

# DTDs und XML-Schemata

## Wie geht es weiter?

### letzte Woche

- Syntax wohlgeformter XML-Dokumente ✓
- Namensräume ✓
- Festlegung der Semantik von XML-Elementen ✓

### heute

- Beschreibung von Dokument-Typen
- DTDs und XML-Schema anhand eines einheitlichen Beispiels

### nächste Woche

- XML-Schema im Detail

## Beispiel Literaturreferenz

oder so?

```
<book>
<title>My Life and Times</title>
<authors>
<author>
<first>Paul</first>
<last>McCartney</last>
</author>
</authors>
<date>
<year>1998</year>
<month>July</month>
</date>
<isbn>94303-12021-43892</isbn>
<publisher>McMillin Publishing</publisher>
</book>
```

so?

```
<Book>
<Title>My Life and Times</Title>
<Author>Paul McCartney</Author>
<Date>July, 1998</Date>
<ISBN>94303-12021-43892</ISBN>
<Publisher>McMillin Publishing</Publisher>
</Book>
```

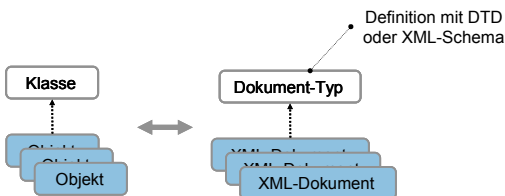
häufig Standardisierung nötig

## Dokument-Typ

- beschreibt *prinzipiellen* Aufbau und Datentypen der Inhalte, keine *konkreten* Inhalte
- beschreibt damit Klasse von XML-Dokumenten
- definiert eine Art Standard

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

## Dokument-Typ



- kann entweder mit einer DTD (*Document Type Definition*) oder einem XML-Schema definiert werden

## DTDs vs. XML-Schemata

- DTDs wurden von SGML übernommen, sind Teil von XML 1.0
- XML-Schema eigener W3C-Standard
- XML-Schemata ausdrucksstärker
- DTDs kompakter und lesbarer
- DTDs zur Beschreibung von Text-Dokumenten ausreichend
- XML-Schemata zur Beschreibung von Daten besser geeignet.

# Document Type Definitions (DTDs)

## Wie könnte eine DTD hierfür aussehen?

```
<BookStore>
  <Book>
    <Title>My Life and Times</Title>
    <Author>Paul McCartney</Author>
    <Date>July, 1998</Date>
    <ISBN>94303-12021-43892</ISBN>
    <Publisher>McMillin Publishing</Publisher>
  </Book>
</BookStore>
```

- BookStore soll mindestens ein Buch enthalten.
- ISBN optional
- alle anderen Kind-Elemente obligatorisch

## Die DTD für das Beispiel-Dokument

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

ähnelt einer regulären Grammatik

## Deklaration von BookStore

```
<!ELEMENT BookStore (Book+)>
  <BookStore>
    <Book>...</Book>
    <Book>...</Book>
  </BookStore>
```

- BookStore hat mindestens ein Kind-Element Book.
- + bezeichnet n Wiederholung des vorstehenden Elementes mit  $n \geq 1$ .
- \* bezeichnet n Wiederholung mit  $n \geq 0$ .
- Außer Book darf BookStore keine anderen Kind-Elemente haben.

## Rekursive Deklaration von BookStore

```
<!ELEMENT BookStore (Book | (Book, BookStore))>
```

- Bookstore besteht aus genau einer der folg. Alternativen:
  - genau ein Kind-Element Book
  - zwei Kind-Elemente, nämlich Book und BookStore
- | bezeichnet Auswahl (Disjunktion).
- , bezeichnet Aufeinanderfolge (Sequenz) von Elementen.
- **Beachte:** Rekursive Deklaration *nicht* äquivalent zur vorherigen, iterativen Definition.

## Rekursive vs. iterative Deklaration

```
<BookStore>
  <Book>...</Book>
  <Book>...</Book>
</BookStore>
<!ELEMENT BookStore (Book+)>

<BookStore>
  <Book>...</Book>
  <BookStore>
    <Book>...</Book>
  </BookStore>
</BookStore>
<!ELEMENT BookStore (Book | (Book, BookStore))>
```

## Deklaration von Book

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

- Title, Author, Date, ISBN und Publisher erscheinen (in dieser Reihenfolge) als Kind-Elemente von Book.
- außer diesen keinen anderen Kind-Elemente
- ? bedeutet, dass Element optional ist.

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

## Deklaration von Title etc.

```
<!ELEMENT Title (#PCDATA)>
<!ELEMENT Author (#PCDATA)>
<!ELEMENT Date (#PCDATA)>
<!ELEMENT ISBN (#PCDATA)>
<!ELEMENT Publisher (#PCDATA)>
```

```
<Title>My Life and Times</Title>
<Author>Paul McCartney</Author>
<Date>July, 1998</Date>
<ISBN>94303-12021-43892</ISBN>
<Publisher>McMillin Publishing</Publisher>
```

- #PCDATA: unstrukturierter Text ohne reservierte Symbole wie „<“.

## Datentypen für Element-Inhalte

- nur drei verschiedene Datentypen stehen zur Verfügung:
- **#PCDATA**: unstrukturierter Text ohne reservierte Symbole.
- **EMPTY**: leerer Inhalt, Element kann aber Attribute haben  

```
<!ELEMENT br EMPTY>
```

 → 

```
<br>
```
- **ANY**: beliebiger Inhalt (strukturiert, unstrukturiert, gemischt oder leer)  

```
<!ELEMENT title ANY>
```
- **Beachte**: Datentypen wie INTEGER oder FLOAT stehen *nicht* zur Verfügung.

## Deklaration von Attributen

```
<!ATTLIST BookStore
  version CDATA #IMPLIED "1.0">
```

- Element BookStore hat Attribut version.
- Außer version hat BookStore keine weiteren Attribute.
- **CDATA**: Attribut ist vom Typ String (ohne reservierte Symbole!)
- **#IMPLIED**: Attribut optional
- **"1.0"**: Standard-Wert des Attributes
- **#REQUIRED**: Attribut obligatorisch
- **#FIXED**: Attribut hat immer den gleichen Wert.

## Vorgegebene Attribut-Werte

```
<!ATTLIST Author
  gender (male | female) "female">
```

- Element Author hat Attribut gender und keine weiteren Attribute.
- Attribut gender hat entweder den Wert male oder female (Aufzählungstyp).
- "female" ist Standard-Wert von gender.

## Datentypen für Attribute

- Neben Strings (CDATA) und Aufzählungstypen stehen folgende Datentypen zur Verfügung:
- **NMTOKEN**: String, der Namenskonventionen von XML entspricht
- **NMTOKENS**: Liste von solchen Namen, jeweils getrennt durch ein Leerzeichen
- **ID**: Bezeichner, der Namenskonventionen von XML entspricht und innerhalb des XML-Dokumentes eindeutig ist.
- **IDREF**: Referenz auf einen eindeutigen Bezeichner
- **IDREFS**: Liste von solchen Referenzen, jeweils getrennt durch ein Leerzeichen

## ID/IDREF

```
<!ATTLIST Author
  key ID #IMPLIED
  keyref IDREF #IMPLIED>
```

- optionale Attribute key und keyref
- key muss im XML-Dokument *eindeutig* sein:  
Zwei Attribute vom Typ ID dürfen niemals gleichen Wert haben.
- keyref muss *gültige Referenz* sein:  
Wert von keyref muss als Wert eines Attributes vom Typ ID erscheinen.

## Beispiel

```
<BookStore>
  <Book>
    <Title>Text</Title>
    <Author key="k1">Text</Author>
    <Date>Text</Date>
    <Publisher>Text</Publisher>
  </Book>
  <Book>
    <Title>Text</Title>
    <Author keyref="k1"/>
    <Date>Text</Date>
    <Publisher>Text</Publisher>
  </Book>
</BookStore>
```

Wert k1 muss eindeutig sein: kein anderes Attribut vom Typ ID darf diesen Wert haben.

Referenz k1 muss existieren: ein Attribut vom Typ ID muss den Wert k1 haben.

## Wohlgeformtheit vs. Zulässigkeit

### wohlgeformt (*well formed*)

XML-Dokument entspricht syntaktischen Regeln von XML 1.0

### zulässig (*valid*) bzgl. einer DTD

1. Wurzel-Element des XML-Dokumentes ist in der DTD deklariert und
2. Wurzel-Element hat genau die Struktur, wie sie in der DTD festgelegt ist.

## Festlegung des Dokument-Typs

- Prozessorinstruktion:

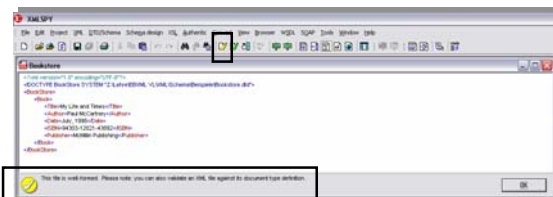
```
<!DOCTYPE Wurzel-Element SYSTEM "DTD">
```

- legt Wurzel-Element und Dokument-Typ fest

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE BookStore SYSTEM "Bookstore.dtd">
<BookStore>
  ...
</BookStore>
```

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE Book SYSTEM "Bookstore.dtd">
<Book>
  ...
</Book>
```

## Prüfung der Wohlgeformtheit



## Prüfung der Zulässigkeit



## Nachteile von DTDs

- keine XML-Syntax, eigener Parser nötig
- nur sehr wenige Datentypen, insbesondere für Element-Inhalte
- keine eigenen Datentypen definierbar
- keine Namensräume:  
DTDs können nur dann kombiniert werden, wenn es *keine* Namenskonflikte gibt!
- keine Vererbungshierarchien, nicht objekt-orientiert

## Und noch ein Nachteil

- Reihenfolge von Kind-Elementen festgelegt:  
<!ELEMENT Book (Title, Author)>
- dadurch starre Struktur
- Für Reihenfolgeunabhängigkeit müssen alle Permutationen explizit aufgezählt werden:  
<!ELEMENT Book ((Title, Length) | (Length, Title))>
- nicht praktikabel: bei n Kind-Elementen n! Permutationen

## XML-Schemata

## Warum XML-Schema?

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

XML-Schema  
↓  
DTD

- **Ortsangabe:** besteht aus Breitengrad, Längengrad und Unsicherheit.
- **Breitengrad:** Dezimalzahl zwischen -90 und +90
- **Längengrad:** Dezimalzahl zwischen -180 und +180
- **Unsicherheit:** nicht-negative Zahl
- **Maßeinheit für Unsicherheit:** Meter oder Fuß

## Vorteile von XML-Schemata

- + Ähnlich wie in Programmiersprachen, steht Vielzahl von vordefinierten Datentypen zur Verfügung.
- + *Eigene* Datentypen können definiert werden.
- + keine eigene Syntax, sondern selbst wiederum XML-Dokumente
- + Vererbungshierarchien
- + integrieren Namensräume
- + Reihenfolgeunabhängige Strukturen können einfach definiert werden.

## Die Beispiel-DTD

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

## Äquivalentes XML-Schema

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.books.org"
  xmlns="http://www.books.org"
  elementFormDefault="qualified">
  <xsd:element name="BookStore">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="Book" maxOccurs="unbounded">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name="Title" type="xsd:string"/>
              <xsd:element name="Author" type="xsd:string"/>
              <xsd:element name="Date" type="xsd:string"/>
              <xsd:element name="ISBN" type="xsd:string"/>
              <xsd:element name="Publisher" type="xsd:string"/>
            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

Für jede DTD gibt es ein äquivalentes XML-Schema.  
 Übersetzung z.B. mit XML Spy  
 Umgekehrt gibt es jedoch XML-Schemata, für die es keine äquivalente DTD gibt.  
 → XML-Schemata *ausdrucksmächtiger* als DTDs

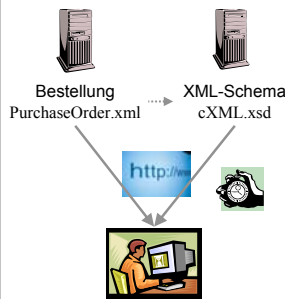
## Äquivalentes XML-Schema

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.books.org"
  xmlns="http://www.books.org"
  elementFormDefault="qualified">
  <xsd:element name="BookStore">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="Book" maxOccurs="unbounded">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name="Title" type="xsd:string"/>
              <xsd:element name="Author" type="xsd:string"/>
              <xsd:element name="Date" type="xsd:string"/>
              <xsd:element name="ISBN" type="xsd:string"/>
              <xsd:element name="Publisher" type="xsd:string"/>
            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

- XML-Schema sind XML-Dokumente.
- Vorteil: kein eigener Parser nötig
- Nachteil: geschwätzig

## Geschwätzigkeit von XML-Schemata

- Beispiel cXML:
  - als DTD: 15KB
  - als Schema: 50KB
  - also mehr als dreimal so groß
- kann Validierung von XML-Dokumenten auf Client erschweren
- XML-Schemata aber besser zu komprimieren:
  - cXML-DTD: 4KB
  - cXML-Schema: 6KB



## XML-Schemata

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

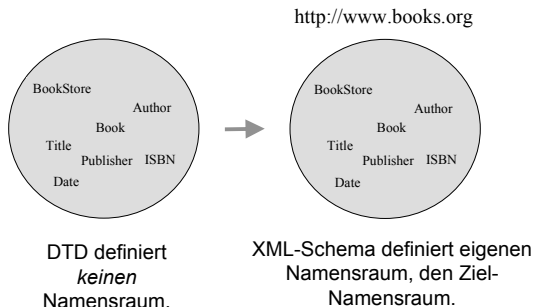
- XML-Schema ist XML-Dokument
- Wurzel-Element immer schema aus W3C-Namensraum `http://www.w3.org/2001/XMLSchema`
- hier ein XML-Schema für XML-Schema hinterlegt
- wird auch Schema der Schemata genannt

## Ziel-Namensraum

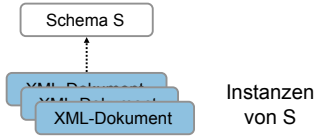
```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.books.org"
  xmlns="http://www.books.org"
  elementFormDefault="qualified">
  ...
</xsd:schema>
```

- XML-Schema definiert bestimmtes Vokabular (Elemente und Attribute).
- Dieses Vokabular wird einem Namensraum zugeordnet: dem Ziel-Namensraum (*target namespace*).
- Vokabular kann so global identifiziert werden.

## Ziel-Namensraum



## Instanz eines XML-Schemas



- XML-Dokument, das dem Ziel-Namensraum eines XML-Schemas zugeordnet ist
- Instanz muss entsprechend dem XML-Schema aufgebaut sein.

## Instanz eines XML-Schemas

```
<?xml version="1.0"?>
<BookStore>
  <Book>
    <Title>My Life and Times</Title>
    <Author>Paul McCartney</Author>
    <Date>July, 1998</Date>
    <ISBN>94303-12021-43892</ISBN>
    <Publisher>McMillin Publishing</Publisher>
  </Book>
  ...
</BookStore>
```

Wie mache ich hieraus eine Instanz eines bestimmten XML-Schemas?

### 1. Schritt

```
<?xml version="1.0"?>
<BookStore xmlns="http://www.books.org"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.books.org
    BookStore.xsd">
  <Book>
    <Title>My Life and Times</Title>
    <Author>Paul McCartney</Author>
    <Date>July, 1998</Date>
    <ISBN>94303-12021-43892</ISBN>
    <Publisher>McMillin Publishing</Publisher>
  </Book>
</BookStore>
```

Ziel-Namensraum des XML-Schemas

1. Wurzel-Element und dessen Namensraum legen zusammen den Dokument-Typ fest.

### 2. Schritt

```
<?xml version="1.0"?>
<BookStore xmlns="http://www.books.org"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.books.org
    BookStore.xsd">
  <Book>
    <Title>My Life and Times</Title>
    <Author>Paul McCartney</Author>
    <Date>July, 1998</Date>
    <ISBN>94303-12021-43892</ISBN>
    <Publisher>McMillin Publishing</Publisher>
  </Book>
</BookStore>
```

lokale Datei, kann aber auch ein Pfad (file://...) oder eine URL sein.

2. Attribut schemaLocation gibt an, wo das entsprechende XML-Schema zu finden ist.

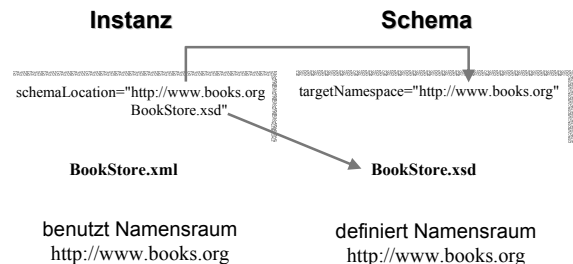
### 3. Schritt

```
<?xml version="1.0"?>
<BookStore xmlns="http://www.books.org"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.books.org
    BookStore.xsd">
  <Book>
    <Title>My Life and Times</Title>
    <Author>Paul McCartney</Author>
    <Date>July, 1998</Date>
    <ISBN>94303-12021-43892</ISBN>
    <Publisher>McMillin Publishing</Publisher>
  </Book>
</BookStore>
```

weiteres Beispiel für die Erweiterbarkeit von XML!  
XML 1.0 wird durch Attribut schemaLocation erweitert.

3. Attribut schemaLocation stammt aus einem W3C-Namensraum für Schema-Instanzen.

## Instanz und Schema

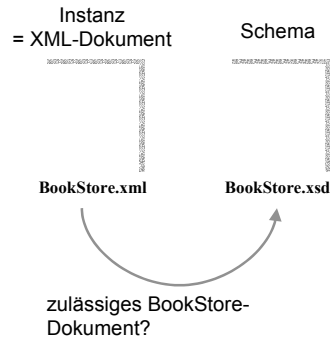


## Software zum Validieren von Instanzen

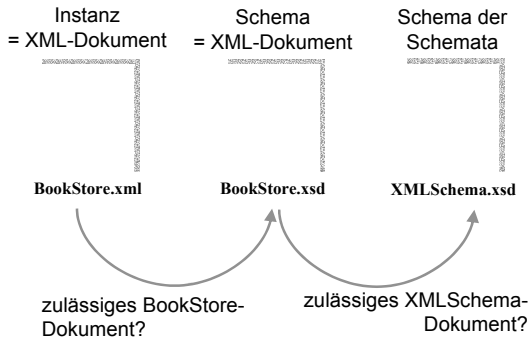


- xerces by Apache (API)
  - <http://www.apache.org/xerces-j/index.html>
- MSXML (API)
  - <http://www.microsoft.com>
- XML Spy (GUI)
  - <http://www.xmlspy.com>
- weitere: <http://www.w3.org/XML/Schema#Tools>

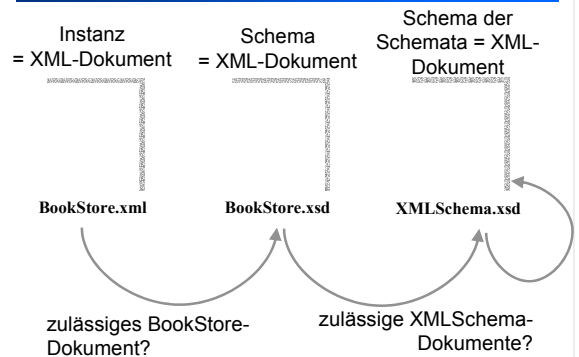
## Validierung auf mehrere Ebenen



## Validierung auf mehrere Ebenen



## Validierung auf mehrere Ebenen



## Deklaration der Element-Struktur



```

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

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

## <!ELEMENT BookStore (Book+)>



```

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

```

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

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

## <!ELEMENT BookStore (Book+)>

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

- minOccurs: minimale Anzahl von Wiederholungen
- maxOccurs: maximale Anzahl von Wiederholungen
- **Beachte:** Standard-Werte für minOccurs und maxOccurs jeweils 1

© Klaus Schild, 2004

49

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

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

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

© Klaus Schild, 2004

50

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

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

- xsd:string: vordefinierter Datentyp

© Klaus Schild, 2004

51

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

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

- xsd.date: vordefinierter Datentyp

© Klaus Schild, 2004

52

## Benannte Datentypen

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

- BookType ist ein benannter Datentyp (*named type*)
- wird auch als globale Definition bezeichnet

© Klaus Schild, 2004

53

## Anonyme Datentypen

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

- äquivalente Formulierung mit anonymen Datentyp
- wird auch als lokale Definition bezeichnet
- kann an anderer Stelle *nicht* wieder verwendet werden

© Klaus Schild, 2004

54

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

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

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

## ElementFormDefault

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

- **"qualified"**: Elemente der Instanz *müssen* namensraumeingeschränkt sein.
- **"unqualified"**: Elemente *dürfen nicht* namensraumeingeschränkt sein.
- **Beachte**: Standard-Wert ist "unqualified".

## ElementFormDefault

```
<?xml version="1.0"?>
<BookStore xmlns="http://www.books.org"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.books.org
    BookStore.xsd">
  <Book>
    <Title>My Life and Times</Title>
    <Author>Paul McCartney</Author>
    <Date>July, 1998</Date>
    <ISBN>94303-12021-43892</ISBN>
    <Publisher>McMillin Publishing</Publisher>
  </Book>
</BookStore>
```

Namensraum-  
eingeschränkt  
(qualified)

- Instanz *nicht* zulässig, falls im Schema elementFormDefault="unqualified" oder nichts festgelegt ist.

## Das vollständige XML-Schema

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.books.org"
  xmlns="http://www.books.org"
  elementFormDefault="qualified">
  <xsd:element name="BookStore">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="Book" maxOccurs="unbounded">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name="Title" type="xsd:string"/>
              <xsd:element name="Author" type="xsd:string"/>
              <xsd:element name="Date" type="xsd:string"/>
              <xsd:element name="ISBN" type="xsd:string"/>
              <xsd:element name="Publisher" type="xsd:string"/>
            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

## Wie geht es weiter?

### heute

- Beschreibung von Dokument-Typen ✓
- DTDs und XML-Schema anhand eines einheitlichen Beispiels ✓

### nächste Woche

- XML-Schema im Detail