Beschreibung von Dokumenten mit DTDs und XML-Schemata

© Klaus Schild, 2003

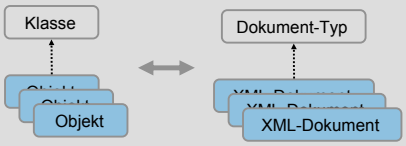
Lernziele

- Was sind Dokument-Typen?
- Wie können diese mit einer DTD oder einem Schema beschrieben werden?
- Was ist der Vorteil von XML-Schemata gegenüber DTDs?

© Klaus Schild, 2003

Dokument-Typen

- beschreiben den *prinzipiellen Aufbau* von Dokumenten eines bestimmten Typs
- können entweder mit DTDs (*Document Type Definitions*) oder XML-Schemata spezifiziert werden
- DTDs wurden von SGML übernommen und sind Teil von XML 1.0.
- XMLSchema sind ein eigener W3C-Standard.



© Klaus Schild, 2003

DTDs vs. XML-Schemata

- XML-Schemata sind ausdrucksstärker
- DTDs sind kompakter und lesbarer
- DTDs für Spezifikation von Text-Dokumenten ausreichend
- XML-Schemata zur Spezifikation von Daten besser geeignet.

© Klaus Schild, 2003

Document Type Definitions (DTDs)

© Klaus Schild, 2003

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>
```

- Ein BookStore soll mindestens ein Buch enthalten.
- Die ISBN soll optional, alle anderen Kind-Elemente sollen obligatorisch sein.

© Klaus Schild, 2003

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 der Elementstruktur

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

```
<BookStore>
   <Book>…</Book>
   <Book>…</Book>
</BookStore>
```

- deklariert das Element BookStore
- BookStore hat mindestens ein Kind-Element Book.
- Außer Book darf BookStore keine Kind-Elemente haben.
- + bezeichnet n Wiederholung des vorstehenden Elementes mit n ≥ 1.
- * bezeichnet n Wiederholung mit n ≥ 0.

Deklaration der Elementstruktur

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

- deklariert das Element Book
- Title, Author, Date, ISBN und Publisher erscheinen (in dieser Reihenfolge) als Kind-Elemente von Book.
- Außer diesen darf Book keinen anderen Kind-Element haben.
- **,** bezeichnet Sequenz (Aufeinanderfolge) von Elementen.
- **?** bedeutet, dass das vorstehende Element in der Sequenz optional ist.

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


Rekursive Element-Deklarationen

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

- rekursive Deklaration von BookStore
- Bookstore besteht entweder aus genau einem Kind-Element Book oder hat zwei Kind-Elemente, nämlich Book und BookStore.
- | bezeichnet Auswahl (Disjunktion).
- <u>Beachte</u>: Diese rekursive Deklaration ist *nicht* äquivalent zur vorherigen:

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


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 der Elementstruktur

```
<!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>
```

- Deklariert jeweils Elemente Title, Author, Date, ISBN und Publisher.
- #PCDATA bezeichnet unstrukturierter Text ohne reservierte Symbole wie „<“.

Vordefinierte Datentypen

- Zur Festlegung von Element-Inhalten stehen drei vordefinierte Datentypen (engl. *built-in datatypes*) zu Verfügung:
- #PCDATA: unstrukturierter Text ohne reservierte Symbole.
- EMPTY: Der Element-Inhalt ist leer. Das Element kann allerdings Attribute haben.

  <!ELEMENT br EMPTY> ➔
- ANY: beliebige XML-Strukturen

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

© Klaus Schild, 2003

Deklaration von Attributen

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

- Das Element BookStore hat ein Attribut version.
- Außer version hat BookStore keine weiteren Attribute.
- Das Attribut version ist vom Typ String (CDATA).
- #IMPLIED: Das Attribut ist optional.
- "1.0" ist der Standard-Wert.
- #REQUIRED: Das Attribut ist obligatorisch.
- #FIXED: Das Attribut hat immer den gleichen Wert.

© Klaus Schild, 2003

Deklaration von Attributen

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

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

© Klaus Schild, 2003

Datentypen für Attribute

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

© Klaus Schild, 2003

ID/IDREF

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

- Author kann ein Attribut key haben.
- Das Attribut key muss *eindeutig* sein: Attribute mit dem Typ ID dürfen niemals den gleichen Wert haben.
- Author kann auch ein Attribut keyref haben.
- Der Wert von keyref muss eine *gültige Referenz* darstellen: Der Wert von keyref muss als Wert eines Attributes mit dem Typ ID existieren.

© Klaus Schild, 2003

ID/IDREF

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

k1 muss eindeutig sein: weder dieses noch ein anderes Attribut mit dem Typ ID darf **k1** als Wert haben.

k1 muss existieren: ein Attribut mit dem Typ ID muss den Wert **k1** haben.

Wohlgeformheit vs. Zulässigkeit

- Ein XML-Dokument heißt *wohlgeformt*, wenn es den syntaktischen Regeln des entsprechenden W3C-Standards entspricht.
- Ein XML-Dokument heißt *zulässig* (engl. *valid*). bzgl. einer Dokument-Typ-Definition, wenn
  1. das Wurzel-Element des XML-Dokumentes in der DTD deklariert ist und
  2. das Wurzel-Element genau die Struktur hat, wie sie in der DTD festgelegt ist.

© Klaus Schild, 2003

Zulässigkeit

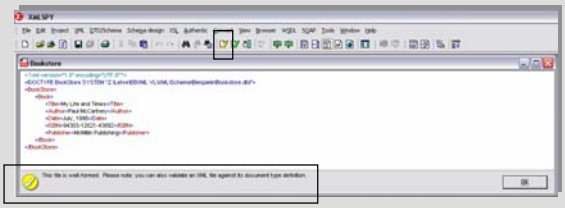
- In einem XML-Dokument kann ein Dokument-Typ spezifiziert werden:

```
<?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>
```

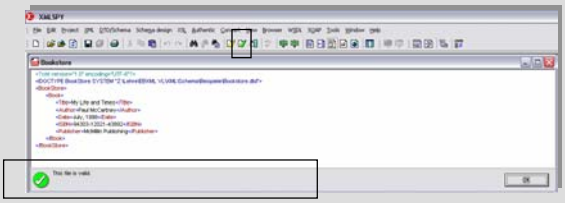
© Klaus Schild, 2003

Prüfung der Wohlgeformheit



© Klaus Schild, 2003

Prüfung der Zulässigkeit



© Klaus Schild, 2003

Nachteil von DTDs

- Reihenfolge von Kind-Elementen ist festgelegt:

  <!ELEMENT Book (Title, Author)>
- dadurch sehr starre Struktur in XML-Dokumenten
- Um Reihenfolgeunabhängigkeit zu garantieren, müssen alle Permutationen explizit aufgezählt werden:

  <!ELEMENT Book ((Title, Length) | (Length, Title))>
- Für n Element gibt es n! verschiedene Permutationen.

© Klaus Schild, 2003

Weitere Nachteile von DTDs

- keine XML-Syntax, daher eigene Parser nötig
- kaum vordefinierte Datentypen, insbesondere für Element-Inhalte
- keine eigenen Datentypen definierbar
- keine Namensräume: Bereits existierende DTDs können nur dann kombiniert werden, wenn es *keine* Namenskonflikte gibt!
- keine Vererbungshierarchien, nicht objekt-orientiert

© Klaus Schild, 2003

XML-Schemata

Was sind XML-Schemata?

- Ähnlich wie eine DTD legt ein XML-Schema (engl. *XML schema*) den prinzipiellen Aufbau von XML-Dokumenten eines bestimmten Typs fest.
- DTDs wurden zur Beschreibung von strukturierten (für Menschen lesbare) Text-Dokumenten entwickelt.
- Für die Beschreibung von Dokumenten/Daten zum Austausch zwischen Computern sind sie allerdings zu ausdrucksschwach.
- Deshalb wurden XML-Schemata entwickelt.

Vorteile von XML-Schemata

- Ähnlich wie in vielen Programmiersprachen, steht eine Vielzahl von vordefinierten Datentypen zur Verfügung.
- Es können auch *eigene* Datentypen definiert werden.
- keine eigene Syntax, sondern XML-Schema sind selbst XML-Dokumente
- objekt-orientiert, erlauben Vererbungshierarchien
- verwenden Namensräume
- Reihenfolgeunabhängige Strukturen können einfach definiert werden.

Datentypen

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

XML-Schema

DTD

- Eine Ortsangabe besteht aus dem Breitengrad, dem Längengrad und einem Maß für die Unsicherheit der beiden Angaben.
- Ein Breitengrad ist eine Dezimalzahl zwischen -90 und +90.
- Ein Längengrad ist eine Dezimalzahl zwischen -180 und +180.
- Das Maß für die Unsicherheit ist eine nicht-negative Zahl.
- Die Unsicherheit wird entweder in Meter oder in Fuß angeben.

Validierung von Daten

Daten

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

XML-Schema

- Eine Ortsangabe besteht aus dem Breitengrad, dem Längengrad und einem Maß für die Unsicherheit der beiden Angaben.
- Ein Breitengrad ist eine Dezimalzahl zwischen -90 und +90.
- Ein Längengrad ist eine Dezimalzahl zwischen -180 und +180.
- Das Maß für die Unsicherheit ist eine nicht-negative Zahl.
- Die Unsicherheit wird entweder in Meter oder in Fuß angeben.

XML-Schema Validierer

Daten ✓

Software zum Validieren

- 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

➔ http://www.w3.org/XML/Schema#Tools

Unsere bekannte 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)>
```

© Klaus Schild, 2003

Die gleiche DTD als XML-Schema

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

- Jede DTD kann in ein äquivalentes XML-Schema übersetzt werden.
- Tools wie XML Spy bieten eine entsprechende Funktionalität an.
- Umgekehrt gibt es allerdings XML-Schemata, für die es *keine* äquivalente DTD gibt.

➔ XML-Schemata sind also *ausdrucksmächtiger* als DTDs.

© Klaus Schild, 2003

Die gleiche DTD als XML-Schema

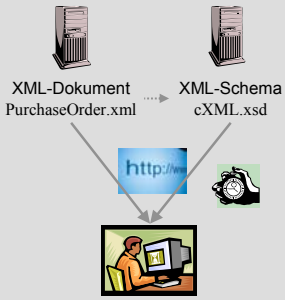
```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            targetNamespace="http://www.books.org"
            xmlns="http://www.books.org"
            elementFormDefault="qualified">
  <xsd:element name="BookStore">
    <xsd:complexType>
      <xsd:sequence>
     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

© Klaus Schild, 2003

Geschwätzigkeit von XML-Schemata

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



© Klaus Schild, 2003

XML-Schemata sind XML-Dokumente

```
<?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>
```

- Ein XML-Schema ist ein XML-Dokument.
- Wurzel-Element eines Schemas ist immer „schema“ aus dem W3C-Namensraum „XMLSchema“.
- Letzteres wird auch Schema der Schemata genannt.

© Klaus Schild, 2003

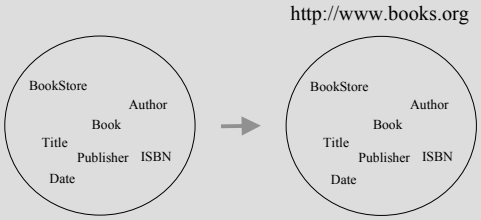
Ziel-Namensraum

```
<?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>
```

- Ein XML-Schema definiert ein bestimmtes Vokabular (Elemente und Attribute).
- Dieses Vokabular wird einem Namensraum zugeordnet, den sog. Ziel-Namensraum (engl. *target namespace*).

© Klaus Schild, 2003

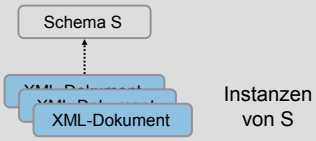
Ziel-Namensraum



Eine DTD definiert *keinen* Namensraum.

Ein XML-Schema definiert eigenen Namensraum, den sog. Ziel-Namensraum.

Instanzen eines XML-Schemas



- Sprechweise: Ein XML-Dokument, das dem Ziel-Namensraum eines XML-Schemas zugeordnet ist, wird *Instanz* des Schemas genannt.
- Instanz muss immer entsprechend der Schema-Beschreibung 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 wird ausgedrückt, dass dieses XML-Dokument eine Instanz eines bestimmten XML-Schemas ist?

Instanz eines XML-Schemas

```
<?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 der Instanz fest.

Instanz eines XML-Schemas

```
<?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 (http://...) sein.

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

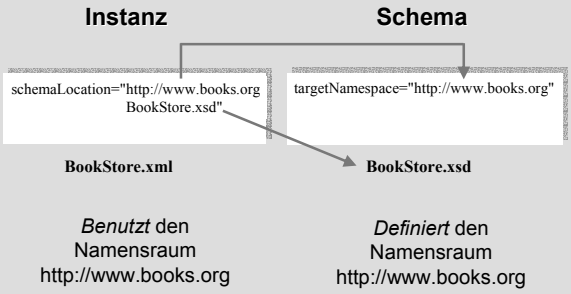
Instanz eines XML-Schemas

```
<?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</Da
        <ISBN>94303-12021-
        <Publisher>McMillin
    </Book>
</BookStore>
```

- Beispiel für die Erweiterbarkeit von XML
- XML 1.0 wird durch das Attribut schemaLocation erweitert.

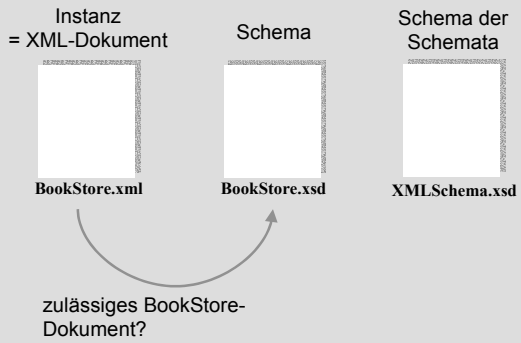
3. Das Attribut „schemaLocation“ stammt aus dem W3C-Namensraum „XMLSchema-instance“.

Instanz und Schema



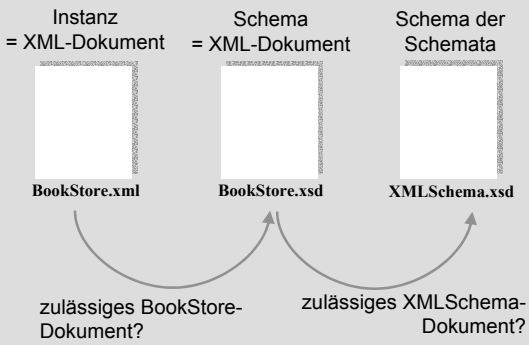
© Klaus Schild, 2003

Validierung auf mehrere Ebenen



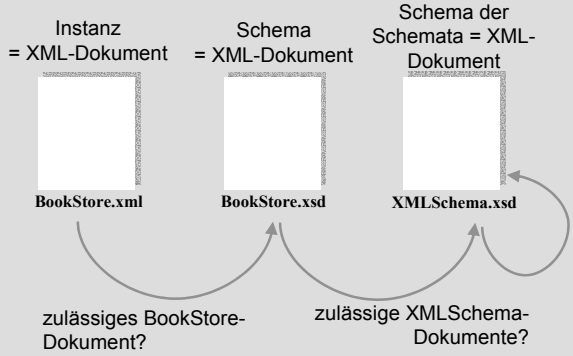
© Klaus Schild, 2003

Validierung auf mehrere Ebenen



© Klaus Schild, 2003

Validierung auf mehrere Ebenen



© Klaus Schild, 2003

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 kann die Element-Struktur dieser DTD mit einem XML-Schema formuliert werden?

© Klaus Schild, 2003

<!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>
```

- Standard-Werte für minOccurs und maxOccurs jeweils 1
- minOccurs="1" kann also weggelassen werden!

© Klaus Schild, 2003

<!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>
```

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

- BookType enthält die Kind-Elemente Title, Author, Date, ISBN und Publisher, und zwar immer genau in dieser Reihenfolge.

© Klaus Schild, 2003

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

```
<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 (engl. *named type*)
- wird auch als globale Definition bezeichnet

© Klaus Schild, 2003

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

```
<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, weil sie an anderer Stelle *nicht* wieder verwendet werden kann

© Klaus Schild, 2003

<!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 mittels minOccurs und maxOccurs festgelegt.
- Beachte: Standard-Wert von minOccurs und maxOccurs jeweils 1

© Klaus Schild, 2003

<!ELEMENT Date (#PCDATA)>

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

- deklariert Element Date mit unstrukturierten Inhalt (String).

```
<xsd:element name="Date" type="xsd:date"/>
```

- deklariert Element-Inhalt von Date als Kalenderdatum (vordefinierter Datentyp xsd:date).

© Klaus Schild, 2003

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>
```

- Legt fest, ob Elemente einer Instanz immer Teil eines Namensraumes sein müssen ("qualified") oder nicht Teil eines Namensraumes sein dürfen ("unqualified").
- Beachte: Standard-Wert von elementFormDefault ist "unqualified".

© Klaus Schild, 2003

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*)

- Beachte: Diese Instanz ist *nicht* zulässig, falls im Schema elementFormDefault="unqualified" oder nichts festgelegt ist.
- Im Schema deshalb *immer* elementFormDefault="qualified" festlegen!

© Klaus Schild, 2003

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>
```

© Klaus Schild, 2003