XML

Zusammenfassung

Grundbegriffe

Elemente

Start-Tag: `<person>`
End-Tag: `</person>`
Jedes Start-Tag braucht ein entsprechendes End-Tag.

Element mit Attribut

```
<Name Vorname="Markus" Nachname="Rigling"></Name>
```

Dieser Code hat ein leeres Element `Name`. Das Element Name hat zwei Attribute `Vorname` und `Nachname`. Attribute **müssen** Werte enthalten. Die Attribut-Werte sind immer ein string.

Kommentar

```
<!-- Das ist ein Kommentar -->
```

Ein Kommentar beginnt in XML mit `<!--` und endet mit `-->`.

CDATA-Abschnitte

Da in einem XML Dokument die Zeichen `<` und `&` als `<` und `&` geschrieben werden müssen, kann ein Text recht unübersichtlich werden. Als Lösung bietet sich ein CDATA-Abschnitt an:

```
<![CDATA[ Hier darf man die Zeichen < und & ohne böses
Erwachen schreiben! ]]>
```

Processing Instructions

Mit Processing Instructions kann man der Applikation, die das XML-Dokument liest, Informationen mitteilen. Eine sehr bekannte Anwendung des Processing Instruction befindet sich in der 1. Zeile des XML-Dokuments:

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

XML-Deklaration

Ein XML-Dokument hat normalerweise die Dateiendung `.xml`. Die Deklaration:

```
<?XML version="1.0" encoding="ASCII" standalone="yes"?>
```

`encoding`:	Was für einen Zeichensatz verwendet wird. Z.B. `ASCII`, `ISO-8859-1`, ...
`standalone`:	`standalone="no"` sagt, dass die XML-Applikation eine DTD oder ein Schema braucht. `Standalone="yes"` sagt, dass die XML-Applikation alleine (ohne DTD / Schema) zurecht kommt.

DTD – Document Type Definition

DTDs beschreiben, welche Elemente und Entitiäten im XML-Dokument vorkommen dürfen und was für einen Inhalt die Elemente und Attribute haben dürfen wie „Jedes Angestellten-Element muss eine AHV-Nummer enthalten".
Beim Parsen des XML-Dokuments mit DTD-„Einschränkungen" kann deshalb ein *validity error* auftauchen, auch wenn die Syntax XML-konform (*well formed*) ist.

Was ein DTD nicht sagt

- Was das Root-Element des Dokumentes ist
- Wie viele Instanzen jedes Element haben darf
- Wie die Daten im Element auszusehen haben
- Was das Element für Daten enthält (z.B. ist es ein Datum oder ein Personenname?)

Ein einfaches DTD

```
<!ELEMENT person     (name, profession*)>
<!ELEMENT name       (first_name, last_name)>
<!ELEMENT first_name (#PCDATA)>
<!ELEMENT last_name  (#PCDATA)>
<!ELEMENT profession (#PCDATA)>
```

Diese DTD beschreibt ein Dokument, das ein Element `person` enthält, das wiederum die Elemente `name` und beliebig viele Elemente `profession` enthält. Das Element Name enthält ein Element `first_name` sowie ein Element `last_name`. `First_name`, `last_name` und `profession` enthalten als Werte normaler Text (PCDATA = parsed character data; Text mit & aber keine Elemente oder tags).

DTD-Deklaration:

DTD ist in `C:\XML\DTDs\file.dtd`

file.xml

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE person SYSTEM „C:\XML\DTDs\file.dtd">
...
```

Falls die DTD sich im Internet befindet kann auch entsprechende Internetadresse wie `http://www.rigling.net/dtd/file.dtd` angegeben werden.
Das Root-Element des XML-Dokuments ist `person` und die DTD kann mit der URI `C:\XML\DTDs\file.dtd` gefunden werden. Eine URI ist ein Uniform Resource Identifier. Das `standalone`-Attribut in der XML-Datei muss als Wert `no` enthalten.

DTD direkt im XML-Dokument schreiben:

Anstelle für die DTD eine eigene Datei zu verwenden (dies wird allerdings empfohlen) kann die DTD auch im XML-Dokument integriert sein. Dies kann aber schnell unübersichtlich werden. Das XML-Dokument mit DTD würde so aussehen:

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE person [
     <!ELEMENT person    (name, profession*)>
     <!ELEMENT name (first_name, last_name)>
     <!ELEMENT first_name (#PCDATA)>
     <!ELEMENT last_name (#PCDATA)>
     <!ELEMENT profession (#PCDATA)>
]>
<!-- Hier fängt das XML-Dokument wirklich an...-->
...
```

Elementdeklaration

`<!ELEMENT` *`name`* `(`*`inhalt`*`)>`

`name`: Name des Elements

`inhalt`: Name und Reihenfolge der Child-Elemente
(z.B. `<!ELEMENT name (first_name, last_name)>`)
Spezifikation der einzelnen Child-Elemente
(z.B `<!ELEMENT last_name (#PCDATA)>`)

Zur Wichtigkeit der Reihenfolge

- Elementdeklaration

```
<!ELEMENT student (name, class)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT class (#PCDATA)>
```

- Gültig:

```
<student>
     <name> dupont </name>
     <class> 1998 </class>
</student>
```

- Ungültig:

```
<student>
   <class> 1998 </class>
   <name> dupont </name>
</student>
```

- Ungültig:

```
<student>
   <name> dupont </name>
</student>
```

- Ungültig:

```
<student>
   <name> dupont </name>
   <vorname> jean </vorname>
   <class> 1908 </class>
</student>
```

Anzahl der Elemente:

Die mögliche Anzahl der Instanzen des Elements wird jeweils hinter das Element geschrieben:

```
<!ELEMENT person (name, profession*)>
```

?: 0 oder 1 Element ist erlaubt
*: beliebig viele Elemente sind erlaubt (von 0 bis ???)
+: mindestens ein Element (von 1 bis ???)

Auswahl

```
<!ELEMENT Zahlungsbetrag( Kreditkarte | Bar )>
<!ELEMENT name( (vorname, mittelname, nachname) | (vorname, nachname) )>
<!ELEMENT text( #PCDATA | fett | kursiv)*>
```

Das Element Zahlungsbetrag hat entweder das Element `Kreditkarte` oder das Element `Bar`.
Die Elemente können auch mit Hilfe von Klammern gruppiert werden.
Im 3. Beispiel kann das Element `text` sowohl normale Zeichen sowie Elemente `fett` und Elemente `kursiv` haben.
Das Element #PCDATA muss dabei immer an erster Stelle stehen.

Leeres Element

```
<!ELEMENT BR EMPTY>
```

Das Element `BR` darf keine Elemente besitzen.

ANY

```
<!ELEMENT Weltall ANY>
```

Das Element Weltall darf beliebig viele Elemente mit beliebigem Namen besitzen.

Attributdeklaration

Attribute werden mit einer `ATTLIST`-Deklaration deklariert.

```
<!ELEMENT book EMPTY>
<!ATTLIST book author CDATA "Markus Rigling"
               editor CDATA "Editout">
```

Das Element `book` darf keine Elemente besitzen. Es hat 2 Attribute `author` und `editor`, die als Vorgabewert `"Markus Rigling"` und `"Editout"` haben.
CDATA ist der Attributstyp.

Es gibt 10 Attributtypen in XML:

`CDATA` (Text-String)	`ENTITIES`
`NMTOKEN`	`ID`
`NMTOKENS`	`IDREF`
`Enumeration`	`IDREFS`
`ENTITY`	`NOTATION`

siehe Buch *XML in a Nutshell* S. 42 oder S. 350

Attributeigenschaften:

```
<!ATTLIST AUTHOR NAME CDATA #FIXED "Dupont">
```
Das Attribut ist konstant und kann nicht verändert werden.

```
<!ATTLIST AUTHOR NAME CDATA #IMPLIED>
```
Das Attribut ist optional.

```
<!ATTLIST AUTHOR NAME CDATA #REQUIRED>
```
Das Attribut muss bestehen.

Entitätsreferenzen (x ersetzen durch y)

```
<!ENTITY name "Ersetzungstext">
```

Ersetzt im XML-Dokument `&name;` durch `Ersetzungstext`.
siehe Buch *XML in a Nutshell* S. 48 (General Entity Declarations)

Notation

Eine Notation beschreibt das Format eines ungeparsten Dokuments.

```
<!NOTATION gif SYSTEM "image/gif">
```

`image/gif` hat ein Zeichen (`/`), das nicht kompatibel in XML ist, deshalb hat man hier eine Notation gemacht.
siehe Buch *XML in a Nutshell* S. 51

Parameter Entitäten

Falls mehrere Elemente die genau gleich lauten Elemente besitzt, können diese zusammengefasst werden.

siehe Buch *XML in a Nutshell* S. 53

IGNORE / INCLUDE

Ein Parser wird folgende Deklaration von einem Produktion Element ignorieren:

```
<![IGNORE[
     <!ELEMENT Produktion (#PCDATA)>
[[>
```

Gegenteil: `INCLUDE` Sagt, dass die aktuelle Deklaration in der DTD benutzt wird.

Namensräume

Namensräume sind dazu da, um gleichnamige Elemente und Attribute verschiedener XML-Applikationen zu unterscheiden.
Verwandte Elemente und Attribute einer XML-Applikation können gruppiert werden.

Syntax

Präfix : *lokaler_Teil*

z.B. `rdf:description`

URI eines Namensraums

Jedem Präfix muss eine URI zugeordnet werden:

```
<rdf:RDF xmlns:rdf= "http://www.w3c.org/TR/REC-rdf-syntax#">
     <rdf:Description>
     ...
     </rdf:Description>
</rdf:RDF>
```

Die erste Applikation ist im Namensraum `rdf`. Ihr Root-Element heisst `RDF`. Das Element `Description` ist im Namespace `rdf`.

- In einem Element können mehrere Namespaces deklariert werden.
- Ein Präfix kann verschiedenen Namespaces entsprechen.

Der Standort der URI muss nicht existieren, da die URI nur für die Einmaligkeit eines Namensraumes steht. So ist `http://www.w3c.org/TR/REC-rdf-syntax#` ein Standard-Namensraum für das Namespace `rdf`.

Präfixe, die mit `xml` beginnen, sind reserviert und dürfen nicht verwendet werden.

Namensräume und DTD

Die Namen eines Elements oder eines Attributs im XML-Dokument und in der DTD müssen genau dieselben sein. => Änderung eines Präfix muss überall berücksichtigt werden!!

Z.B.: `<!ELEMENT dc:Titel (#PCDATA)>`

Das Element Titel muss auch im XML-Dokument im Namespace `dc` sein!
Parameter Entity References für Namespace Prefixes

```
<!ENTITY % dc-prefix "dc">
<!ENTITY % dc-colon ":">

<!ENTITY % dc-title "%dc-prefix;%dc-colon;title">
<!ENTITY % dc-description "%dc-prefix;%dc-colon;description">
```

In der ersten und zweiten Zeile definiere ich `dc` als `dc-prefix` und `:` als `dc-colon`.
In der dritten und vierten Zeile kann ich diese Definition mit `%dc-prefix;` bzw. mit `%dc-colon;` verwenden.

Default Namespace

Falls man alle Elemente, die keinem Namespace zugehören in ein Standard-Namespace legen will, so gibt es die Möglichkeit, mit dem xmlns-Attribut ein Standard-Namespace anzulegen.
Dazu hat man dem xmlns-Attribut kein Prefix anzuhängen:

```
<svg xmlns="http://www.w3.org/2000/svg"> </svg>
```

Cascading Style Sheets CSS

Ein CSS-Dokument beschreibt das Aussehen eines XML-Dokuments. Damit das XML-Dokument weiss, dass es ein CSS-Dokument gibt und wo sich dieses befindet, muss das CSS-Dokument im XML-Dokument "referenziert" werden.

CSS in XML-Dokument einbinden

```
<?xml version="1.0" encoding="UTF-8" standaloe="yes"?>
<?xml-stylesheet type="text/css" href="recipe.css"?>
...
```

recipe.css befindet sich im selben Verzeichnis wie das XML-Dokument.

Syntax

CSS ist eine Liste von den Elemente, die dem Aussehen des XML-Dokuments hinzugefügt werden sollen. Einige Beispiele befinden sich im Buch *XML in a Nutshell* S. 205.

Bei der Deklaration kommt es nicht auf die Gross-Kleinschreibung an. So ist FONT-FAMILY das gleiche wie Font-Family.

Falls ein Aussehen für alle Elemente gilt kann mit Hilfe eines Sterns deklariert werden.

```
* {font-size:large}
```

```
zutaten menge {font-size:medium}
```
Alle `menge`-Elemente sind Nachkommen von `zutaten`-Elemente, also auch Kind von Kind von Element.

```
zutaten > menge {font-size: inherit}
```
Das Element `menge` muss direkter Nachfolger von `zutaten` sein. Kind von Kind von Element sind nicht mehr betroffen.

```
directions + story {border-top-style: solid}
```
Die Elemente `directions` und `story` sind *sibling*-Elemente (Schwester-Elemente, d.h. keines ist child von anderem). Sie müssen im XML-Dokument direkt nebeneinander sein.

Weiteres im Buch *XML in a Nutshell* S. 205ff.

XML Schemas

Ein XML-Schema ist ein Dokument, das den Inhalt eines gültigen XML-Dokument beschreibt (wie DTD).
Ein Instanz-Dokument ist ein Dokument, welches durch ein XML-Schema beschrieben wird.

XML-Schemas und DTD

Vorteile von XML-Schemas:

- Unterstützung von Datentypen
- Unterstützung von Namensräumen
- Typen-Ableitung und -Vererbung

XML-Schemas ersetzen keine DTDs, sondern ergänzen sie. Ein Dokument kann gleichzeitig eine DTD und ein XML-Schema haben.

XML-Schema in XML-Dokument einbinden

file.xml

```
<?xml version="1.0?>
<Text xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:noNamespaceSchemaLocation="schema.xsd">
</Text>
```

schema.xsd

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
     <xs:element name="Text" type="xs:string"/>
</xs:schema>
```

Das Rootelement ist `xs:schema`. Die Aufzeichnungselemente eines XML-Schemas gehören zum Namensraum `http://www.w3.org/2001/XMLSchema`. Dieser wird mit dem Präfig `xs` oder `xsd` verbunden. Es wird ein `Text`-Element generiert, das den Typ `string` haben muss. `string` befindet sich in der URI und wird deshalb zu `xs:string`.

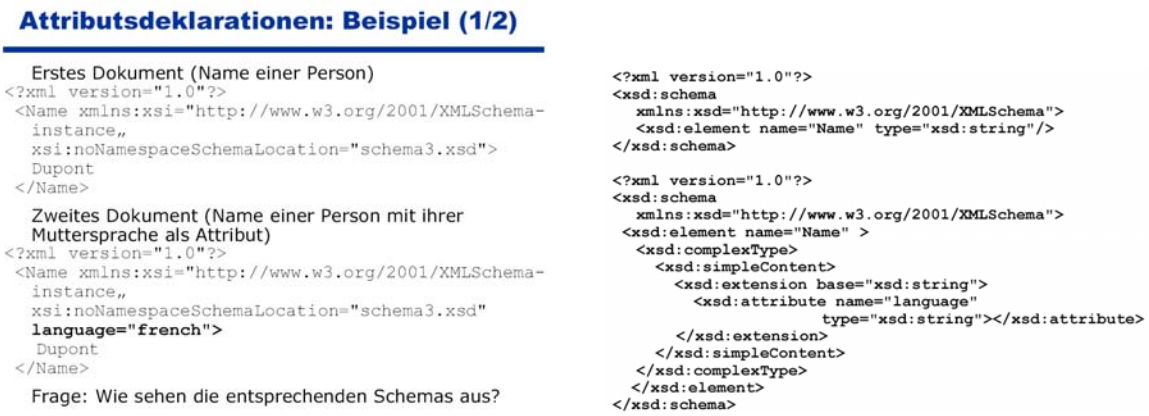
XML-Dokument: Das Attribut `xsi:noNamespaceSchemaLocation` wird dem ersten Element hinzugefügt, auf das das Schema angewendet werden soll. Dies ist meistens das Root-Element. xsi wird auf `http://www.w3.org/2001/XMLSchema` abgebildet.

Element-Deklaration

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

Attribut-Deklaration

Attribute werde mit dem `xs:attribute`-Element deklariert
Hierzu ein paar Beispiele aus den Folien:



Attributsdeklarationen: Beispiel (1/2)

Erstes Dokument (Name einer Person)

```
<?xml version="1.0"?>
<Name xmlns:xsi="http://www.w3.org/2001/XMLSchema-
  instance„
  xsi:noNamespaceSchemaLocation="schema3.xsd">
  Dupont
</Name>
```

Zweites Dokument (Name einer Person mit ihrer Muttersprache als Attribut)

```
<?xml version="1.0"?>
<Name xmlns:xsi="http://www.w3.org/2001/XMLSchema-
  instance„
  xsi:noNamespaceSchemaLocation="schema3.xsd"
  language="french">
  Dupont
</Name>
```

Frage: Wie sehen die entsprechenden Schemas aus?

```
<?xml version="1.0"?>
<xsd:schema
   xmlns:xsd="http://www.w3.org/2001/XMLSchema">
   <xsd:element name="Name" type="xsd:string"/>
</xsd:schema>

<?xml version="1.0"?>
<xsd:schema
   xmlns:xsd="http://www.w3.org/2001/XMLSchema">
 <xsd:element name="Name" >
   <xsd:complexType>
     <xsd:simpleContent>
       <xsd:extension base="xsd:string">
         <xsd:attribute name="language"
                        type="xsd:string"></xsd:attribute>
       </xsd:extension>
     </xsd:simpleContent>
   </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

Element Name ist ein einfacher Typ. Einfache Typen dürfen kein Element enthalten.

XML-Schema mit Namensräume

file.xsd

```
<xsd:schema
targetNamespace="http://ibiblio.org/xml/namespace/song"
xmlns="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified">
```

`targetNamespace`: URI des Namespaces der XML-Anwendung, welche durch dieses Schema beschrieben ist.
`elementFormDefault`: ist `qualified`, wenn sich die beschriebenen Elemente in einem `targetNamespace`-Namensraum befinden oder `unqualified`, wenn sich die beschriebenen Elemente nicht in einem `targetNamespace`-Namensraum befinden.
Globale Elemente sind immer im `targetNamespace`-Namensraum.
Wie im "normalen" Namespace, überprüft der Parser die URI und nicht der Präfix. Falls der Präfix verschieden ist, aber die URI gleich so stimmen die beiden Namespaces überein.

minOccurs / maxOccurs

Beschreiben die minimale/maximale Anzahl von Instanzen eines Elements. `minOccurs` / `maxOccurs` hat als Wert eine natürliche Zahl oder `unbounded`. Der Default-Wert von minOccurs / maxOccurs ist 1.

```
<xsd:element name="TITLE" type="xsd:string" minOccurs="1"
  maxOccurs="1"/>
<xsd:element name="ARTIST" type="xsd:string" minOccurs="1"
  maxOccurs="unbounded"/>
```

Typen

mit dem Attribut type kann in der Elementbeschreibung der gewünschte Typ angegeben werden. Es gibt 44 vordefinierte Typen.

- Numerische Typen (`xs:float, xs:integer, xs:double,...`)
- Zeittypen (`xs:timeInstant, xs:gYear,...`)
- XML-Typen
- Stringtypen (`xs:string, ...`)
- Boolescher Typ (`xs:boolean`)
- URI-Referent-Typ

Weiteres im Buch *XML in a Nutshell* S. 396.

Neue Typen kreieren

Neue einfache Typen können mit der Syntax `<xs:simpleType>` definiert werden. So kann man aus mehreren einfachen Typen einen eigenen `simpleType` zusammenstellen. Es gibt drei Methoden, um neue einfache Typen von existierenden einfachen Typen abzuleiten:

Einschränkung:

```
<xs:simpleType name="NewString">
     <xs:restriction base = "xs:int"/>
</xs:simpleType>
```

`NewString` ist gleich wie `int`

Vereinigung:

```
<xs:simpleType name="YearOrInteger">
     <xs:union memberTypes="xs:gYear xs:int"/>
</xs:simpleType>
```

Datentyp von `YearOrInteger` ist entweder `gYear` oder `int`

Liste:

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

`listType` ist wie Liste von String

Facetten

Schemas können einfach die möglichen Werte von Einfachen Typen durch Facetten prüfen. Facetten sind Betrachtungsweisen eines einfachen Datentyps.
Facetten werden durch `xs:restriction` zu den einfachen Typen hinzugefügt.

Es gibt folgende Facetten:

- whitespace
- length (minLength, maxLength)
- pattern
- enumeration
- maxInclusive, maxExclusive
- minInclusive, minExclusive
- totalDigits
- fractionDigits

Beispiel Facette mit XML-Dokument gültig / nicht gültig

schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
      <xs:element name="Kurzeichen" type="Abbreviation"/>
      <xs:simpleType name="Abbreviation">
            <xs:restriction base="xs:string">
                  <xs:maxLength value="8"/>
            </xs:restriction>
      </xs:simpleType>
</xs:schema>
```

gültig

```
<?xml version="1.0" encoding="UTF-8"?>
<Kurzeichen xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="ex54.xsd">12345678</Kurzeichen>
```

ungültig

```
<?xml version="1.0" encoding="UTF-8"?>
<Kurzeichen xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="ex54.xsd">
  12345678 </Kurzeichen>
```

Pattern Value

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="Betrag" type="money"/>
  <xsd:simpleType name="money">
    <xsd:restriction base="xsd:string">
    <xsd:whiteSpace value="collapse"/>
    <xsd:pattern value="\p{Sc}\p{Nd}+(\.\p{Nd}\p{Nd})?"/>
   </xsd:restriction>
  </xsd:simpleType>
</xsd:schema>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<Betrag xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="ex55.xsd"> € 123.56 </Betrag>
```

`\p{Sc}` Währungszeichen (€)
`\p{Nd}+` mindestens ein integer (123)

`\.` Punkt (.)
`\p{Nd}` integer (56)
Punkt und die letzten beiden Integer `\p{Nd}` sind ausgeklammert und mit einem ? versehen. D.h. dass es diese Zeichenfolge gibt (z.B. € 5.99) oder es gibt sie nicht (z.B. € 5).
`\d` Digit (ein Zeichen)

Buch *XML in a Nutshell* S. 388-390

Komplexe Typen

- Ein komplexer Typ wird durch das `xs:complexTyp`-Element definiert.
- Elemente von einem komplexen Typ haben Attribute und/oder Child-Elemente
- Attribute haben immer einen einfachen Typ
- Ein komplexer Typ, welcher durch einen Top-Element deklariert ist, muss einen Namen haben und kann referenziert werden
- Ein komplexer Typ, der innerhalb einer Element-Deklaration deklariert ist, braucht keinen Namen zu haben (anonymous Typ)
- Der komplexe Typ eines Elements beschreibt den Inhalt dieses Elements

Beispiel eines komplexen Typs
```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
   <xsd:element name="SONG" type="SongType"/>
   <xsd:complexType name="SongType">
       <xsd:sequence>
          <xsd:element name="TITLE" type="xsd:string"/>
          <xsd:element name="COMPOSER"   type="xsd:string"/>
          <xsd:element name="PRODUCER"   type="xsd:string"/>
          <xsd:element name="PUBLISHER"  type="xsd:string"/>
          <xsd:element name="LENGTH"     type="xsd:string"/>
          <xsd:element name="YEAR"       type="xsd:string"/>
```

Zuerst wird ein Element SONG als `complexType` erstellt (wie eine Klasse). Danach wird in einer Sequenz die gewünschten Elemente eingefügt und deren Typ angegeben (wie Attribute).

Anonymer Typ

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
   xsd:element name = "SONG">
   <xsd:complexType>
      <xsd:sequence>
         <xsd:element name="TITLE"       type="xsd:string"/>
         <xsd:element name="COMPOSER"    type="xsd:string"/>
      <xsd:sequence>
   </xsd:complexType>
</xsd:element>
</xsd:schema>
```

Elementinhalt

Vier Möglichkeiten:

- Leerer Inhalt
- Einfacher Inhalt: nur Text (parsed character data)
- Gemischter Inhalt: Text und Elemente
- beliebiger Inhalt

Leeres Element

file.xml

```
<phoneNumber number="055 212 21 60"/>
```

file.xsd

```
<xs:element name="phoneNumber">
   <xs:complexType>
      <xs:attribute name="number" type="xs:string"/>
   </xs:complexType>
</xs:element>
```

Einfacher Inhalt

Elemente mit einfachem Inhalt sind entweder von einem einfachen Typ oder von einem komplexen Typ mit einem `simpleContent`-Element.

file.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<Name xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="simpleElement.xsd">
   Jean Dupont
</name>
```

simpleElement.xsd

```
<?xml version=1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
   <xs:element name="Name">
      <xs:complexType>
         <xs:simpleContent/>
      </xs:complexType>
   </xs:element>
</xs:schema>
```

Gemischter Inhalt

Der Inhalt eines gemischten Elements darf Text und untergeordnete Elemente enthalen, wenn sein Typ komplex ist und das Attribut `mixed` des entsprechenden `complexTyp`-Elements den Wert `true` hat. Die Position der untergeordneten Elemente ist durch die Elemente `xs:sequence`, `xs:choice` oder `xs:all` festgelegt. Diese 3 Elemente werden Gruppen genannt.

`xs:all`-Gruppe

> Alle Elemente treten höchstens einmal auf, wobei die Reihenfolge unwichtig ist.

`xs:choice`-Gruppe

> Ein beliebiges Element aus der Gruppe darf auftreten.

`xs:sequence`-Gruppe

> Alle Elemente treten genau einmal in der angegebenen Reihenfolge auf.

Setzen von `mixed="true"` ist nicht das selbe wie ein Element zu deklarieren das ein string enthält. (Seite 275)

Siehe auch Folien Schemas Seite 41

Beliebiger Inhalt

```
<xs:element name="notes">
   <xs:complexType>
      <xs:sequence>
         <xs:any namespace="http://www.w3.org/1999/xhtml"
         minOccurs="0" maxOccurs="unbounded"
         processContents="skip">
         </xs:any>
      </xs:sequence>
   </xs:complexType>
</xs:element>
```

Herleitung komplexer Typen

Ableitung durch Erweiterung

- Der neue Typ wird vom alten Typ durch Anhängen von zusätzlichen Elementen abgeleitet
- Nur die neuen Elemente sind zu deklarieren

alt

```
<xs:complexType name="Adresse">
  <xs:sequence>
    <xs:element name="Name" type="xs:string"/>
    <xs:element name="Strasse" type="xs:string"/>
    <xs:element name="Postleitzahl" type="xs:string"/>
    <xs:element name="Stadt" type="xs:string"/>
</xs:sequence>
```

neu

```
<xs:complexType name="LangAdresse">
  <xs:complexContent>
    <xs:extension base="Adresse">
      <xs:sequence>
        <xs:element name="TelefonNr" type="xs:string"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

```
<xs:complexContent>
<xs:extension base = "Adresse">
<xs:sequence>
<xs:element name="TelefonNr" type="xs:string"/>
...
```

Die `extension base` muss immmer im `complexContent`-Block liegen. Basistyp `Adresse` wird mit Attribut `TelefonNr` zu `LangAdresse` erweitert.

Ableitung durch Einschränkung

- Der neue Typ wird durch Auslassung von Teilen des alten Typs abgeleitet
- Die Teile des alten Typs, die im neuen Typ übernommen werden, müssen neu deklariert werden

alt

```
<xs:complexType name="Adresse">
  <xs:sequence>
    <xs:element name="Name" type="xs:string"/>
    <xs:element name="Strasse" type="xs:string"/>
    <xs:element name="Postleitzahl" type="xs:string"/>
    <xs:element name="Stadt" type="xs:string"/>
</xs:sequence>
```

neu

```
<xs:complexType name="KurzAdresse">
  <xs:complexContent>
    <xs:restriction base="Adresse">
      <xs:sequence>
        <xs:element name="Name" type="xs:string"/>
        <xs:element name="Strasse" type="xs:string"/>
      </xs:sequence>
    </xs:restriction>
  </xs:complexContent>
</xs:complexType>
```