DTDs und XML-Schemata

Wie geht es weiter?

letzte Woche

- ☑ XML-Syntax
- ☑ Namensräume
- ☑ Semantik von XML-Elementen

heute

- Definition 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>
```

⇨ Standardisierung nötig

Standardisierung

- **prinzipieller Aufbau von Dokumenten**: Welche Elemente/Attribute dürfen wo verwendet werden?
- **Datentypen der Inhalte**: Welche Inhalte sind erlaubt?

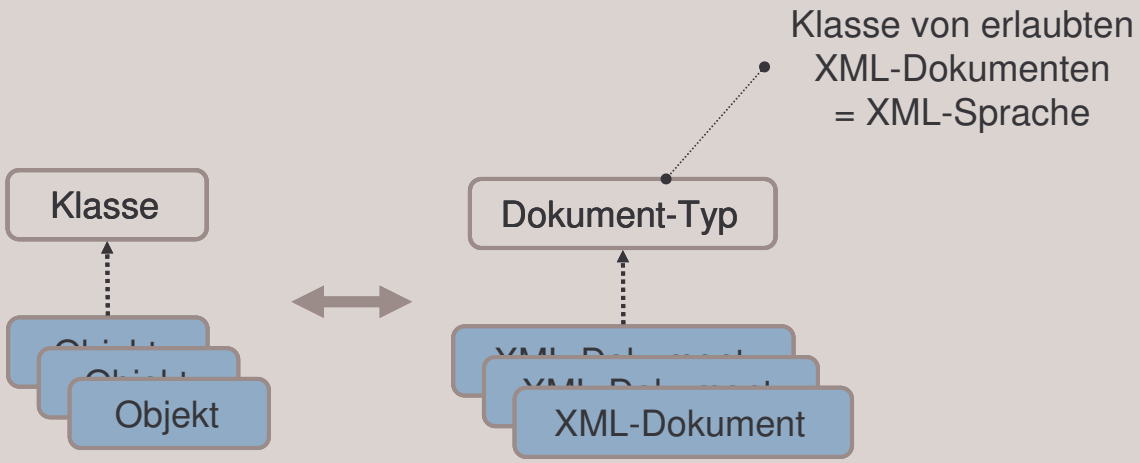
```
<Book>
   <Title> PCDATA </Title>
   <Author> PCDATA </Author>
   <Date> PCDATA </Date>
   <ISBN> PCDATA </ISBN>
   <Publisher> PCDATA </Publisher>
</Book>
```

- konkrete Inhalte werden *nicht* beschrieben

⇨ Klasse von erlaubten XML-Dokumenten

- auch **Dokument-Typ**, **XML-Sprache** oder **Anwendung** von XML genannt

Dokument-Typ



- Dokument-Typ kann mit einer DTD oder einem XML-Schema definiert werden.

DTDs vs. XML-Schemata

- DTDs = vereinfachte SGML-DTDs, Teil von XML 1.0/1.1.
- XML-Schema eigener W3C-Standard

- XML-Schemata ausdrucksstärker
- DTDs kompakter und lesbarer

- XML-Schemta = XML-Sprache
- DTDs haben eigene Syntax

- 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 ≥ 1.
- Außer Book darf BookStore keine anderen Kind-Elemente haben.
- Element-Deklaration genannt
- * bezeichnet n Wiederholung mit n ≥ 0.

Rekursive Deklarationen


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

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

Rekursive vs. iterative Deklaration


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

BookStore mit 3 Büchern

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

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

BookStore mit 3 Büchern

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

Deklaration von Book


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

- Title, Author, Date, ISBN und Publisher (in dieser Reihenfolge) Kind-Elemente von Book.
- außer diesen keinen anderen Kind-Elemente

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

- ? bedeutet, dass Element optional ist.

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 Inhalt ohne reservierte Symbole < und &.

Datentypen für Element-Inhalte

nur drei verschiedene Datentypen:

1. **#PCDATA**: unstrukturierter Inhalt ohne reservierte Symbole < und &.
2. **EMPTY**: leerer Inhalt, Element kann aber Attribute haben

   <!ELEMENT br EMPTY> ➔
3. **ANY**: beliebiger Inhalt (strukturiert, unstrukturiert, gemischt oder leer)

   <!ELEMENT title ANY>

- Beachte: Datentypen wie INTEGER oder FLOAT stehen *nicht* zur Verfügung.

Verschachtelungen

- beliebige Verschachtelung von Sequenz, Auswahl |, ?, *, + und Rekursion erlaubt
- Beispiel:

```
<!ELEMENT Chap (Title, (Text | Chap)+)>
<!ELEMENT Text ANY>
<!ELEMENT Title (#PCDATA)>
```

```
<Chap>
  <Title>Kap1</Title>
  <Text>…</Text>
  <Chap>
    <Title>Kap1.1</Title>
    <Text>…</Text>
  </Chap>
  <Text>…</Text>
  <Chap>
    <Title>Kap1.2</Title>
    <Text>…</Text>
  </Chap>
</Chap>
```

Deklaration von Attributen

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

```
<BookStore version="1.0">
   …
</BookStore>
```

- Element BookStore hat Attribut version.
- Außer version hat BookStore keine weiteren Attribute.
- **Attribut-Deklaration** genannt
- **CDATA**: Attribut-Wert ist String ohne <, &, ' und "
- Beachte: nicht zu verwechseln mit <![CDATA[…]]>
- daher Entity References für <, &, ' und " verwenden!

Deklaration von Attributen

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

- **#IMPLIED**: Attribut optional
- **"1.0"**: Standard-Wert des Attributes

⇨ wenn Attribut nicht vorhanden, fügt XML-Parser Attribut mit Standard-Wert hinzu

statt #IMPLIED auch möglich:

- **#REQUIRED**: Attribut obligatorisch
- **#FIXED**: Attribut hat immer den gleichen Wert.

Aufzählungstypen

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

- hier statt CDATA **Aufzählungstyp**:
- Attribut gender hat entweder den Wert male oder female (Aufzählungstyp).
- "female" ist Standard-Wert von gender.

Datentypen für Attribute

Zusätzlich zu CDATA (Strings) und Aufzählungstypen:

- **NMTOKEN**: String, der Namenskonventionen von XML entspricht
- **ID**: eindeutiger Bezeichner, der Namenskonventionen von XML entspricht
- **IDREF**: Referenz auf einen eindeutigen Bezeichner

ID/IDREF

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

- Wert des Attributes key muss *eindeutig* sein:

  Zwei Attribute vom Typ ID dürfen niemals gleichen Wert haben.
- Wert des Attributes 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.

Wohlgeformheit vs. Zulässigkeit

wohlgeformt (well formed)

- XML-Dokument entspricht syntaktischen Regeln von XML

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

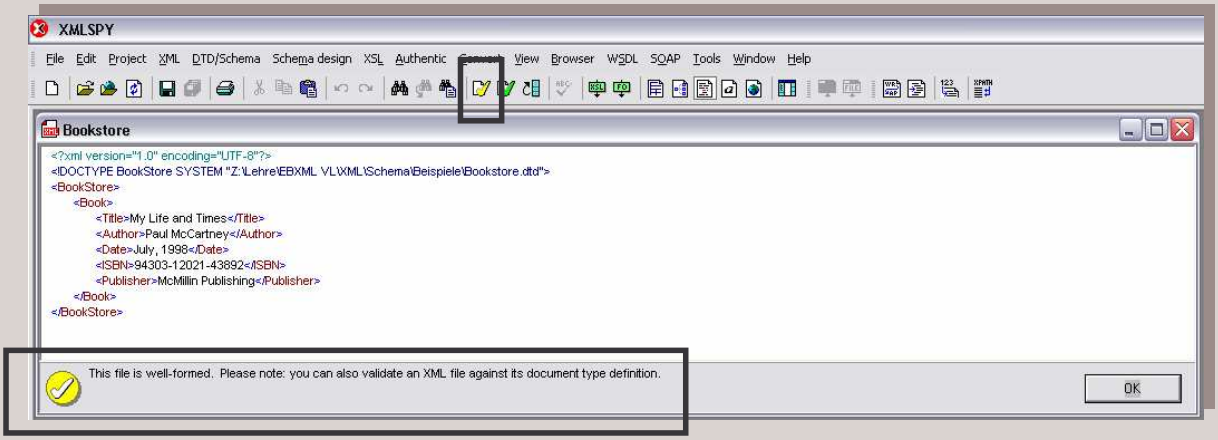
- Prozessorinstruktion direkt nach der XML-Deklaration:

  <!DOCTYPE *Wurzel-Element* SYSTEM "*DTD*">
- legt Wurzel-Element und Dokument-Typ fest

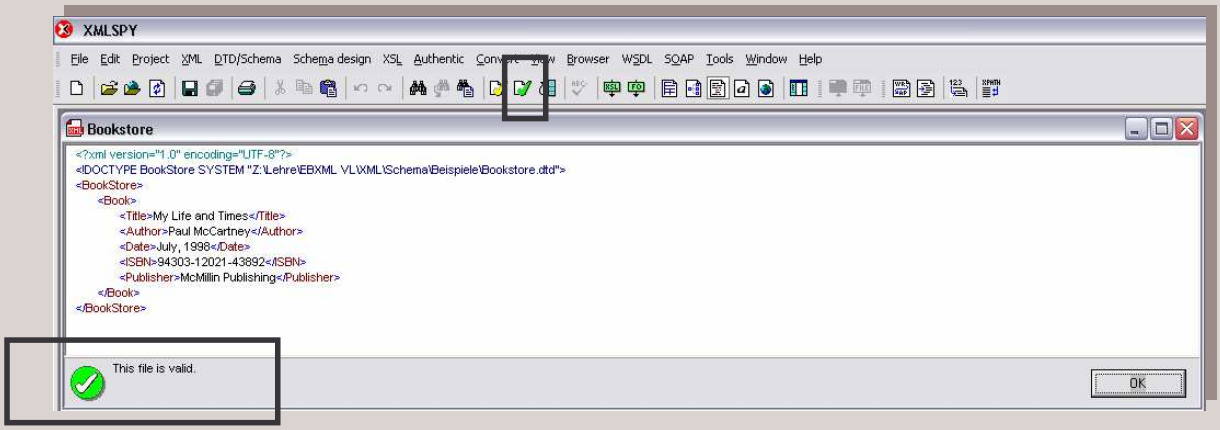
```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE BookStore SYSTEM "Bookstore.dtd">
<BookStore>
    …
</BookStore>
```

```
<?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

- Sequenzen einfach zu definieren:

  <!ELEMENT Book (Title, Author)>

⇨ starre Struktur in XML-Dokumenten

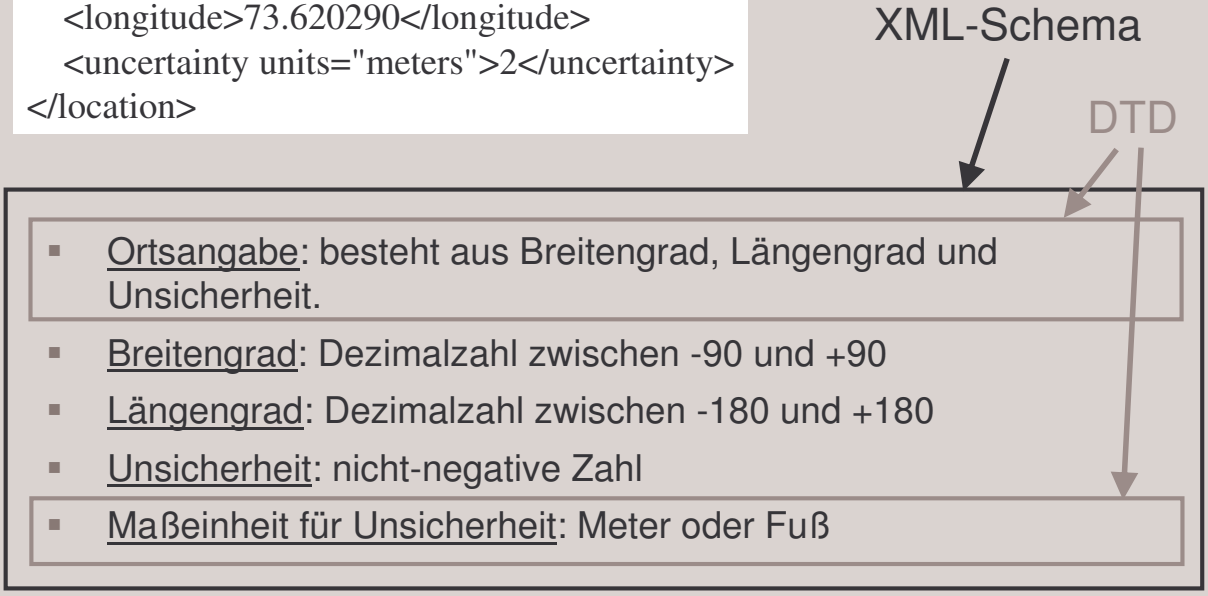
- soll Reihenfolge der Kind-Elmente egal sein, müssen alle Permutationen explizit aufgezählt werden:

  <!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>
```



Vorteile von XML-Schemata

- Wie in Programmiersprachen, steht Vielzahl von vordefinierten Datentypen zur Verfügung.
- eigene Datentypen definierbar
- keine eigene Syntax, sondern selbst XML-Sprache
- 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">
  <xsd:element name="BookStore">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="Book"  maxOccurs="unbounded">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name="Ti
              <xsd:element name="A
              <xsd:element name="D
              <xsd:element name="IS
              <xsd:element name="Pu
            </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

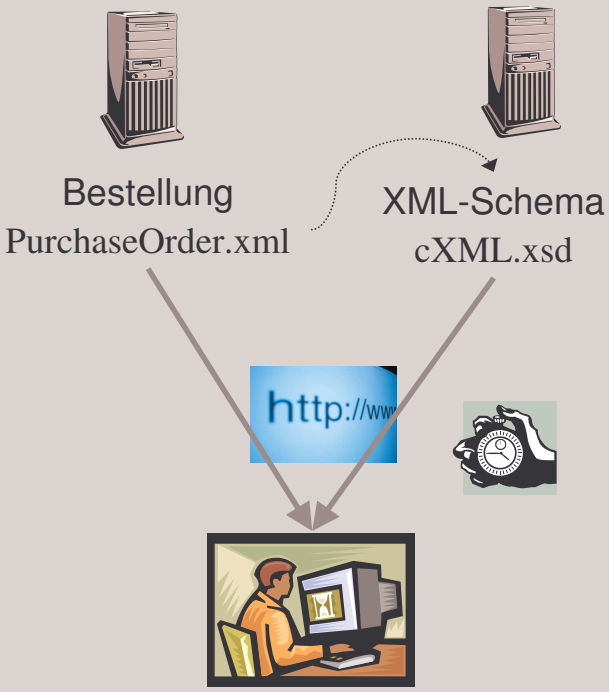
XML-Schemata


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

- XML-Schemata sind wohlgeformte XML-Dokumente.
- Vorteil: kein eigener Parser nötig
- Nachteil: geschwätzig
- Wurzel-Element: schema aus W3C-Namensraum http://www.w3.org/2001/XMLSchema
- hier XML-Schema für XML-Schema hinterlegt: **Schema der Schemata**

Geschwätzigkeit von XML-Schemata

- PurchaseOrder.xml verweist auf cXML-Schema, das Client von anderen Server laden muss.
- cXML
  - als Schema: 50KB
  - als DTD: 15KB
- XML-Schemata aber besser zu komprimieren:
  - cXML-Schema: 6KB
  - cXML-DTD: 4KB



Ziel-Namensraum

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

- jedes XML-Schema definiert bestimmtes Vokabular (Elemente und Attribute).
- Dieses Vokabular wird einem Namensraum zugeordnet: **Ziel-Namensraum** (*target namespace*).
- Ziel-Namensraum wird wie jeder Namensraum mit URI identifiziert
- definiertes Vokabular kann über URI identifiziert werden.

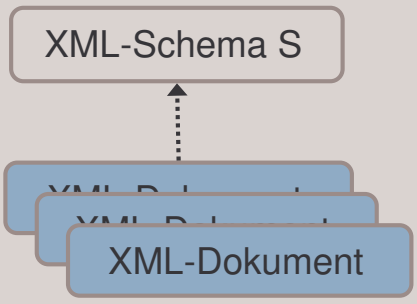
Ziel-Namensraum

DTD definiert *keinen* Namensraum.

http://www.books.org

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

Instanz eines XML-Schemas



Instanzen von S

Instanz eines XML-Schemas S ist ein XML-Dokument, das

1. dem Ziel-Namensraum von S zugeordnet ist und
2. gültig (valid) bzgl. S ist.

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 aus diesem XML-Dokument 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">
      …
</BookStore>
```

Ziel-Namensraum des XML-Schemas

- Wurzel-Element und Namensraum legen zusammen den Dokument-Typ fest.
- Wurzel-Element muss in XML-Schema global deklariert sein.
- Für Namensräume wie http://www.w3.org/1999/xhtml keine weiteren Schritte nötig.

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
                               http://www.books.org/BookStore.xsd">
        …
</BookStore>
```

kann auch lokale Datei wie z.B. BookStore.xsd sein

- Attribut schemaLocation gibt Hinweis, wo entsprechendes XML-Schema zu finden ist.
- Beachte: XML-Parser darf diese Information ignorieren und anderes XML-Schema berücksichtigen!

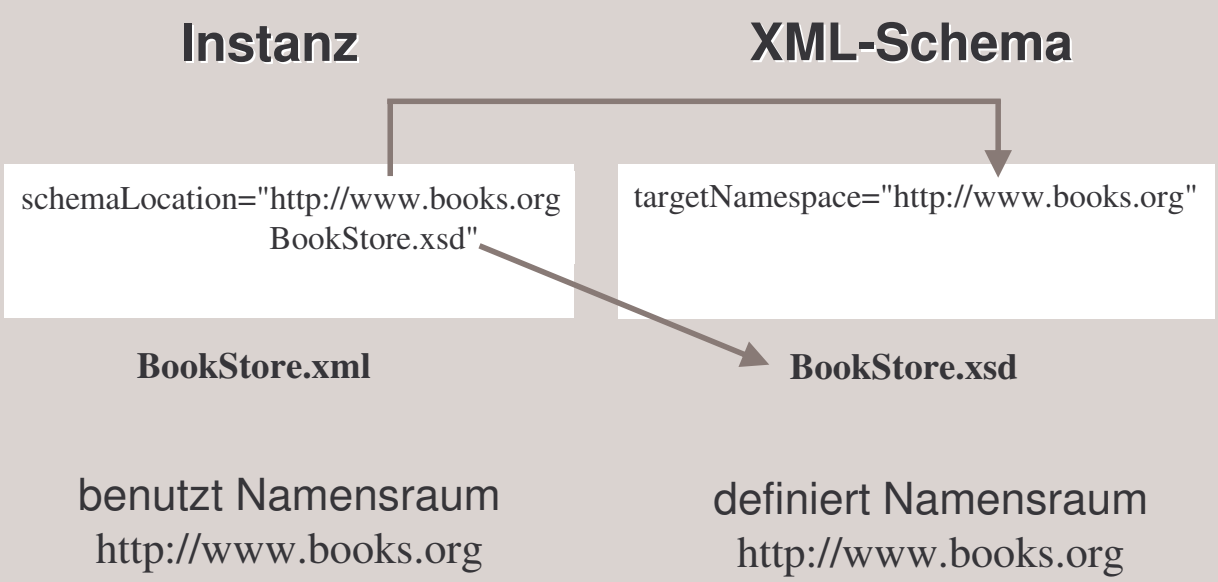
3. Schritt

```
<?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
                               http://www.books.org/BookStore.xsd">
        …
</BookStore>
```

- Attribut schemaLocation stammt aus dem W3C-Namensraum für Schema-Instanzen.

- weiteres Beispiel für die Erweiterbarkeit von XML!
- XML durch Attribut schemaLocation erweitert

Instanz und Schema



benutzt Namensraum
http://www.books.org

definiert Namensraum
http://www.books.org

Software zum Validieren von Instanzen

- xerces by Apache (API)
  - ➔ http://xml.apache.org/
- MSXML (API)
  - ➔ http://www.microsoft.com
- XML Spy (GUI)
  - ➔ http://www.altova.com/
- weitere: http://www.w3.org/XML/Schema#Tools

Validierung auf mehrere Ebenen

Instanz
= XML-Dokument

XML-Schema

BookStore.xml

BookStore.xsd

zulässiges BookStore-Dokument?

Validierung auf mehrere Ebenen

Instanz
= XML-Dokument

XML-Schema
= XML-Dokument

Schema der Schemata

BookStore.xml

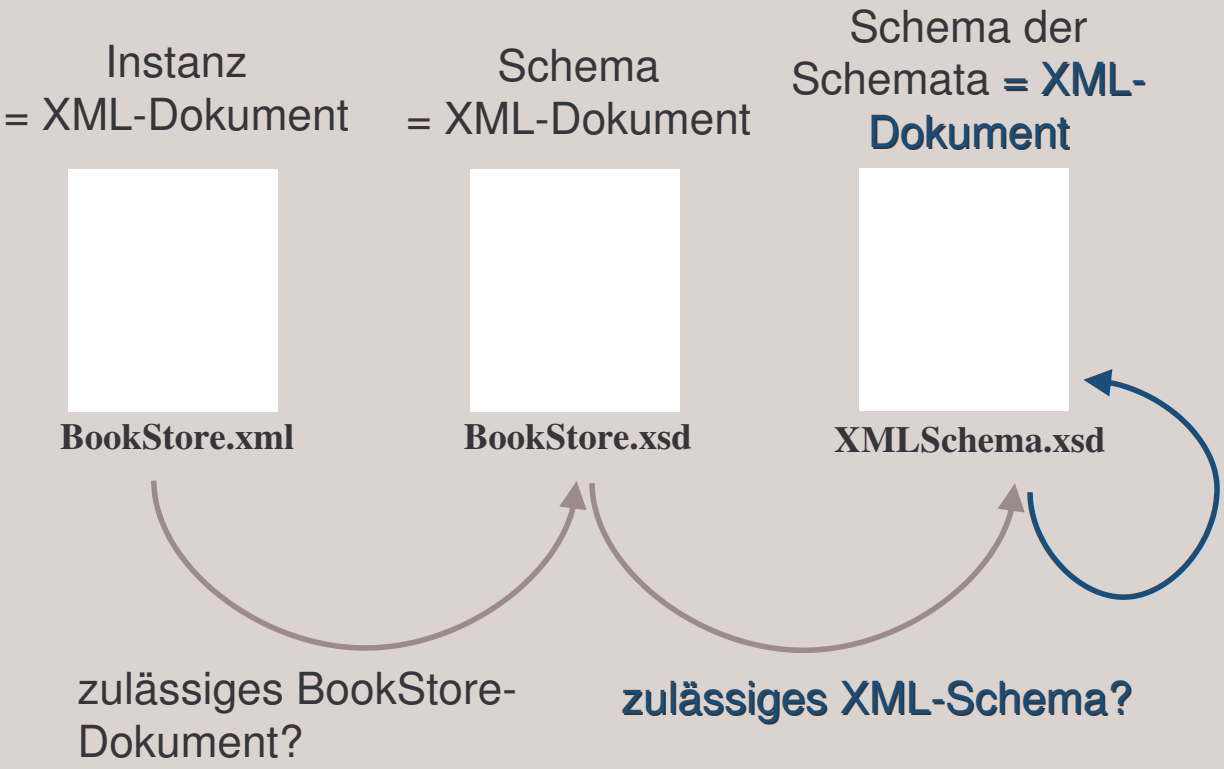
BookStore.xsd

XMLSchema.xsd

zulässiges BookStore-Dokument?

zulässiges XML-Schema?

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>
```

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

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

<!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>
```

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

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

- xsd:string: vordefinierter Datentyp

<!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>
```

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

- xsd:date: vordefinierter Datentyp

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 hier ein **benannter Datentyp** (*named type*)
- auch als **globale Typ-Definition** bezeichnet

Alternative: 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 Typ-Definition** bezeichnet
- Nachteil: kann an anderer Stelle *nicht* wieder verwendet werden

<!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>
```

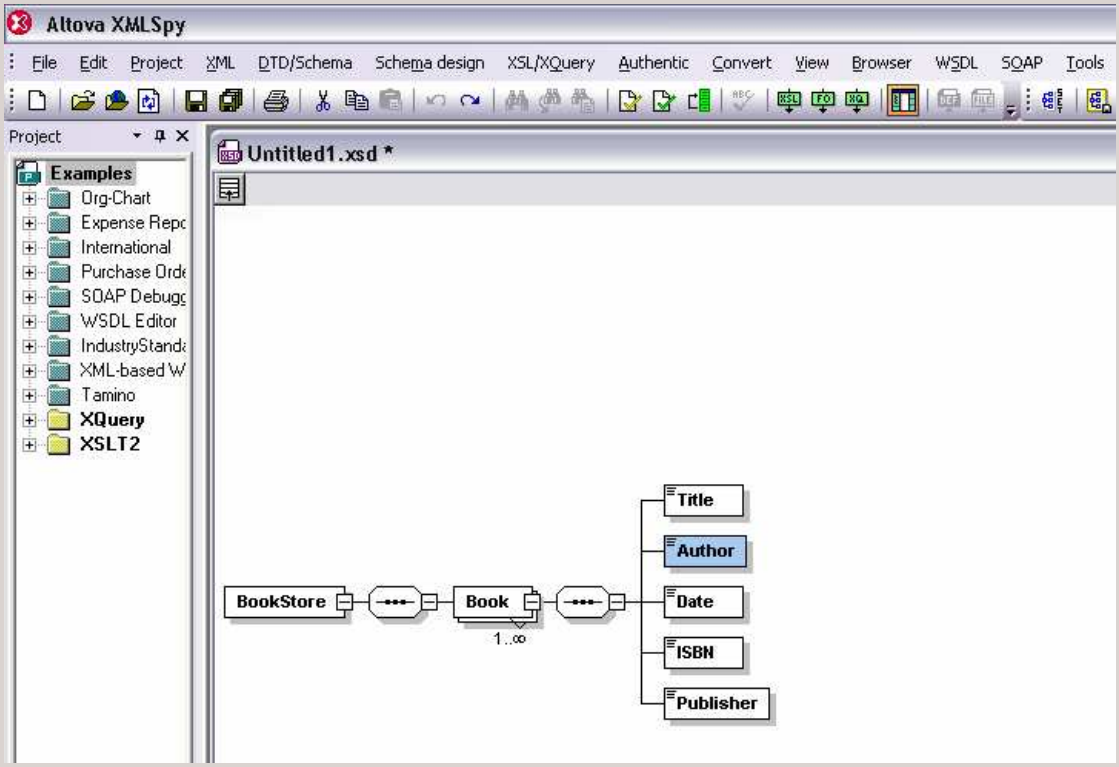
- Jedes Elemente erscheint in der Sequenz so häufig, wie mit minOccurs und maxOccurs festgelegt.

Das vollständige XML-Schema

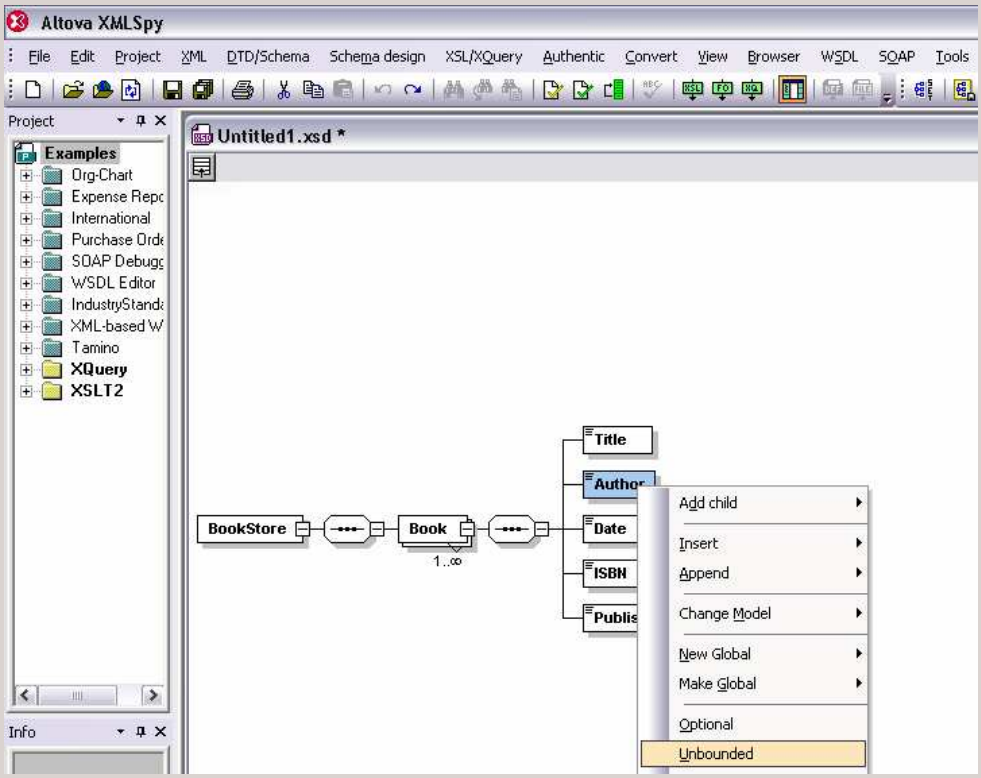
```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
                targetNamespace="http://www.books.org">
  <xsd:element name="BookStore">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="Book"  maxOccurs="unbounded">
          <xsd:complexType>
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

Lernziel: Dieses Schema und die entsprechende DTD verstehen!

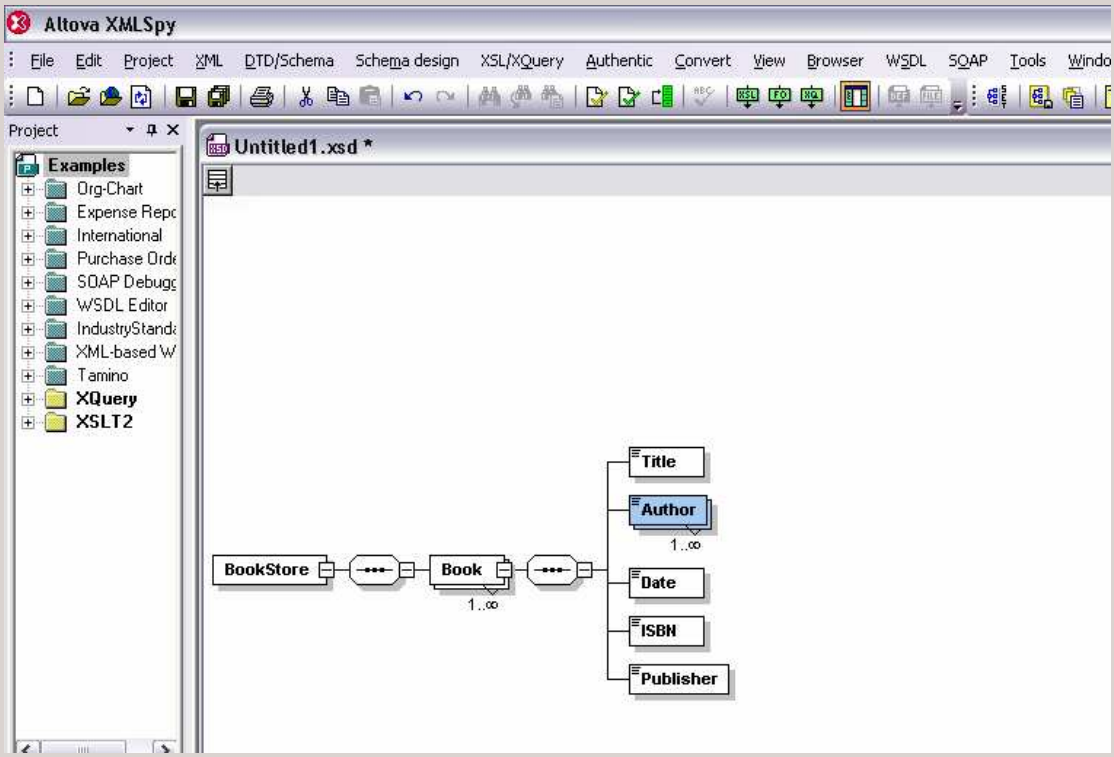
XML Spy: Visualisierung



XML Spy: Editieren



XML Spy: Editieren



Wie geht es weiter?

heute

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

nächste Woche

- XML-Schema im Detail