen sind die selben wie bei den Elementen.
- Enthält `xs:complexType` ein `xs:sequence` - Element, **muss** das Element `xs:attribute` nach dem `xs:sequence` - Element stehen!

## Attribute

- Das Element `xs:attribute` legt die *Attribute* eines Elements fest.
- Es kann nur innerhalb einer `xs:complexType` - Umgebung verwendet werden.
- `xs:attribute` hat folgende Attribute:
  - `use` kann den Wert "required" oder "optional" haben.
  - `type` legt den Datentyp fest. Mögliche Basistypen sind die selben wie bei den Elementen.
- Enthält `xs:complexType` ein `xs:sequence` - Element, **muss** das Element `xs:attribute` nach dem `xs:sequence` - Element stehen!

## Attribute

- Das Element `xs:attribute` legt die *Attribute* eines Elements fest.
- Es kann nur innerhalb einer `xs:complexType` - Umgebung verwendet werden.
- `xs:attribute` hat folgende Attribute:
  - `use` kann den Wert "required" oder "optional" haben.
  - `type` legt den Datentyp fest. Mögliche Basistypen sind die selben wie bei den Elementen.
- Enthält `xs:complexType` ein `xs:sequence` - Element, **muss** das Element `xs:attribute` nach dem `xs:sequence` - Element stehen!

## Attribute

- Das Element `xs:attribute` legt die *Attribute* eines Elements fest.
- Es kann nur innerhalb einer `xs:complexType` - Umgebung verwendet werden.
- `xs:attribute` hat folgende Attribute:
  - `use` kann den Wert "required" oder "optional" haben.
  - `type` legt den Datentyp fest. Mögliche Basistypen sind die selben wie bei den Elementen.
- Enthält `xs:complexType` ein `xs:sequence` - Element, **muss** das Element `xs:attribute` nach dem `xs:sequence` - Element stehen!

## Attribute

- Das Element `xs:attribute` legt die *Attribute* eines Elements fest.
- Es kann nur innerhalb einer `xs:complexType` - Umgebung verwendet werden.
- `xs:attribute` hat folgende Attribute:
  - `use` kann den Wert "required" oder "optional" haben.
  - `type` legt den Datentyp fest. Mögliche Basistypen sind die selben wie bei den Elementen.
- Enthält `xs:complexType` ein `xs:sequence` - Element, **muss** das Element `xs:attribute` nach dem `xs:sequence` - Element stehen!

## Attribute

- Das Element `xs:attribute` legt die *Attribute* eines Elements fest.
- Es kann nur innerhalb einer `xs:complexType` - Umgebung verwendet werden.
- `xs:attribute` hat folgende Attribute:
  - `use` kann den Wert "required" oder "optional" haben.
  - `type` legt den Datentyp fest. Mögliche Basistypen sind die selben wie bei den Elementen.
- Enthält `xs:complexType` ein `xs:sequence` - Element, **muss** das Element `xs:attribute` nach dem `xs:sequence` - Element stehen!

## Einfaches Element mit Attribut

### Element mit Attribut ohne Kindselemente und ohne Textnoten

```
1 <xs:element name="timestamp">
2   <xs:complexType>
3     <xs:attribute name="utc" type="xs:long" />
4   </xs:complexType>
5 </xs:element>
```

Beispiel eines validen Elements:

```
<timestamp utc="123456789"/>
```

## Einfaches Element mit Attribut

### Element mit Attribut ohne Kindselemente und ohne Textnoten

```
1 | <xs:element name="timestamp">
2 |   <xs:complexType>
3 |     <xs:attribute name="utc" type="xs:long" />
4 |   </xs:complexType>
5 | </xs:element>
```

Beispiel eines validen Elements:

```
<timestamp utc="123456789"/>
```

## Einfaches Element mit Attribut

### Element mit Attribut ohne Kindselemente und ohne Textnoten

```
1 | <xs:element name="timestamp">
2 |   <xs:complexType>
3 |     <xs:attribute name="utc" type="xs:long"/>
4 |   </xs:complexType>
5 | </xs:element>
```

Beispiel eines validen Elements:

```
<timestamp utc="123456789"/>
```

## Einfaches Element mit Attribut

### Element mit Attribut ohne Kindselemente und ohne Textnoten

```
1 | <xs:element name="timestamp">
2 |   <xs:complexType>
3 |     <xs:attribute name="utc" type="xs:long"/>
4 |   </xs:complexType>
5 | </xs:element>
```

Beispiel eines validen Elements:

```
<timestamp utc="123456789"/>
```

## Einfaches Element mit Attribut

### Element mit Attribut ohne Kindselemente und ohne Textnoten

```
1 | <xs:element name="timestamp">
2 |   <xs:complexType>
3 |     <xs:attribute name="utc" type="xs:long" />
4 |   </xs:complexType>
5 | </xs:element>
```

Beispiel eines validen Elements:

```
<timestamp utc="123456789"/>
```

## Einfaches Element mit Attribut

### Element mit Attribut ohne Kindselemente und ohne Textnoten

```
1 | <xs:element name="timestamp">
2 |   <xs:complexType>
3 |     <xs:attribute name="utc" type="xs:long" />
4 |   </xs:complexType>
5 | </xs:element>
```

Beispiel eines validen Elements:

```
<timestamp utc="123456789"/>
```

## Einfaches Element mit Attribut

### Element mit Attribut ohne Kindselemente und ohne Textnoten

```
1 | <xs:element name="timestamp">
2 |   <xs:complexType>
3 |     <xs:attribute name="utc" type="xs:long" />
4 |   </xs:complexType>
5 | </xs:element>
```

Beispiel eines validen Elements:

```
<timestamp utc="123456789"/>
```

## Einfaches Element mit Attribut

### Element mit Attribut ohne Kindselemente und ohne Textnoten

```
1 | <xs:element name="timestamp">
2 |   <xs:complexType>
3 |     <xs:attribute name="utc" type="xs:long" />
4 |   </xs:complexType>
5 | </xs:element>
```

### Beispiel eines validen Elements:

```
<timestamp utc="123456789"/>
```

# Element mit Kindern und Attribut

## Element mit Attribut und Kindselementen

```
1 <xs:element name="Person" maxOccurs="unbounded">
2   <xs:complexType>
3     <xs:sequence>
4       <xs:element name="name" type="xs:string" />
5       <xs:element name="adresse" type="xs:string" minOccurs=
        "0" />
6     </xs:sequence>
7     <xs:attribute name="id" type="xs:long" use="required"
        />
8   </xs:complexType>
9 </xs:element>
```

Beispiel eines validen Elements:

```
<Person id="19700101">
  <name>neuman, alfred e.</name>
  <adresse>locaRoca 666, 1234 dancelingen</adresse>
</Person>
```

## Element mit Kindern und Attribut

### Element mit Attribut und Kindselementen

```
1 | <xs:element name="Person" maxOccurs="unbounded">
2 |   <xs:complexType>
3 |     <xs:sequence>
4 |       <xs:element name="name" type="xs:string" />
5 |       <xs:element name="adresse" type="xs:string" minOccurs=
      "0" />
6 |     </xs:sequence>
7 |     <xs:attribute name="id" type="xs:long" use="required"
      />
8 |   </xs:complexType>
9 | </xs:element>
```

### Beispiel eines validen Elements:

```
<Person id="19700101">
  <name>neuman, alfred e.</name>
  <adresse>locaRoca 666, 1234 dancelingen</adresse>
</Person>
```

# Element mit Kindern und Attribut

## Element mit Attribut und Kindselementen

```
1 | <xs:element name="Person" maxOccurs="unbounded">
2 |   <xs:complexType>
3 |     <xs:sequence>
4 |       <xs:element name="name" type="xs:string" />
5 |       <xs:element name="adresse" type="xs:string" minOccurs=
      "0" />
6 |     </xs:sequence>
7 |     <xs:attribute name="id" type="xs:long" use="required"
      />
8 |   </xs:complexType>
9 | </xs:element>
```

Beispiel eines validen Elements:

```
<Person id="19700101">
  <name>neuman, alfred e.</name>
  <adresse>locaRoca 666, 1234 dancelingen</adresse>
</Person>
```

## Element mit Kindern und Attribut

### Element mit Attribut und Kindselementen

```
1 <xs:element name="Person" maxOccurs="unbounded">
2   <xs:complexType>
3     <xs:sequence>
4       <xs:element name="name" type="xs:string" />
5       <xs:element name="adresse" type="xs:string" minOccurs=
          "0" />
6     </xs:sequence>
7     <xs:attribute name="id" type="xs:long" use="required"
          />
8   </xs:complexType>
9 </xs:element>
```

Beispiel eines validen Elements:

```
<Person id="19700101">
  <name>neuman, alfred e.</name>
  <adresse>locaRoca 666, 1234 dancelingen</adresse>
</Person>
```

# Element mit Kindern und Attribut

## Element mit Attribut und Kindselementen

```
1 <xs:element name="Person" maxOccurs="unbounded">
2   <xs:complexType>
3     <xs:sequence>
4       <xs:element name="name" type="xs:string" />
5       <xs:element name="adresse" type="xs:string" minOccurs=
          "0" />
6     </xs:sequence>
7     <xs:attribute name="id" type="xs:long" use="required"
          />
8   </xs:complexType>
9 </xs:element>
```

Beispiel eines validen Elements:

```
<Person id="19700101">
  <name>neuman, alfred e.</name>
  <adresse>locaRoca 666, 1234 dancelingen</adresse>
</Person>
```

## Element mit Kindern und Attribut

### Element mit Attribut und Kindselementen

```
1 <xs:element name="Person" maxOccurs="unbounded">
2   <xs:complexType>
3     <xs:sequence>
4       <xs:element name="name" type="xs:string" />
5       <xs:element name="adresse" type="xs:string" minOccurs=
          "0" />
6     </xs:sequence>
7     <xs:attribute name="id" type="xs:long" use="required"
          />
8   </xs:complexType>
9 </xs:element>
```

Beispiel eines validen Elements:

```
<Person id="19700101">
  <name>neuman, alfred e.</name>
  <adresse>locaRoca 666, 1234 dancelingen</adresse>
</Person>
```

# Element mit Kindern und Attribut

## Element mit Attribut und Kindselementen

```
1 <xs:element name="Person" maxOccurs="unbounded">
2   <xs:complexType>
3     <xs:sequence>
4       <xs:element name="name" type="xs:string" />
5       <xs:element name="adresse" type="xs:string" minOccurs=
          "0" />
6     </xs:sequence>
7     <xs:attribute name="id" type="xs:long" use="required"
          />
8   </xs:complexType>
9 </xs:element>
```

## Beispiel eines validen Elements:

```
<Person id="19700101">
  <name>neuman, alfred e.</name>
  <adresse>locaRoca 666, 1234 dancelingen</adresse>
</Person>
```

# Element mit Kindern und Attribut

## Element mit Attribut und Kindselementen

```
1 <xs:element name="Person" maxOccurs="unbounded">
2   <xs:complexType>
3     <xs:sequence>
4       <xs:element name="name" type="xs:string" />
5       <xs:element name="adresse" type="xs:string" minOccurs=
6         "0" />
7     </xs:sequence>
8     <xs:attribute name="id" type="xs:long" use="required"
9     />
10  </xs:complexType>
11 </xs:element>
```

## Beispiel eines validen Elements:

```
<Person id="19700101">
  <name>neuman, alfred e.</name>
  <adresse>locaRoca 666, 1234 dancelingen</adresse>
</Person>
```

# Element mit Kindern und Attribut

## Element mit Attribut und Kindselementen

```
1 <xs:element name="Person" maxOccurs="unbounded">
2   <xs:complexType>
3     <xs:sequence>
4       <xs:element name="name" type="xs:string" />
5       <xs:element name="adresse" type="xs:string" minOccurs=
          "0" />
6     </xs:sequence>
7     <xs:attribute name="id" type="xs:long" use="required"
          />
8   </xs:complexType>
9 </xs:element>
```

Beispiel eines validen Elements:

```
<Person id="19700101">
  <name>neuman, alfred e.</name>
  <adresse>locaRoca 666, 1234 dancelingen</adresse>
</Person>
```

# Element mit Kindern und Attribut

## Element mit Attribut und Kindselementen

```
1 <xs:element name="Person" maxOccurs="unbounded">
2   <xs:complexType>
3     <xs:sequence>
4       <xs:element name="name" type="xs:string" />
5       <xs:element name="adresse" type="xs:string" minOccurs=
          "0" />
6     </xs:sequence>
7     <xs:attribute name="id" type="xs:long" use="required"
          />
8   </xs:complexType>
9 </xs:element>
```

Beispiel eines validen Elements:

```
<Person id="19700101">
  <name>neuman, alfred e.</name>
  <adresse>locaRoca 666, 1234 dancelingen</adresse>
</Person>
```

# Element mit Kindern und Attribut

## Element mit Attribut und Kindselementen

```
1 <xs:element name="Person" maxOccurs="unbounded">
2   <xs:complexType>
3     <xs:sequence>
4       <xs:element name="name" type="xs:string" />
5       <xs:element name="adresse" type="xs:string" minOccurs=
          "0" />
6     </xs:sequence>
7     <xs:attribute name="id" type="xs:long" use="required"
          />
8   </xs:complexType>
9 </xs:element>
```

Beispiel eines validen Elements:

```
<Person id="19700101">
  <name>neuman, alfred e.</name>
  <adresse>locaRoca 666, 1234 dancelingen</adresse>
</Person>
```

## Element mit Kindern und Attribut

### Element mit Attribut und Kindselementen

```
1 <xs:element name="Person" maxOccurs="unbounded">
2   <xs:complexType>
3     <xs:sequence>
4       <xs:element name="name" type="xs:string" />
5       <xs:element name="adresse" type="xs:string" minOccurs=
6         "0" />
7     </xs:sequence>
8     <xs:attribute name="id" type="xs:long" use="required"
9     />
10  </xs:complexType>
11 </xs:element>
```

### Beispiel eines validen Elements:

```
<Person id="19700101">
  <name>neuman, alfred e.</name>
  <adresse>locaRoca 666, 1234 dancelingen</adresse>
</Person>
```

## Ergänzung zu Complex Types

### Element mit Attribut und Textknoten

```
1 <xs:element name="termin">
2   <xs:complexType>
3     <xs:simpleContent>
4       <xs:extension base="xs:string">
5         <xs:attribute name="utc" type="xs:long"/>
6       </xs:extension>
7     </xs:simpleContent>
8     <xs:attribute name="utc" type="xs:long"/>
9   </xs:complexType>
10 </xs:element>
```

Beispiel eines validen Elements:

```
<termin utc="123456789">klassenarbeit</termin>
```

## Ergänzung zu Complex Types

### Element mit Attribut und Textknoten

```
1 | <xs:element name="termin">
2 |   <xs:complexType>
3 |     <xs:simpleContent>
4 |       <xs:extension base="xs:string">
5 |         <xs:attribute name="utc" type="xs:long"/>
6 |       </xs:extension>
7 |     </xs:simpleContent>
8 |     <xs:attribute name="utc" type="xs:long"/>
9 |   </xs:complexType>
10| </xs:element>
```

Beispiel eines validen Elements:

```
<termin utc="123456789">klassenarbeit</termin>
```

## Ergänzung zu Complex Types

### Element mit Attribut und Textknoten

```
1 | <xs:element name="termin">
2 |   <xs:complexType>
3 |     <xs:simpleContent>
4 |       <xs:extension base="xs:string">
5 |         <xs:attribute name="utc" type="xs:long"/>
6 |       </xs:extension>
7 |     </xs:simpleContent>
8 |     <xs:attribute name="utc" type="xs:long"/>
9 |   </xs:complexType>
10| </xs:element>
```

Beispiel eines validen Elements:

```
<termin utc="123456789">klassenarbeit</termin>
```

## Ergänzung zu Complex Types

### Element mit Attribut und Textknoten

```
1 | <xs:element name="termin">
2 |   <xs:complexType>
3 |     <xs:simpleContent>
4 |       <xs:extension base="xs:string">
5 |         <xs:attribute name="utc" type="xs:long"/>
6 |       </xs:extension>
7 |     </xs:simpleContent>
8 |     <xs:attribute name="utc" type="xs:long"/>
9 |   </xs:complexType>
10| </xs:element>
```

Beispiel eines validen Elements:

```
<termin utc="123456789">klassenarbeit</termin>
```

## Ergänzung zu Complex Types

### Element mit Attribut und Textknoten

```
1 | <xs:element name="termin">
2 |   <xs:complexType>
3 |     <xs:simpleContent>
4 |       <xs:extension base="xs:string">
5 |         <xs:attribute name="utc" type="xs:long"/>
6 |       </xs:extension>
7 |     </xs:simpleContent>
8 |     <xs:attribute name="utc" type="xs:long"/>
9 |   </xs:complexType>
10| </xs:element>
```

Beispiel eines validen Elements:

```
<termin utc="123456789">klassenarbeit</termin>
```

## Ergänzung zu Complex Types

### Element mit Attribut und Textknoten

```
1 | <xs:element name="termin">
2 |   <xs:complexType>
3 |     <xs:simpleContent>
4 |       <xs:extension base="xs:string">
5 |         <xs:attribute name="utc" type="xs:long"/>
6 |       </xs:extension>
7 |     </xs:simpleContent>
8 |     <xs:attribute name="utc" type="xs:long"/>
9 |   </xs:complexType>
10| </xs:element>
```

Beispiel eines validen Elements:

```
<termin utc="123456789">klassenarbeit</termin>
```

## Ergänzung zu Complex Types

### Element mit Attribut und Textknoten

```
1 <xs:element name="termin">
2   <xs:complexType>
3     <xs:simpleContent>
4       <xs:extension base="xs:string">
5         <xs:attribute name="utc" type="xs:long"/>
6       </xs:extension>
7     </xs:simpleContent>
8     <xs:attribute name="utc" type="xs:long"/>
9   </xs:complexType>
10 </xs:element>
```

Beispiel eines validen Elements:

```
<termin utc="123456789">klassenarbeit</termin>
```

## Ergänzung zu Complex Types

### Element mit Attribut und Textknoten

```
1 | <xs:element name="termin">
2 |   <xs:complexType>
3 |     <xs:simpleContent>
4 |       <xs:extension base="xs:string">
5 |         <xs:attribute name="utc" type="xs:long"/>
6 |       </xs:extension>
7 |     </xs:simpleContent>
8 |     <xs:attribute name="utc" type="xs:long"/>
9 |   </xs:complexType>
10| </xs:element>
```

Beispiel eines validen Elements:

```
<termin utc="123456789">klassenarbeit</termin>
```

## Ergänzung zu Complex Types

### Element mit Attribut und Textknoten

```
1 | <xs:element name="termin">
2 |   <xs:complexType>
3 |     <xs:simpleContent>
4 |       <xs:extension base="xs:string">
5 |         <xs:attribute name="utc" type="xs:long"/>
6 |       </xs:extension>
7 |     </xs:simpleContent>
8 |     <xs:attribute name="utc" type="xs:long"/>
9 |   </xs:complexType>
10| </xs:element>
```

Beispiel eines validen Elements:

```
<termin utc="123456789">klassenarbeit</termin>
```

## Ergänzung zu Complex Types

### Element mit Attribut und Textknoten

```
1 | <xs:element name="termin">
2 |   <xs:complexType>
3 |     <xs:simpleContent>
4 |       <xs:extension base="xs:string">
5 |         <xs:attribute name="utc" type="xs:long"/>
6 |       </xs:extension>
7 |     </xs:simpleContent>
8 |     <xs:attribute name="utc" type="xs:long"/>
9 |   </xs:complexType>
10| </xs:element>
```

Beispiel eines validen Elements:

```
<termin utc="123456789">klassenarbeit</termin>
```

## Ergänzung zu Complex Types

### Element mit Attribut und Textknoten

```
1 | <xs:element name="termin">
2 |   <xs:complexType>
3 |     <xs:simpleContent>
4 |       <xs:extension base="xs:string">
5 |         <xs:attribute name="utc" type="xs:long"/>
6 |       </xs:extension>
7 |     </xs:simpleContent>
8 |     <xs:attribute name="utc" type="xs:long"/>
9 |   </xs:complexType>
10| </xs:element>
```

Beispiel eines validen Elements:

```
<termin utc="123456789">klassenarbeit</termin>
```

## Ergänzung zu Complex Types

### Element mit Attribut und Textknoten

```
1 | <xs:element name="termin">
2 |   <xs:complexType>
3 |     <xs:simpleContent>
4 |       <xs:extension base="xs:string">
5 |         <xs:attribute name="utc" type="xs:long"/>
6 |       </xs:extension>
7 |     </xs:simpleContent>
8 |     <xs:attribute name="utc" type="xs:long"/>
9 |   </xs:complexType>
10| </xs:element>
```

Beispiel eines validen Elements:

```
<termin utc="123456789">klassenarbeit</termin>
```

## Ergänzung zu Complex Types

### Element mit Attribut und Textknoten

```
1 | <xs:element name="termin">
2 |   <xs:complexType>
3 |     <xs:simpleContent>
4 |       <xs:extension base="xs:string">
5 |         <xs:attribute name="utc" type="xs:long"/>
6 |       </xs:extension>
7 |     </xs:simpleContent>
8 |     <xs:attribute name="utc" type="xs:long"/>
9 |   </xs:complexType>
10| </xs:element>
```

### Beispiel eines validen Elements:

```
<termin utc="123456789">klassenarbeit</termin>
```

## Ein komplettes Beispiel

- Die XSD-Datei beschreibt eine xml-Datei, die beliebig viele (`maxOccurs="unbounded"`) `<Person>`-Elemente enthalten darf.
- Jedes `<Person>`-Element muss ein Attribut **id** haben.
- Jedes `<Person>`-Element enthält **genau ein** (weder `minOccurs` noch `maxOccurs` vorhanden) Element `<name>`.
- Jedes `<Person>`-Element enthält **optional** (`minOccurs=0`, `maxOccurs` nicht vorhanden, d.h. `maxOccurs="1"`) ein Element `<adresse>`.

## Ein komplettes Beispiel

- Die XSD-Datei beschreibt eine xml-Datei, die beliebig viele (`maxOccurs="unbounded"`) `<Person>`-Elemente enthalten darf.
- Jedes `<Person>`-Element muss ein Attribut `id` haben.
- Jedes `<Person>`-Element enthält **genau ein** (weder `minOccurs` noch `maxOccurs` vorhanden) Element `<name>`.
- Jedes `<Person>`-Element enthält **optional** (`minOccurs=0`, `maxOccurs` nicht vorhanden, d.h. `maxOccurs="1"`) ein Element `<adresse>`.

## Ein komplettes Beispiel

- Die XSD-Datei beschreibt eine xml-Datei, die beliebig viele (`maxOccurs="unbounded"`) `<Person>`-Elemente enthalten darf.
- Jedes `<Person>`-Element muss ein Attribut **id** haben.
- Jedes `<Person>`-Element enthält **genau ein** (weder `minOccurs` noch `maxOccurs` vorhanden) Element `<name>`.
- Jedes `<Person>`-Element enthält **optional** (`minOccurs=0`, `maxOccurs` nicht vorhanden, d.h. `maxOccurs="1"`) ein Element `<adresse>`.

## Ein komplettes Beispiel

- Die XSD-Datei beschreibt eine xml-Datei, die beliebig viele (`maxOccurs="unbounded"`) `<Person>`-Elemente enthalten darf.
- Jedes `<Person>`-Element muss ein Attribut **id** haben.
- Jedes `<Person>`-Element enthält **genau ein** (weder `minOccurs` noch `maxOccurs` vorhanden) Element `<name>`.
- Jedes `<Person>`-Element enthält **optional** (`minOccurs=0`, `maxOccurs` nicht vorhanden, d.h. `maxOccurs="1"`) ein Element `<adresse>`.

## Ein komplettes Beispiel

- Die XSD-Datei beschreibt eine xml-Datei, die beliebig viele (`maxOccurs="unbounded"`) `<Person>`-Elemente enthalten darf.
- Jedes `<Person>`-Element muss ein Attribut **id** haben.
- Jedes `<Person>`-Element enthält **genau ein** (weder `minOccurs` noch `maxOccurs` vorhanden) Element `<name>`.
- Jedes `<Person>`-Element enthält **optional** (`minOccurs=0`, `maxOccurs` nicht vorhanden, d.h. `maxOccurs="1"`) ein Element `<adresse>`.

## Zum Schema passende XML-Datei

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<kontakte>
  <Person id="19631223">
    <name>dienert, michael</name>
    <adresse>waelderstrasse 7, 79341 kenzingen</adresse>
  </Person>
  <Person id="19700101">
    <name>neuman, alfred e.</name>
    <adresse>locaRoca 666, 1234 dancelingen</adresse>
  </Person>
  <Person id="20000210">
    <name>bosak, jon</name>
    <adresse>
      4150 Network Cir, Santa Clara, CA USA
    </adresse>
  </Person>
</kontakte>
```

# Ergänzungen zu Simple Types

## Simple Types anpassen

- Simple Types können durch Einschränkungen *angepasst* werden.
- Die Definitionen dieser Einschränkungen werden *Facets* genannt.

## Ergänzungen zu Simple Types

### Simple Types anpassen

- Simple Types können durch Einschränkungen *angepasst* werden.
- Die Definitionen dieser Einschränkungen werden *Facets* genannt.

## Ergänzungen zu Simple Types

### Simple Types anpassen

- Simple Types können durch Einschränkungen *angepasst* werden.
- Die Definitionen dieser Einschränkungen werden *Facets* genannt.

## Ergänzungen zu Simple Types

### Simple Types anpassen

- Simple Types können durch Einschränkungen *angepasst* werden.
- Die Definitionen dieser Einschränkungen werden *Facets* genannt.

# Beispiele zu Facets

## Aufzählungen - Enumerations

```
1 <xs:simpleType name="wochentag">
2   <xs:restriction base="xs:string">
3     <xs:enumeration value="Montag" />
4     <xs:enumeration value="Dienstag" />
5     <xs:enumeration value="Mittwoch" />
6     ...
7   </xs:restriction>
8 </xs:simpleType>
```

## Beispiele zu Facets

### Aufzählungen - Enumerations

```
1 <xs:simpleType name="wochentag">
2   <xs:restriction base="xs:string">
3     <xs:enumeration value="Montag" />
4     <xs:enumeration value="Dienstag" />
5     <xs:enumeration value="Mittwoch" />
6     ...
7   </xs:restriction>
8 </xs:simpleType>
```

## Beispiele zu Facets

### Aufzählungen - Enumerations

```
1 | <xs:simpleType name="wochentag">
2 |   <xs:restriction base="xs:string">
3 |     <xs:enumeration value="Montag" />
4 |     <xs:enumeration value="Dienstag" />
5 |     <xs:enumeration value="Mittwoch" />
6 |     ...
7 |   </xs:restriction>
8 | </xs:simpleType>
```

## Beispiele zu Facets

### Aufzählungen - Enumerations

```
1 <xs:simpleType name="wochentag">
2   <xs:restriction base="xs:string">
3     <xs:enumeration value="Montag" />
4     <xs:enumeration value="Dienstag" />
5     <xs:enumeration value="Mittwoch" />
6     ...
7   </xs:restriction>
8 </xs:simpleType>
```

## Beispiele zu Facets

### Aufzählungen - Enumerations

```
1 <xs:simpleType name="wochentag">
2   <xs:restriction base="xs:string">
3     <xs:enumeration value="Montag" />
4     <xs:enumeration value="Dienstag" />
5     <xs:enumeration value="Mittwoch" />
6     ...
7   </xs:restriction>
8 </xs:simpleType>
```

## Beispiele zu Facets

### Aufzählungen - Enumerations

```
1 <xs:simpleType name="wochentag">
2   <xs:restriction base="xs:string">
3     <xs:enumeration value="Montag" />
4     <xs:enumeration value="Dienstag" />
5     <xs:enumeration value="Mittwoch" />
6     ...
7   </xs:restriction>
8 </xs:simpleType>
```

## Beispiele zu Facets

### Aufzählungen - Enumerations

```
1 <xs:simpleType name="wochentag">
2   <xs:restriction base="xs:string">
3     <xs:enumeration value="Montag" />
4     <xs:enumeration value="Dienstag" />
5     <xs:enumeration value="Mittwoch" />
6     ...
7   </xs:restriction>
8 </xs:simpleType>
```

## Beispiele zu Facets

### Aufzählungen - Enumerations

```
1 <xs:simpleType name="wochentag">
2   <xs:restriction base="xs:string">
3     <xs:enumeration value="Montag" />
4     <xs:enumeration value="Dienstag" />
5     <xs:enumeration value="Mittwoch" />
6     ...
7   </xs:restriction>
8 </xs:simpleType>
```

## Beispiele zu Facets

### Aufzählungen - Enumerations

```
1 <xs:simpleType name="wochentag">
2   <xs:restriction base="xs:string">
3     <xs:enumeration value="Montag" />
4     <xs:enumeration value="Dienstag" />
5     <xs:enumeration value="Mittwoch" />
6     ...
7   </xs:restriction>
8 </xs:simpleType>
```

## Beispiele zu Facets

### Aufzählungen - Enumerations

```
1 <xs:simpleType name="wochentag">
2   <xs:restriction base="xs:string">
3     <xs:enumeration value="Montag" />
4     <xs:enumeration value="Dienstag" />
5     <xs:enumeration value="Mittwoch" />
6     ...
7   </xs:restriction>
8 </xs:simpleType>
```

## Beispiele zu Facets

### Anzahl Zeichen in einem String

```
1 <xs:simpleType name="password">
2   <xs:restriction base="xs:string">
3     <xs:minLength value="6" />
4     <xs:maxLength value="8" />
5   </xs:restriction>
6 </xs:simpleType>
```

## Beispiele zu Facets

### Anzahl Zeichen in einem String

```
1 <xs:simpleType name="password">  
2   <xs:restriction base="xs:string">  
3     <xs:minLength value="6" />  
4     <xs:maxLength value="8" />  
5   </xs:restriction>  
6 </xs:simpleType>
```

## Beispiele zu Facets

### Anzahl Zeichen in einem String

```
1 | <xs:simpleType name="password">
2 |   <xs:restriction base="xs:string">
3 |     <xs:minLength value="6" />
4 |     <xs:maxLength value="8" />
5 |   </xs:restriction>
6 | </xs:simpleType>
```

## Beispiele zu Facets

### Anzahl Zeichen in einem String

```
1 | <xs:simpleType name="password">  
2 |   <xs:restriction base="xs:string">  
3 |     <xs:minLength value="6" />  
4 |     <xs:maxLength value="8" />  
5 |   </xs:restriction>  
6 | </xs:simpleType>
```

## Beispiele zu Facets

### Anzahl Zeichen in einem String

```
1 <xs:simpleType name="password">  
2   <xs:restriction base="xs:string">  
3     <xs:minLength value="6" />  
4     <xs:maxLength value="8" />  
5   </xs:restriction>  
6 </xs:simpleType>
```

## Beispiele zu Facets

### Anzahl Zeichen in einem String

```
1 <xs:simpleType name="password">  
2   <xs:restriction base="xs:string">  
3     <xs:minLength value="6" />  
4     <xs:maxLength value="8" />  
5   </xs:restriction>  
6 </xs:simpleType>
```

## Beispiele zu Facets

### Anzahl Zeichen in einem String

```
1 | <xs:simpleType name="password">
2 |   <xs:restriction base="xs:string">
3 |     <xs:minLength value="6" />
4 |     <xs:maxLength value="8" />
5 |   </xs:restriction>
6 | </xs:simpleType>
```

## Beispiele zu Facets

### Anzahl Zeichen in einem String

```
1 | <xs:simpleType name="password">  
2 |   <xs:restriction base="xs:string">  
3 |     <xs:minLength value="6" />  
4 |     <xs:maxLength value="8" />  
5 |   </xs:restriction>  
6 | </xs:simpleType>
```

## Beispiele zu Facets

### Anzahl Zeichen in einem String

```
1 | <xs:simpleType name="password">  
2 |   <xs:restriction base="xs:string">  
3 |     <xs:minLength value="6" />  
4 |     <xs:maxLength value="8" />  
5 |   </xs:restriction>  
6 | </xs:simpleType>
```

# Beispiele zu Facets

## Ein Facet für E-Mail-Adressen

```
1 <xsd:simpleType name="emailAddress">
2   <xsd:restriction base="xsd:string">
3     <xsd:pattern value="\w+([-+.']\w+)*@\w+([-.]
4       +([-.]
5     </xsd:restriction>
  </xsd:simpleType>
```

## Beispiele zu Facets

### Ein Facet für E-Mail-Adressen

```
1 <xsd:simpleType name="emailAddress">
2   <xsd:restriction base="xsd:string">
3     <xsd:pattern value="\w+([-+.']\w+)*@\w+([-.] \w+)*\.\w
      +([-.] \w+)*" />
4   </xsd:restriction>
5 </xsd:simpleType>
```

## Beispiele zu Facets

### Ein Facet für E-Mail-Adressen

```
1 | <xsd:simpleType name="emailAddress">
2 |   <xsd:restriction base="xsd:string">
3 |     <xsd:pattern value="\w+([-+.']\w+)*@\w+([-.] \w+)*\.\w
      +([-.] \w+)*" />
4 |   </xsd:restriction>
5 | </xsd:simpleType>
```

## Beispiele zu Facets

### Ein Facet für E-Mail-Adressen

```
1 | <xsd:simpleType name="emailAddress">
2 |   <xsd:restriction base="xsd:string">
3 |     <xsd:pattern value="\w+([-+.']\w+)*@\w+([-.] \w+)*\.\w
      +([-.] \w+)*" />
4 |   </xsd:restriction>
5 | </xsd:simpleType>
```

## Beispiele zu Facets

### Ein Facet für E-Mail-Adressen

```
1 | <xsd:simpleType name="emailAddress">
2 |   <xsd:restriction base="xsd:string">
3 |     <xsd:pattern value="\w+([-+.']\w+)*@\w+([-.] \w+)*\.\w
      +([-.] \w+)*" />
4 |   </xsd:restriction>
5 | </xsd:simpleType>
```

## Beispiele zu Facets

### Ein Facet für E-Mail-Adressen

```
1 | <xsd:simpleType name="emailAddress">
2 |   <xsd:restriction base="xsd:string">
3 |     <xsd:pattern value="\w+([-+.']\w+)*@\w+([-.] \w+)*\.\w
      +([-.] \w+)*" />
4 |   </xsd:restriction>
5 | </xsd:simpleType>
```

## Beispiele zu Facets

### Ein Facet für E-Mail-Adressen

```
1 | <xsd:simpleType name="emailAddress">
2 |   <xsd:restriction base="xsd:string">
3 |     <xsd:pattern value="\w+([-+.']\w+)*@\w+([-.] \w+)*\.\w
      +([-.] \w+)*" />
4 |   </xsd:restriction>
5 | </xsd:simpleType>
```

## Beispiele zu Facets

### Ein Facet für E-Mail-Adressen

```
1 | <xsd:simpleType name="emailAddress">
2 |   <xsd:restriction base="xsd:string">
3 |     <xsd:pattern value="\w+([-+.']\w+)*@\w+([-.] \w+)*\.\w
   |       +([-.] \w+)*" />
4 |   </xsd:restriction>
5 | </xsd:simpleType>
```

## Generierung von JAVA-Code

Java-Code generieren mit dem Kommando `xjc`:

```
xjc kontakte.xsd
```

Direktes Erzeugen der Quellen innerhalb eines NB-Projekts:

```
File-Menue → New File → Kategorie: XML → Type: JAXB Binding
```

## Generierung von JAVA-Code

Java-Code generieren mit dem Kommando `xjc`:

```
xjc kontakte.xsd
```

Direktes Erzeugen der Quellen innerhalb eines NB-Projekts:

```
File-Menue → New File → Kategorie: XML → Type: JAXB Binding
```

## Generierung von JAVA-Code

Java-Code generieren mit dem Kommando `xjc`:

```
xjc kontakte.xsd
```

Direktes Erzeugen der Quellen innerhalb eines NB-Projekts:

```
File-Menue → New File → Kategorie: XML → Type: JAXB Binding
```

## Generierung von JAVA-Code

Java-Code generieren mit dem Kommando `xjc`:

```
xjc kontakte.xsd
```

Direktes Erzeugen der Quellen innerhalb eines NB-Projekts:

```
File-Menue → New File → Kategorie: XML → Type: JAXB Binding
```

## Generierung von JAVA-Code

Java-Code generieren mit dem Kommando `xjc`:

```
xjc kontakte.xsd
```

Direktes Erzeugen der Quellen innerhalb eines NB-Projekts:

```
File-Menue → New File → Kategorie: XML → Type: JAXB Binding
```

## Generierung von JAVA-Code

- JAXB erzeugt eine Klasse, mit dem Namen des Wurzelements: `Kontakte.java`
- Die vielen `<Personen>`-Elemente bildet JAXB auf eine `ArrayList` ab, die Objekte vom Typ `Kontakte.Person` enthalten darf. d.h. `Person` ist innere Klasse von `Kontakte`.
- Die innere Klasse **Person** wiederum ist nichts anderes wie eine **JavaBean** mit den Eigenschaften:
  - `String name;`
  - `String adresse;`
  - `long id;`

## Generierung von JAVA-Code

- JAXB erzeugt eine Klasse, mit dem Namen des Wurzelements: `Kontakte.java`
- Die vielen `<Personen>`-Elemente bildet JAXB auf eine `ArrayList` ab, die Objekte vom Typ `Kontakte.Person` enthalten darf. d.h. `Person` ist innere Klasse von `Kontakte`.
- Die innere Klasse **Person** wiederum ist nichts anderes wie eine **JavaBean** mit den Eigenschaften:
  - `String name;`
  - `String adresse;`
  - `long id;`

## Generierung von JAVA-Code

- JAXB erzeugt eine Klasse, mit dem Namen des Wurzelements: `Kontakte.java`
- Die vielen `<Personen>`-Elemente bildet JAXB auf eine `ArrayList` ab, die Objekte vom Typ `Kontakte.Person` enthalten darf. d.h. `Person` ist innere Klasse von `Kontakte`.
- Die innere Klasse **Person** wiederum ist nichts anderes wie eine **JavaBean** mit den Eigenschaften:
  - `String name;`
  - `String adresse;`
  - `long id;`

## Generierung von JAVA-Code

- JAXB erzeugt eine Klasse, mit dem Namen des Wurzelements: `Kontakte.java`
- Die vielen `<Personen>`-Elemente bildet JAXB auf eine `ArrayList` ab, die Objekte vom Typ `Kontakte.Person` enthalten darf. d.h. `Person` ist innere Klasse von `Kontakte`.
- Die innere Klasse **Person** wiederum ist nichts anderes wie eine **JavaBean** mit den Eigenschaften:
  - `String name;`
  - `String adresse;`
  - `long id;`

## Generierung von JAVA-Code

- JAXB erzeugt eine Klasse, mit dem Namen des Wurzelements: `Kontakte.java`
- Die vielen `<Personen>`-Elemente bildet JAXB auf eine `ArrayList` ab, die Objekte vom Typ `Kontakte.Person` enthalten darf. d.h. `Person` ist innere Klasse von `Kontakte`.
- Die innere Klasse **Person** wiederum ist nichts anderes wie eine **JavaBean** mit den Eigenschaften:
  - `String name;`
  - `String adresse;`
  - `long id;`

## Generierung von JAVA-Code

- JAXB erzeugt eine Klasse, mit dem Namen des Wurzelements: `Kontakte.java`
- Die vielen `<Personen>`-Elemente bildet JAXB auf eine `ArrayList` ab, die Objekte vom Typ `Kontakte.Person` enthalten darf. d.h. `Person` ist innere Klasse von `Kontakte`.
- Die innere Klasse **Person** wiederum ist nichts anderes wie eine **JavaBean** mit den Eigenschaften:
  - `String name;`
  - `String adresse;`
  - `long id;`

## Generierung von JAVA-Code

- JAXB erzeugt eine Klasse, mit dem Namen des Wurzelements: `Kontakte.java`
- Die vielen `<Personen>`-Elemente bildet JAXB auf eine `ArrayList` ab, die Objekte vom Typ `Kontakte.Person` enthalten darf. d.h. `Person` ist innere Klasse von `Kontakte`.
- Die innere Klasse **Person** wiederum ist nichts anderes wie eine **JavaBean** mit den Eigenschaften:
  - `String name;`
  - `String adresse;`
  - `long id;`

## Generierung von JAVA-Code

- JAXB erzeugt eine Klasse, mit dem Namen des Wurzelements: `Kontakte.java`
- Die vielen `<Personen>`-Elemente bildet JAXB auf eine `ArrayList` ab, die Objekte vom Typ `Kontakte.Person` enthalten darf. d.h. `Person` ist innere Klasse von `Kontakte`.
- Die innere Klasse **Person** wiederum ist nichts anderes wie eine **JavaBean** mit den Eigenschaften:
  - `String name;`
  - `String adresse;`
  - `long id;`

## Generierung von JAVA-Code

- JAXB erzeugt eine Klasse, mit dem Namen des Wurzelements: `Kontakte.java`
- Die vielen `<Personen>`-Elemente bildet JAXB auf eine `ArrayList` ab, die Objekte vom Typ `Kontakte.Person` enthalten darf. d.h. `Person` ist innere Klasse von `Kontakte`.
- Die innere Klasse **Person** wiederum ist nichts anderes wie eine **JavaBean** mit den Eigenschaften:
  - `String name;`
  - `String adresse;`
  - `long id;`

## Marshalling: Java-Ojekt nach XML

- Marshalling: ähnlich Serialisierung von Objekten; Originalbedeutung: Einweisen eines Flugzeugs auf dem Flugfeld zur Startbahn durch den Marshall.
- Code-Schnipsel:

```
1 // create JAXB context and instantiate marshaller
2 JAXBContext context = JAXBContext.newInstance(Kontakte.
   class);
3 Marshaller m = context.createMarshaller();
4 m.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, Boolean.
   TRUE);
5
6 // Write to System.out
7 m.marshal(kontakte, System.out);
8
9 // Write to File
10 m.marshal(kontakte, new File("/tmp/kontakte.xml"));
```

## Marshalling: Java-Ojekt nach XML

- **Marshalling:** ähnlich Serialisierung von Objekten;  
Originalbedeutung: Einweisen eines Flugzeugs auf dem Flugfeld zur Startbahn durch den Marshall.
- Code-Schnipsel:

```
1 // create JAXB context and instantiate marshaller
2 JAXBContext context = JAXBContext.newInstance(Kontakte.
   class);
3 Marshaller m = context.createMarshaller();
4 m.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, Boolean.
   TRUE);
5
6 // Write to System.out
7 m.marshal(kontakte, System.out);
8
9 // Write to File
10 m.marshal(kontakte, new File("/tmp/kontakte.xml"));
```

## Marshalling: Java-Ojekt nach XML

- Marshalling: ähnlich Serialisierung von Objekten; Originalbedeutung: Einweisen eines Flugzeugs auf dem Flugfeld zur Startbahn durch den Marshall.
- Code-Schnipsel:

```
1 // create JAXB context and instantiate marshaller
2 JAXBContext context = JAXBContext.newInstance(Kontakte.
   class);
3 Marshaller m = context.createMarshaller();
4 m.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, Boolean.
   TRUE);
5
6 // Write to System.out
7 m.marshal(kontakte, System.out);
8
9 // Write to File
10 m.marshal(kontakte, new File("/tmp/kontakte.xml"));
```

## Marshalling: Java-Objekt nach XML

- Marshalling: ähnlich Serialisierung von Objekten; Originalbedeutung: Einweisen eines Flugzeugs auf dem Flugfeld zur Startbahn durch den Marshall.
- Code-Schnipsel:

```
1 // create JAXB context and instantiate marshaller
2 JAXBContext context = JAXBContext.newInstance(Kontakte.
   class);
3 Marshaller m = context.createMarshaller();
4 m.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, Boolean.
   TRUE);
5
6 // Write to System.out
7 m.marshal(kontakte, System.out);
8
9 // Write to File
10 m.marshal(kontakte, new File("/tmp/kontakte.xml"));
```

## Marshalling: Java-Objekt nach XML

- Marshalling: ähnlich Serialisierung von Objekten; Originalbedeutung: Einweisen eines Flugzeugs auf dem Flugfeld zur Startbahn durch den Marshall.
- Code-Schnipsel:

```
1 // create JAXB context and instantiate marshaller
2 JAXBContext context = JAXBContext.newInstance(Kontakte.
   class);
3 Marshaller m = context.createMarshaller();
4 m.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, Boolean.
   TRUE);
5
6 // Write to System.out
7 m.marshal(kontakte, System.out);
8
9 // Write to File
10 m.marshal(kontakte, new File("/tmp/kontakte.xml"));
```

## Marshalling: Java-Objekt nach XML

- Marshalling: ähnlich Serialisierung von Objekten; Originalbedeutung: Einweisen eines Flugzeugs auf dem Flugfeld zur Startbahn durch den Marshall.
- Code-Schnipsel:

```
1 // create JAXB context and instantiate marshaller
2 JAXBContext context = JAXBContext.newInstance(Kontakte.
   class);
3 Marshaller m = context.createMarshaller();
4 m.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, Boolean.
   TRUE);
5
6 // Write to System.out
7 m.marshal(kontakte, System.out);
8
9 // Write to File
10 m.marshal(kontakte, new File("/tmp/kontakte.xml"));
```

## Marshalling: Java-Objekt nach XML

- Marshalling: ähnlich Serialisierung von Objekten; Originalbedeutung: Einweisen eines Flugzeugs auf dem Flugfeld zur Startbahn durch den Marshall.
- Code-Schnipsel:

```
1 // create JAXB context and instantiate marshaller
2 JAXBContext context = JAXBContext.newInstance(Kontakte.
   class);
3 Marshaller m = context.createMarshaller();
4 m.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, Boolean.
   TRUE);
5
6 // Write to System.out
7 m.marshal(kontakte, System.out);
8
9 // Write to File
10 m.marshal(kontakte, new File("/tmp/kontakte.xml"));
```

## Marshalling: Java-Objekt nach XML

- Marshalling: ähnlich Serialisierung von Objekten; Originalbedeutung: Einweisen eines Flugzeugs auf dem Flugfeld zur Startbahn durch den Marshall.
- Code-Schnipsel:

```
1 // create JAXB context and instantiate marshaller
2 JAXBContext context = JAXBContext.newInstance(Kontakte.
   class);
3 Marshaller m = context.createMarshaller();
4 m.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, Boolean.
   TRUE);
5
6 // Write to System.out
7 m.marshal(kontakte, System.out);
8
9 // Write to File
10 m.marshal(kontakte, new File("/tmp/kontakte.xml"));
```

## Unmarshalling: XML-Dokument nach Java-Objekt

- Unmarshalling: Erzeugung von Objekten der mit `xjc` generierten Klassen. Hier: `Kontakte` und `Kontakte.Person`
- Code Schnipsel:

```
1 private List<Kontakte.Person> personListe;  
2  
3 JAXBContext context = JAXBContext.newInstance(Kontakte.  
4     class);  
5 Unmarshaller u = context.createUnmarshaller();  
6 Kontakte kontakteNeu = (Kontakte) u.unmarshal(  
7     new File("/tmp/andereKontakte.xml"));  
8  
9     personListe = kontakteNeu.getPerson();
```

## Unmarshalling: XML-Dokument nach Java-Objekt

- Unmarshalling: Erzeugung von Objekten der mit `xjc` generierten Klassen. Hier: `Kontakte` und `Kontakte.Person`
- Code Schnipsel:

```
1 private List<Kontakte.Person> personListe;  
2  
3 JAXBContext context = JAXBContext.newInstance(Kontakte.  
4     class);  
5 Unmarshaller u = context.createUnmarshaller();  
6 Kontakte kontakteNeu = (Kontakte) u.unmarshal(  
7     new File("/tmp/andereKontakte.xml"));  
8  
9     personListe = kontakteNeu.getPerson();
```

## Unmarshalling: XML-Dokument nach Java-Objekt

- Unmarshalling: Erzeugung von Objekten der mit `xjc` generierten Klassen. Hier: `Kontakte` und `Kontakte.Person`
- Code Schnipsel:

```
1 private List<Kontakte.Person> personListe;  
2  
3 JAXBContext context = JAXBContext.newInstance(Kontakte.  
4     class);  
5 Unmarshaller u = context.createUnmarshaller();  
6 Kontakte kontakteNeu = (Kontakte) u.unmarshal(  
7     new File("/tmp/andereKontakte.xml"));  
8  
9     personListe = kontakteNeu.getPerson();
```

## Unmarshalling: XML-Dokument nach Java-Objekt

- Unmarshalling: Erzeugung von Objekten der mit `xjc` generierten Klassen. Hier: `Kontakte` und `Kontakte.Person`
- Code Schnipsel:

```
1 private List<Kontakte.Person> personListe;  
2  
3 JAXBContext context = JAXBContext.newInstance(Kontakte.  
4     class);  
5 Unmarshaller u = context.createUnmarshaller();  
6 Kontakte kontakteNeu = (Kontakte) u.unmarshal(  
7     new File("/tmp/andereKontakte.xml"));  
8  
9     personListe = kontakteNeu.getPerson();
```

## Unmarshalling: XML-Dokument nach Java-Objekt

- Unmarshalling: Erzeugung von Objekten der mit `xjc` generierten Klassen. Hier: `Kontakte` und `Kontakte.Person`
- Code Schnipsel:

```
1 | private List<Kontakte.Person> personListe ;
2 |
3 | JAXBContext context = JAXBContext.newInstance(Kontakte.
4 |     class) ;
5 | Unmarshaller u = context.createUnmarshaller() ;
6 | Kontakte kontakteNeu = (Kontakte) u.unmarshal(
7 |     new File("/tmp/andereKontakte.xml")) ;
8 |
9 |     personListe = kontakteNeu.getPerson() ;
```

## Unmarshalling: XML-Dokument nach Java-Objekt

- Unmarshalling: Erzeugung von Objekten der mit `xjc` generierten Klassen. Hier: `Kontakte` und `Kontakte.Person`
- Code Schnipsel:

```
1 | private List<Kontakte.Person> personListe;  
2 |  
3 | JAXBContext context = JAXBContext.newInstance(Kontakte.  
   |     class);  
4 | Unmarshaller u = context.createUnmarshaller();  
5 |  
6 | Kontakte kontakteNeu = (Kontakte) u.unmarshal(  
   |     new File("/tmp/andereKontakte.xml"));  
7 |  
8 |  
9 |     personListe = kontakteNeu.getPerson();
```

## Unmarshalling: XML-Dokument nach Java-Objekt

- Unmarshalling: Erzeugung von Objekten der mit `xjc` generierten Klassen. Hier: `Kontakte` und `Kontakte.Person`
- Code Schnipsel:

```
1 private List<Kontakte.Person> personListe;  
2  
3 JAXBContext context = JAXBContext.newInstance(Kontakte.  
4     class);  
5  
6 Unmarshaller u = context.createUnmarshaller();  
7  
8 Kontakte kontakteNeu = (Kontakte) u.unmarshal(  
9     new File("/tmp/andereKontakte.xml"));  
10  
11 personListe = kontakteNeu.getPerson();
```

## Unmarshalling: XML-Dokument nach Java-Objekt

- Unmarshalling: Erzeugung von Objekten der mit `xjc` generierten Klassen. Hier: `Kontakte` und `Kontakte.Person`
- Code Schnipsel:

```
1 private List<Kontakte.Person> personListe;  
2  
3 JAXBContext context = JAXBContext.newInstance(Kontakte.  
4     class);  
5 Unmarshaller u = context.createUnmarshaller();  
6 Kontakte kontakteNeu = (Kontakte) u.unmarshal(  
7     new File("/tmp/andereKontakte.xml"));  
8  
9     personListe = kontakteNeu.getPerson();
```

## Unmarshalling: XML-Dokument nach Java-Objekt

- Unmarshalling: Erzeugung von Objekten der mit `xjc` generierten Klassen. Hier: `Kontakte` und `Kontakte.Person`
- Code Schnipsel:

```
1 private List<Kontakte.Person> personListe;  
2  
3 JAXBContext context = JAXBContext.newInstance(Kontakte.  
4     class);  
5 Unmarshaller u = context.createUnmarshaller();  
6 Kontakte kontakteNeu = (Kontakte) u.unmarshal(  
7     new File("/tmp/andereKontakte.xml"));  
8  
9     personListe = kontakteNeu.getPerson();
```

## Unmarshalling: XML-Dokument nach Java-Objekt

- Unmarshalling: Erzeugung von Objekten der mit `xjc` generierten Klassen. Hier: `Kontakte` und `Kontakte.Person`
- Code Schnipsel:

```
1 private List<Kontakte.Person> personListe;  
2  
3 JAXBContext context = JAXBContext.newInstance(Kontakte.  
4     class);  
5 Unmarshaller u = context.createUnmarshaller();  
6 Kontakte kontakteNeu = (Kontakte) u.unmarshal(  
7     new File("/tmp/andereKontakte.xml"));  
8  
9     personListe = kontakteNeu.getPerson();
```

## Unmarshalling: XML-Dokument nach Java-Objekt

- Unmarshalling: Erzeugung von Objekten der mit `xjc` generierten Klassen. Hier: `Kontakte` und `Kontakte.Person`
- Code Schnipsel:

```
1 private List<Kontakte.Person> personListe;  
2  
3 JAXBContext context = JAXBContext.newInstance(Kontakte.  
4     class);  
5 Unmarshaller u = context.createUnmarshaller();  
6 Kontakte kontakteNeu = (Kontakte) u.unmarshal(  
7     new File("/tmp/andereKontakte.xml"));  
8  
9     personListe = kontakteNeu.getPerson();
```

# Danksagung

Vielen Dank für's Zuhören und Mitmachen!