

XML-Grundlagen

"In 1956 he [Malcom McLean] used a converted World War II oil tanker to sail 58 cargo-filled containers from Port Newark in New Jersey to Houston. [...] Ninety percent of all cargo is now transported by container."

(International Herald Tribune, May 28, 2001)

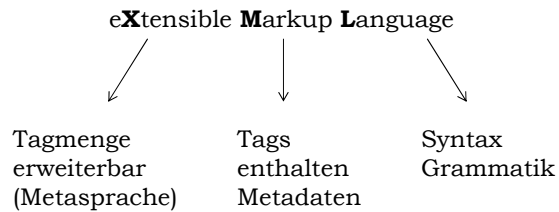
Einführung

Inhalt

- Was ist XML?
- XML vs HTML
- Geschichte
- Anwendungen
- Literatur

Was ist XML?

XML wurde 1998 von einer Arbeitsgruppe des World Wide Web Consortium (W3C) definiert, um Informationen im World Wide Web bereitzustellen.



HTML-Beispiel

```
<html>
  <h1>People</h1>
  <table border="1">
    <tr>
      <th>Name</th>
      <th>Profession</th>
    </tr>
    <tr>
      <td>Alan Turing</td>
      <td>computer scientist</td>
    </tr>
    <tr>
      <td>Richard M. Feynman</td>
      <td>physicist</td>
    </tr>
  </table>
</html>
```

XML-Beispiel

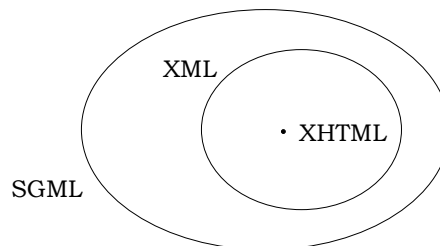
```
<?xml version="1.0"?>
<!-- file people.xml -->

<people>
  <person>
    <name>Alan Turing</name>
    <profession>computer scientist</profession>
  </person>
  <person>
    <name>Richard M. Feynman</name>
    <profession>physicist</profession>
  </person>
</people>
```

XML vs HTML

HTML: endliche Menge von Formatierungselementen wie
Überschriften, Absätze, Listen, Tabellen usw.
→ Darstellungssprache

XML: erweiterbare Menge struktureller und inhaltlicher
Elemente
→ allgemeine Datenbeschreibungssprache



Geschichte

- 1969 Goldfarb, Moster, Lorie entwickeln bei IBM die Generalized Markup Language (GML)
- 1986 SGML (Standardized GML) als ISO-Standard verabschiedet
- 1993 HTML 1.0 als IETF-Entwurf
- 1994 Gründung des World Wide Web Consortium (W3C)
- 1996 Jon Bosak et. al. entwerfen XML
- 1998 XML 1.0 als W3C-Recommendation veröffentlicht
- 1999 Namespaces, XSL Transformations, XPath
- 2000 DOM Level 2, XHTML
- 2001 XML Schema, XLink, XSL
- 2002 XML-Signature

Standard-Anwendungen

- Mathematical Markup Language (MathML)
- Resource Description Framework (RDF)
- Extensible HyperText Markup Language (XHTML)
- Scalable Vector Graphics (SVG)
- Chemical Markup Language (CML)
- usw.

Allgemeine Anwendungen

- Erstellen und Verarbeiten von Textdokumenten
- Elektronischer Datenaustausch zwischen Firmen
- Konfigurationsfiles von Software
- Deklarative Programmierung
- Definition von Netzwerkprotokollen
- Schnittstelle zu Datenbanken
- usw.

Literatur

- W3C Recommendations (www.w3.org):
 - Extensible Markup Language (XML) 1.0
 - Namespaces in XML
 - XML Schema
- Bücher:
 - Elliotte Rusty Harold, W. Scott Means
XML in a Nutshell, O'Reilly, 2001
 - Neil Bradley
The XML Companion, Addison-Wesley, 2000

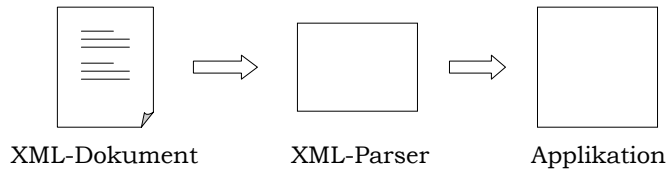
XML-Dokumente

Inhalt

- Wohlgeformte XML-Dokumente
- Elemente
- Attribute
- Zeichen- und Entity-Referenzen
- CDATA-Abschnitte
- Kommentare
- Verarbeitungsanweisungen
- XML-Deklaration

Wohlgeformte XML-Dokumente

Ein XML-Dokument heisst wohlgeformt, wenn es die syntaktischen Regeln der Spezifikation einhlt



Aufbau eines XML-Dokuments

```
<?xml version="1.0"?>
<!-- file people.xml -->

<people>
  <person>
    <name>Alan Turing</name>
    <profession>
      computer scientist
    </profession>
  </person>
  <person>
    <name>Richard M. Feynman</name>
    <profession>
      physicist
    </profession>
  </person>
</people>
```

} Prolog

} Dokument-element

Struktur eines Elements

```
<person>                                — Start-Tag
  <name>Alan Turing</name>               }
  <profession>                           } Inhalt
    computer scientist
  </profession>                         }
</person>                                — End-Tag
```

Bemerkung:

Die Namen im Start- und End-Tag müssen übereinstimmen,
auch bezüglich Gross-/Kleinschreibung

Gültige Elementnamen

- Elementnamen können Buchstaben, Ziffern, Unterstrich (_), Bindestrich (-) und Punkt (.) enthalten
- Elementnamen müssen mit einem Buchstaben oder einem Unterstrich beginnen
- Doppelpunkte sind für Namespaces reserviert

Gültige Namen:

```
<Drivers_License_Number>
<year-month-day>
```

Ungültige Namen:

```
<Driver's_License_Number>
<year/month/day>
<first name>
```

Elementinhalt

Ein Element kann Zeichendaten oder weitere Elemente enthalten

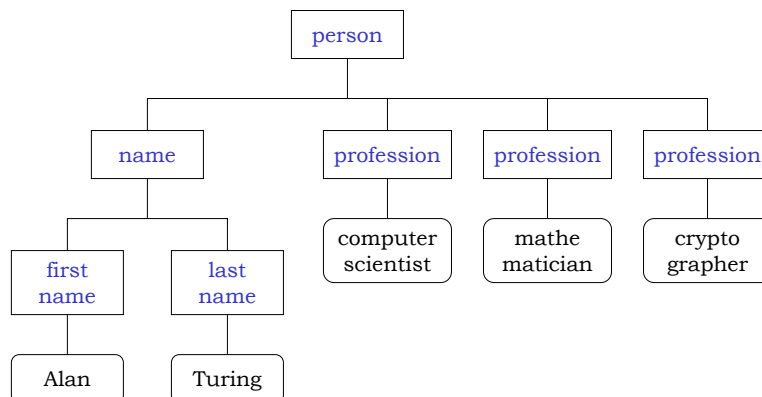
Beispiel:

```
<person>
  <name>
    <firstname>Alan</firstname>
    <lastname>Turing</lastname>
  </name>
  <profession>computer scientist</profession>
  <profession>mathematician</profession>
  <profession>cryptographer</profession>
</person>
```

Bemerkung:

In Zeichendaten dürfen die Zeichen <, & und die Zeichenkette]]> nicht verwendet werden

XML-Baum



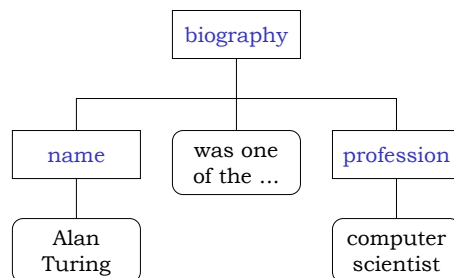
Gemischter Inhalt

Textorientierte XML-Dokumente enthalten oft Elemente mit gemischtem Inhalt bestehend aus Zeichendaten und Kindelementen

Beispiel:

```
<biography>
  <name>Alan Turing</name> was one of the first
  people to truly deserve the name
  <profession>computer scientist</profession>
</biography>
```

XML-Baum



Leere Elemente

Bei leeren Elementen kann der End-Tag fehlen

Beispiel:

```
<image></image>  
<image/>
```

Attribute

Elemente können im Start-Tag Attribute enthalten

Beispiel:

```
<person born="1912/06/23" died="1954/06/07">  
  <name>Alan Turing</name>  
  <profession>computer scientist</profession>  
  <profession>mathematician</profession>  
  <profession>cryptographer</profession>  
</person>
```

Gültige Attributwerte

- Attributwerte müssen in Anführungszeichen (") oder in Apostrophe (') eingeschlossen werden
- Attributwerte dürfen das Zeichen, in das sie eingeschlossen sind, nicht enthalten
- Attributwerte dürfen die Zeichen < und & nicht enthalten

Attribute vs Elemente

- Attribute sind für Metadaten, Elemente für die eigentlichen Daten zu verwenden
- Ein Element kann nur ein Attribut eines bestimmten Namens haben
- Attribute können nicht verschachtelt werden
- Die Reihenfolge von Attributen ist unwesentlich
- Attribute sind zum Teil typisierbar

Zeichenreferenzen

Sonderzeichen können als Zeichenreferenzen mit dem entsprechenden Unicode eingefügt werden

Beispiel:

```
<copyright>Copyright &#xa9; 2001</copyright>
```

Entity-Referenzen

Für die Zeichen &, <, >, ', " gibt es spezielle Entity-Referenzen

Beispiel:

```
<publisher>O&apos;Reilly &amp; Associates</publisher>
```

Zeichen	Entity-Referenz
---------	-----------------

&	&
<	<
>	>
'	'
"	"

CDATA-Abschnitte

Alle Zeichen innerhalb eines CDATA-Abschnitts werden als rohe Daten behandelt

Beispiel:

A table example follows:

```
<![CDATA[
  <table border="1.0">
    <tr>
      <td>Alan Turing</td>
      <td>computer scientist</td>
    </tr>
  </table>
]]>
```

Kommentare

Kommentare dienen der Dokumentation des Source Codes und können überall ausser in Tags stehen

Beispiel:

```
<!-- These links need to be verified -->
```

Verarbeitungsanweisungen

Verarbeitungsanweisungen liefern Informationen für bestimmte Anwendungen (Target) und stehen meistens im Prolog des XML-Dokuments

Beispiel:

```
<?robots index="yes" follow="no"?>
```

XML-Deklaration

Die XML-Deklaration ist optional, muss aber – wenn vorhanden – ganz am Anfang des XML-Dokuments stehen

Beispiel:

```
<?xml version="1.0"
      encoding="US-ASCII" standalone="yes"?>
```

Bemerkungen:

- version hat immer den Wert 1.0
- encoding gibt das verwendete Codierschema an
- standalone="yes" besagt, dass keine externe Dokumenttyp-Definition benötigt wird

Dokumenttyp-Definitionen

Inhalt

- Warum Dokumenttyp-Definitionen?
- Gültige XML-Dokumente
- Element-Deklarationen
- Attribut-Deklarationen
- Entity-Deklarationen

Warum Dokumenttyp-Definitionen?

- Eine Dokumenttyp-Definition (DTD) definiert eine Klasse von XML-Dokumenten, indem sie die Struktur der Dokumente (erlaubte Elemente und Attribute) festlegt
- DTDs sind die Grundlage für den Austausch von XML-Dokumenten und deren Verarbeitung

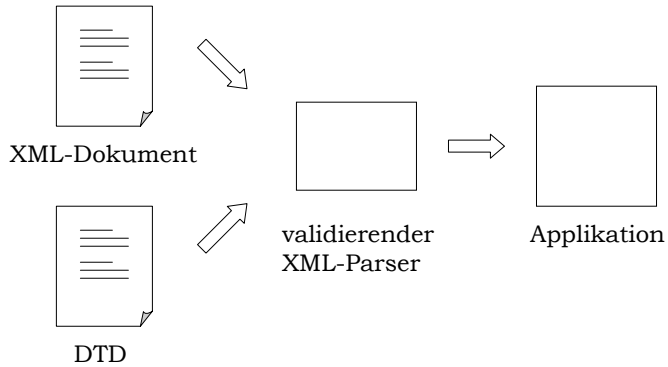
Beispiel einer DTD

```
<!-- file person.dtd -->

<!ELEMENT person (name,profession*)>
<!ATTLIST person born CDATA #IMPLIED
               died CDATA #IMPLIED>
<!ELEMENT name (firstname,lastname)>
<!ELEMENT firstname (#PCDATA)>
<!ELEMENT lastname (#PCDATA)>
<!ELEMENT profession (#PCDATA)>
```

Gültige XML-Dokumente

Ein gültiges XML-Dokument ist ein wohlgeformtes XML-Dokument, dessen Struktur einer bestimmten DTD entspricht



Verknüpfung XML-Dokument mit DTD

```
<?xml version="1.0" standalone="no"?>
<!-- file person.xml -->
<!DOCTYPE person SYSTEM "person.dtd">

<person>
  <name>
    <firstname>Alan</firstname>
    <lastname>Turing</lastname>
  </name>
  <profession>computer scientist</profession>
  <profession>mathematician</profession>
  <profession>cryptographer</profession>
</person>
```

Inhalt einer DTD

- Kommentare
- Verarbeitungsanweisungen
- Element-Deklarationen (ELEMENT)
- Attribut-Deklarationen (ATTLIST)
- Entity-Deklarationen (ENTITY)
- Notations-Deklarationen (NOTATION)

Element-Deklarationen

Für jedes Element muss in der DTD dessen Inhalt definiert werden

Syntax:

```
<!ELEMENT Elementname (Inhaltsmodell)>
```

Inhaltsmodelle:

1. Zeichendaten
2. Kindelemente (Sequenz, Wiederholung, Auswahl)
3. Gemischter Inhalt
4. Leerer Inhalt
5. Beliebiger Inhalt

1. Zeichendaten

Ein Element kann aus Zeichendaten bestehen

Beispiel:

```
<!ELEMENT profession (#PCDATA)>
```

Gültiges Element:

```
<profession>computer scientist</profession>
```

2a. Sequenz

Eine Sequenz legt die Kindelemente eines Elements und deren Reihenfolge fest

Beispiel:

```
<!ELEMENT name (firstname, lastname)>
```

Gültiges Element:

```
<name>
  <firstname>Alan</firstname>
  <lastname>Turing</lastname>
</name>
```

2b. Wiederholung

Wiederholungen von Kindelementen werden mit einem Suffix gekennzeichnet:

- ? null- oder einmaliges Auftreten
- * null- oder mehrmaliges Auftreten
- + ein- oder mehrmaliges Auftreten

Beispiel:

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

Gültiges Element:

```
<person>  
  <profession>computer scientist</profession>  
  <profession>mathematician</profession>  
  <profession>cryptographer</profession>  
</person>
```

2c. Auswahl

Bei einer Auswahl kann das Kindelement aus einer Liste ausgewählt werden

Beispiel:

```
<!ELEMENT activity (profession|hobby)>
```

Gültiges Element:

```
<activity>  
  <hobby>Chess</hobby>  
</activity >
```

2d. Kombinationen

Sequenzen, Wiederholungen und Auswahl können beliebig kombiniert werden

Beispiel:

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

Gültiges Element:

```
<person>
  <name>Alan Turing</name>
  <profession>computer scientist</profession>
  <profession>mathematician</profession>
  <profession>cryptographer</profession>
</person>
```

3. Gemischter Inhalt

Ein gemischter Inhalt eines Elements wird als Wiederholung einer Auswahl von Zeichendaten und Kindelementen definiert

Beispiel:

```
<!ELEMENT biography (#PCDATA|name|profession)*>
```

Gültiges Element:

```
<biography>
  <name>Alan Turing</name> was one of the first
  people to truly deserve the name
  <profession>computer scientist</profession>
</biography>
```

4. Leerer Inhalt

Ein leerer Inhalt eines Elements wird mit dem Schlüsselwort
EMPTY definiert

Beispiel:

```
<!ELEMENT image EMPTY>
```

Gültiges Element:

```
<image source="turing.gif"/>
```

5. Beliebiger Inhalt

Ein beliebiger Inhalt eines Elements wird mit dem Schlüsselwort
ANY definiert

Beispiel:

```
<!ELEMENT page ANY>
```

Gültiges Element:

```
<page>
```

```
...
```

```
</page>
```

Attribut-Deklarationen

Für jedes Element müssen in der DTD dessen Attribute deklariert werden

Syntax:

```
<!ATTLIST Elementname Attributname Attributtyp Vorgabe>
```

Attributtypen:

1. Zeichenkette (CDATA)
2. Namenstoken (NMTOKEN, NMTOKENS)
3. Aufzählung
4. Eindeutiger Wert (ID)
5. Referenz auf ID-Attribut (IDREF, IDREFS)
6. Name einer Entity (ENTITY, ENTITIES)
7. Name einer Notation (NOTATION)

1. Zeichenkette

Ein Attribut kann eine beliebige Zeichenkette enthalten

Beispiel:

```
<!ATTLIST image alt CDATA #REQUIRED>
```

Gültiges Element:

```
<image alt="Alan Turing"/>
```

2. Namenstoken

Ein Namenstoken besteht aus Buchstaben, Ziffern und den Zeichen `_`, `-`, `.`, `:`

Beispiele:

```
<!ATTLIST journal year NMTOKEN #REQUIRED>
<!ATTLIST performances dates NMTOKENS #REQUIRED>
```

Gültige Elemente:

```
<journal year="1990">...</journal>
<performances dates="2001-08-21 2001-08-23">
    ...
</performances>
```

3. Aufzählung

Eine Aufzählung legt die möglichen Attributwerte in einer Liste fest

Beispiel:

```
<!ATTLIST date weekday
    (Monday|Tuesday|Wednesday|Thursday|
    Friday|Saturday|Sunday) #REQUIRED>
```

Gültiges Element:

```
<date weekday="Saturday">...</date>
```

4. Eindeutiger Wert

Ein ID-Attribut enthält einen Namen, der innerhalb des XML-Dokuments eindeutig sein muss

Beispiele:

```
<!ATTLIST department id ID #REQUIRED>  
<!ATTLIST project id ID #REQUIRED>
```

Gültige Elemente:

```
<department id="D5">...</department>  
<project id="P101">...</project>  
<project id="P102">...</project>  
<project id="P103">...</project>
```

5. Referenz auf ID-Attribut

Ein Referenz-Attribut enthält den Namen eines ID-Attributs im gleichen XML-Dokument

Beispiele:

```
<!ATTLIST employee department IDREF #REQUIRED  
                projects IDREFS #REQUIRED>
```

Gültige Elemente:

```
<employee department="D5" projects="P101 P102">  
    ...  
</employee>
```

Vorgabedeklaration

Es gibt vier Vorgabedeklarationen:

#REQUIRED	Attribut ist obligatorisch
#IMPLIED	Attribut ist optional
#FIXED	Attribut hat einen festen Wert
<i>Literal</i>	Attribut hat einen Defaultwert

Beispiele:

```
<!ATTLIST image alt CDATA #IMPLIED>
<!ATTLIST table border NMTOKEN #FIXED "1">
<!ATTLIST url protocol NMTOKEN "http">
```

Allgemeine Entity-Deklarationen

Eine allgemeine Entity ordnet einem Namen (zum Beispiel footer) einen Text zu, mit welchem eine entsprechende Entity-Referenz &footer; im XML-Dokument ersetzt wird

Syntax:

```
<!ENTITY Entity-Name Ersetzungstext>
```

Beispiel:

```
<!ENTITY footer
'<hr/><a href="http://www.oreilly.com/"
O&apos;Reilly Home</a>'>
```

Externe geparste allgemeine Entities

Ist der Text einer allgemeinen Entity zu lang, so kann er in einem externen Dokument gespeichert werden

Syntax:

```
<!ENTITY Entity-Name SYSTEM URI>
```

Beispiel:

```
<!ENTITY footer SYSTEM  
    "http://www.oreilly.com/footer.xml">
```

Parameter-Entities

Eine Parameter-Entity funktioniert wie eine allgemeine Entity, eine entsprechende Entity-Referenz kann aber in der DTD selbst verwendet werden

Syntax:

```
<!ENTITY % Entity-Name Ersetzungstext>
```

Beispiel:

```
<!ENTITY % residential "address,footage,bathroom">  
<!ELEMENT apartment (%residential;,rent)>  
<!ELEMENT house (%residential;,price)>
```

Externe Parameter-Entities

Mit einer externen Parameter-Entity können Definitionen aus einer andern DTD eingefügt werden

Syntax:

```
<!ENTITY % Entity-Name SYSTEM URI>
```

Beispiel:

```
<!ENTITY % names SYSTEM "names.dtd">  
%names;
```

Namespaces

Inhalt

- Warum Namespaces?
- Definition eines Namespace
- Verwendung eines Namespace

Warum Namespaces?

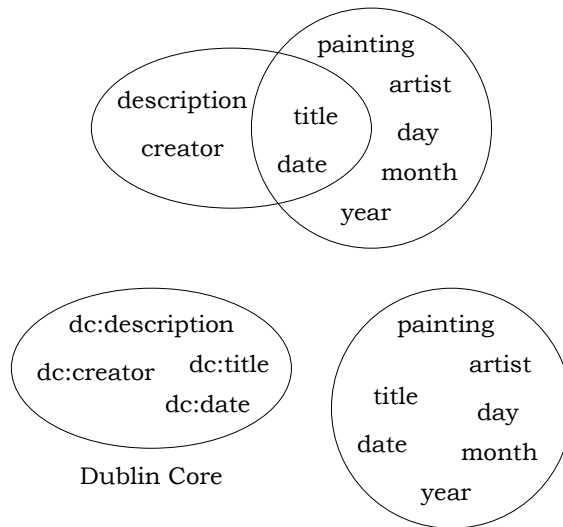
Namespaces werden benötigt um

- Namenskonflikte bei Elementen und Attributen zu vermeiden
- Elemente und Attribute einer bestimmten XML-Anwendung zuzuordnen

Beispiel eines Namenskonflikts

```
<?xml version="1.0"?>
<catalog>
  <description>
    <title>Impressionist Paintings</title>
    <creator>Thomas Crown</creator>
    <date>2000-07-21</date>
  </description>
  <painting>
    <title>Memory of the Garden at Etten</title>
    <artist>Vincent van Gogh</artist>
    <date>
      <month>November</month>
      <year>1888</year>
    </date>
  </painting>
</catalog>
```

Auflösung des Namenskonflikts



XML-Dokument mit Namespace

```
<?xml version="1.0"?>
<!DOCTYPE catalog SYSTEM "catalog.dtd">

<catalog>
  <dc:description xmlns:dc="http://purl.org/dc">
    <dc:title>Impressionist Paintings</dc:title>
    <dc:creator>Thomas Crown</dc:creator>
    <dc:date>2000-07-21</dc:date>
  </dc:description>
  <painting>
    <title>Memory of the Garden at Etten</title>
    <artist>Vincent van Gogh</artist>
    <date>
      <month>November</month>
      <year>1888</year>
    </date>
  </painting>
</catalog>
```

Zugehörige DTD

```
<!ELEMENT catalog (dc:description,painting*)>

<!ELEMENT dc:description (dc:title,dc:creator,dc:date)>
<!ATTLIST dc:description xmlns:dc
    CDATA #FIXED "http://purl.org/dc">
<!ELEMENT dc:title      (#PCDATA)>
<!ELEMENT dc:creator    (#PCDATA)>
<!ELEMENT dc:date       (#PCDATA)>

<!ELEMENT painting (title,artist,date)>
<!ELEMENT title     (#PCDATA)>
<!ELEMENT artist    (#PCDATA)>
<!ELEMENT date      (day?,month?,year)>
<!ELEMENT day       (#PCDATA)>
<!ELEMENT month     (#PCDATA)>
<!ELEMENT year      (#PCDATA)>
```

Definition eines Namespace

Ein Namespace wird durch einen URI (Uniform Resource Identifier) definiert

Beispiel:

<http://purl.org/dc>

Bemerkungen:

- Der URI wird in der Spezifikation der entsprechenden XML-Applikation festgelegt
- Der URI zeigt nicht unbedingt auf eine aktuelle HTML-Seite
- Parser vergleichen Namespace-URIs zeichenweise

Zuordnung eines Präfix

Mit einem `xmlns`-Attribut kann einem Namespace-URI ein Präfix zugeordnet werden

Beispiel:

```
xmlns:dc="http://purl.org/dc"
```

Bemerkungen:

- Der Präfix ist nicht standardisiert
- Der Präfix ist nur innerhalb des Elements gültig, in dem er definiert wird

Qualifizierte Elementnamen

Um Elemente in einen Namespace einzufügen, werden sie mit dem entsprechenden Präfix qualifiziert

Beispiel:

```
<dc:title>Impressionist Paintings</dc:title>
```

Setzen eines Default-Namespace

Wird in einem Element ein Namespace ohne Präfix definiert, so gehören das Element selbst und seine Kindelemente implizit zu diesem Namespace

Beispiel:

```
<catalog>
  <description xmlns="http://purl.org/dc">
    <title>Impressionist Paintings</title>
    <creator>Thomas Crown</creator>
    <date>2000-07-21</date>
  </description>
  <painting>
    ...
  </painting>
</catalog>
```

XML-Schemas

Inhalt

- Warum XML-Schemas?
- Aufbau eines XML-Schemas
- Element- und Attributdeklarationen
- Einfache und komplexe Typen
- Ableitungen von Typen
- Schlüssel und Referenzen
- Target-Namespace

Warum XML-Schemas?

XML-Schemas definieren wie DTDs Klassen von XML-Dokumenten, haben aber folgende Vorteile:

- XML-Schemas sind in XML geschrieben
- XML-Schemas erlauben die vollständige Typisierung von Elementen und Attributen
- XML-Schemas unterstützen Namespaces

Beispiel eines XML-Schemas

```
<?xml version="1.0"?>
<!-- file addressbook.xsd -->

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="addressbook">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="address" type="Address"
          minOccurs="0" maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

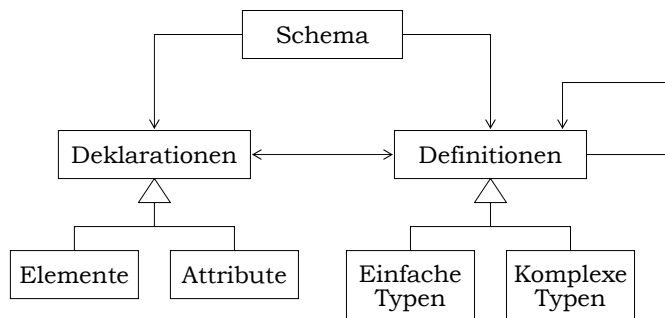
  <xsd:complexType name="Address">
    <xsd:sequence>
      <xsd:element name="name" type="xsd:string"/>
      <xsd:element name="street" type="xsd:string"/>
      <xsd:element name="city" type="xsd:string"/>
    </xsd:sequence>
    <xsd:attribute name="category" type="xsd:string"/>
  </xsd:complexType>
</xsd:schema>
```

Gültiges Instanzdokument

```
<?xml version="1.0"?>
<!-- file addressbook.xml -->

<addressbook>
  <address category="home">
    <name>Alice Smith</name>
    <street>123 Maple Street</street>
    <city>Mill Valley</city>
  </address>
  <address category="office">
    <name>Helen Zoe</name>
    <street>47 Eden Street</street>
    <city>Cambridge</city>
  </address>
</addressbook>
```

Aufbau eines XML-Schemas



Die Elemente von XML-Schema gehören zum Namespace
<http://www.w3.org/2001/XMLSchema>

Deklarationen vs Definitionen

- Deklarationen beschreiben Elemente und Attribute, die in einem Instanzdokument auftreten dürfen

Beispiel:

```
<xsd:element name="addressbook">
  ...
</xsd:element>
```

- Definitionen definieren Typen, die in Element- oder Attributdeklarationen verwendet werden können

Beispiel:

```
<xsd:complexType name="Address">
  ...
</xsd:complexType>
```

Globale vs lokale Elementdeklarationen

- Globale Elementdeklarationen sind Kindelemente des schema-Elements und dürfen als Dokumentschemaelement eines Instanzdokuments auftreten

Beispiel:

```
<xsd:schema>
  <xsd:element name="addressbook">
    ...
  </xsd:element>
</xsd:schema>
```

- Lokale Elementdeklarationen sind Kindelemente einer Typdefinition

Beispiel:

```
<xsd:complexType name="Address">
  <xsd:sequence>
    <xsd:element name="name" type="xsd:string"/>
    ...
  </xsd:sequence>
</xsd:complexType>
```

Anonyme vs benannte Typen

- Anonyme Typen sind in einer Element- oder Attributdeklaration eingebettet

Beispiel:

```
<xsd:element name="addressbook">
  <xsd:complexType>
    ...
  </xsd:complexType>
</xsd:element>
```

- Benannte Typen werden separat definiert und können in Element- oder Attributdeklarationen referenziert werden

Beispiel:

```
<xsd:element name="address" type="Address">
<xsd:complexType name="Address">
  ...
</xsd:complexType>
```

Einfache vs komplexe Typen

- Einfache Typen sind Strings, Zahlen, Zeiteinheiten usw.

Beispiel:

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

- Komplexe Typen enthalten Kindelemente oder Attribute

Beispiel:

```
<xsd:complexType name="Address">
  <xsd:sequence>
    ...
  </xsd:sequence>
  <xsd:attribute name="category" type="xsd:string"/>
</xsd:complexType>
```

Elementdeklarationen

Eine Elementdeklaration definiert den Namen eines Elements und dessen Typ, welcher das Inhaltsmodell des Elements bestimmt

Syntax:

```
<xsd:element name="Elementname" type="Elementtyp"  
             minOccurs/maxOccurs="Auftreten"  
             default/fixed="Vorgabe" />
```

Auftreten von Elementen

Mit den Attributen `minOccurs` und `maxOccurs` kann die mögliche Anzahl Wiederholungen eines Elements festgelegt werden (Default ist 1)

Beispiel:

```
<xsd:element name="addressbook">  
  <xsd:sequence>  
    <xsd:element name="address" type="Address"  
                 minOccurs="0" maxOccurs="unbounded" />  
  </xsd:sequence>  
</xsd:complexType>
```

Vorgabe für Elemente

Mit den Attributen `default` und `fixed` kann ein Default- bzw. ein fester Wert eines einfachen Elements festgelegt werden

Beispiel:

```
<xsd:element name="protocol" type="xsd:string"
             fixed="http"/>
<xsd:element name="port" type="xsd:integer"
             minOccurs="0" default="80"/>
```

Attributdeklarationen

Eine Attributdeklaration definiert den Namen eines Attributs und dessen Typ, der immer einfach ist

Syntax:

```
<xsd:attribute name="Attributname" type="Attributtyp"
               use="Auftreten"
               default/fixed="Vorgabe"/>
```

Auftreten von Attributen

Mit dem Attribut `use` kann das Auftreten eines Attributs festgelegt werden:

<code>required</code>	Attribut ist obligatorisch
<code>optional</code>	Attribut ist optional (Default)
<code>prohibited</code>	Attribut ist verboten

Beispiel:

```
<xsd:attribute name="category" type="xsd:string"
               use="required"/>
```

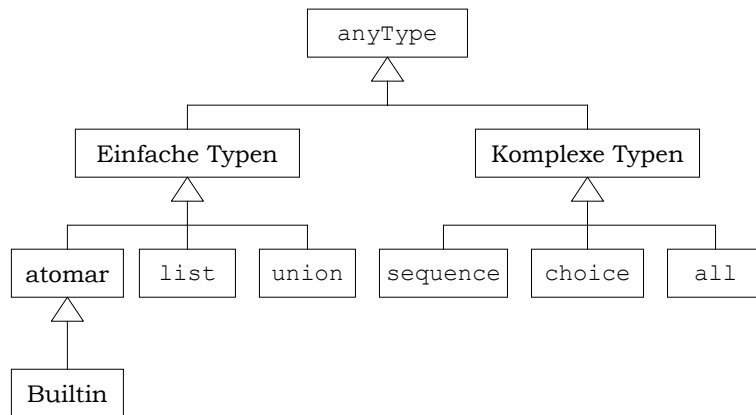
Vorgabe von Attributen

Mit den Attributen `default` und `fixed` kann ein Default- bzw. ein fester Wert eines optionalen Attributs definiert werden

Beispiel:

```
<xsd:attribute name="category" type="xsd:string"
               use="optional" default="office"/>
```

Typsystem



Einfache Typen

Einfache Typen dienen der Spezifikation einfacher Textinhalte von Elementen und Attributen:

1. Atomare Typen (Builtin, abgeleitete Typen)
2. List-Typen
3. Union-Typen

1a. Builtin-Typen

boolean	string	time
byte	normalizedString	dateTime
unsignedByte	token	duration
base64Binary	Name	gMonth
hexBinary	QName	gYear
integer	NCName	gYearMonth
positiveInteger	anyURI	gDay
negativeInteger	language	gMonthDay
nonPositiveInteger		
nonNegativeInteger		
int/unsignedInt		
long/unsignedLong		
short/unsignedShort		
decimal		
float		
double		

1b. Abgeleitete einfache Typen

Durch Einschränkung sogenannter Facetten können aus einfachen Typen neue abgeleitet werden

Syntax:

```
<xsd:simpleType name="Typname">  
  <xsd:restriction base="Basistyp">  
    Facetten  
  </xsd:restriction>  
</xsd:simpleType>
```

1b. Anwendbare Facetten

	minInclusive minExclusive maxInclusive maxExclusive	totalDigits fraction- Digits	length minLength maxLength	pattern enumeration whiteSpace
Numerische Typen	ja	ja	nein	ja
String-Typen	nein	nein	ja	ja
Zeit-Typen	ja	nein	nein	ja

1b. Beispiele abgeleiteter Typen

```
<xsd:simpleType name="LotteryNumber">
  <xsd:restriction base="xsd:integer">
    <xsd:minInclusive value="1"/>
    <xsd:maxInclusive value="45"/>
  </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="UKPostcode">
  <xsd:restriction base="xsd:string">
    <xsd:pattern value="[A-Z]{2}\d\s\d[A-Z]{2}"/>
  </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="USState">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="AK"/>
    <xsd:enumeration value="AL"/>
    ...
  </xsd:restriction>
</xsd:simpleType>
```

2. List-Typen

Ein List-Typ definiert eine Liste einfacher Typen

Beispiel:

```
<xsd:simpleType name="LotteryNumbers">
  <xsd:list itemType="LotteryNumber"/>
</xsd:simpleType>
<xsd:simpleType name="LotteryDrawing">
  <xsd:restriction base="LotteryNumbers">
    <xsd:length value="6"/>
  </xsd:restriction>
</xsd:simpleType>
```

3. Union-Typen

Ein Union-Typ definiert eine Vereinigung einfacher Typen

Beispiel:

```
<xsd:simpleType name="Infinity">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="infinity"/>
  </xsd:restriction>
</xsd:simpleType>
<xsd:simpleType name="ExtendedInteger">
  <xsd:union memberTypes="xsd:integer Infinity">
  </xsd:union>
</xsd:simpleType>
```

Komplexe Typen

Komplexe Typen dienen der Spezifikation folgender Inhaltsmodelle von Elementen:

1. Elementinhalt
2. Gemischter Inhalt
3. Leerer Inhalt
4. Beliebiger Inhalt

1a. Sequenz

Das Element `sequence` definiert eine Sequenz von Kindelementen

Beispiel:

```
<xsd:complexType name="Address">
  <xsd:sequence>
    <xsd:element name="name" type="xsd:string"/>
    <xsd:element name="street" type="xsd:string"/>
    <xsd:element name="city" type="xsd:string"/>
  </xsd:sequence>
</xsd:complexType>
```

1b. Auswahl

Das Element `choice` definiert eine Auswahl von Kindelementen

Beispiel:

```
<xsd:complexType name="Address">
  ...
  <xsd:choice>
    <xsd:element name="state" type="xsd:string"/>
    <xsd:element name="province" type="xsd:string"/>
  </xsd:choice>
</xsd:complexType>
```

1c. Beliebige Reihenfolge

Das Element `all` definiert eine Auswahl, in welcher die Kindelemente in beliebiger Reihenfolge auftreten dürfen

Beispiel:

```
<xsd:complexType name="Contact">
  <xsd:all>
    <xsd:element name="phone" type="xsd:string"
      minOccurs="0"/>
    <xsd:element name="email" type="xsd:string"
      minOccurs="0"/>
  </xsd:all>
</xsd:complexType>
```

1d. Kombinationen

Sequenzen und Auswahl können beliebig kombiniert werden

Beispiel:

```
<xsd:complexType name="Person">
  <xsd:choice>
    <xsd:element name="name" type="xsd:string"/>
    <xsd:sequence>
      <xsd:element name="firstname" type="xsd:string"/>
      <xsd:element name="lastname" type="xsd:string"/>
    </xsd:sequence>
  </xsd:choice>
</xsd:complexType>
```

2. Gemischter Inhalt

Mit dem Attribut `mixed` werden Typen definiert, die sowohl Text als auch Kindelemente enthalten dürfen

Beispiel:

```
<xsd:complexType name="Letter" mixed="true">
  <xsd:sequence>
    <xsd:element name="salutation" type="xsd:string"/>
    <xsd:element name="greetings" type="xsd:string"/>
  </xsd:sequence>
</xsd:complexType>
```

3. Leerer Inhalt

Ein leerer Inhalt wird durch einen komplexen Typ ohne Kindelemente definiert

Beispiel:

```
<xsd:complexType name="Image">
  <xsd:attribute name="source" type="xsd:string"/>
</xsd:complexType>
```

4. Beliebiger Inhalt

Ein beliebiger Inhalt kann mit dem Element any definiert werden

Beispiel:

```
<xsd:complexType name="ExtensionSpace">
  <xsd:sequence>
    <xsd:any minOccurs="0" maxOccurs="unbounded"
      processContents="skip"/>
  </xsd:sequence>
</xsd:complexType>
```

Abgeleitete komplexe Typen

Durch Hinzufügen zusätzlicher Elemente oder Attribute können von bestehenden Typen neue Typen abgeleitet werden

Syntax:

```
<xsd:complexType name="Typname">
  <xsd:simpleContent/complexContent>
    <xsd:extension base="Basistyp">
      Elemente und Attribute
    </xsd:extension>
  </xsd:simpleContent/complexContent>
</xsd:complexType>
```

Beispiele abgeleiteter Typen

```
<xsd:complexType name="Price">
  <xsd:simpleContent>
    <xsd:extension base="xsd:decimal">
      <xsd:attribute name="currency"
        type="xsd:string"/>
    </xsd:extension>
  </xsd:simpleContent>
</xsd:complexType>

<xsd:complexType name="USAddress">
  <xsd:complexContent>
    <xsd:extension base="Address">
      <xsd:sequence>
        <xsd:element name="state" type="USState"/>
        <xsd:element name="zip" type="xsd:integer"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```

Schlüssel

Mittels `key` können Elemente oder Attribute als Schlüssel definiert und damit die Eindeutigkeit innerhalb eines Instanzdokuments garantiert werden

Beispiel:

```
<xsd:key name="departmentId">
  <xsd:selector xpath="department"/>
  <xsd:field xpath="@id"/>
</xsd:key>
```

Bemerkung:

Das Element `selector` bezeichnet eine Menge von Elementen, `field` das Schlüsselfeld relativ zu diesen Elementen (mehrere möglich)

Referenzen auf Schlüssel

Mittels `keyref` können Referenzen auf Schlüssel definiert werden

Beispiel:

```
<xsd:keyref name="departmentRef"
  refer="departmentId">
  <xsd:selector xpath="employee/department"/>
  <xsd:field xpath="@ref"/>
</xsd:keyref>
```

Target-Namespace

Der Target-Namespace legt fest, zu welchem Namespace die definierten Elemente und Typen gehören

Beispiel:

```
<xsd:schema
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:ab="http://www.example.org/addresses"
  targetNamespace="http://www.example.org/addresses"
  elementFormDefault="qualified"
  attributeFormDefault="unqualified">
```

Bemerkung:

Die Attribute `elementFormDefault` und `attributeFormDefault` spezifizieren, ob lokale Elemente bzw. Attribute qualifiziert werden müssen

Verknüpfung von XML-Dokument und Schema

Mit dem Attribut `schemaLocation` (bzw. `noNamespaceSchemaLocation` wenn kein Target-Namespace verwendet wird) kann ein XML-Dokument mit einem Schema verknüpft werden

Beispiel:

```
<addressbook
  xmlns="http://www.example.org/addresses"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.example.org/addresses
    addressbook.xsd">
  ...
</addressbook>
```