



# ***XML Schema: Strukturen und Datentypen***

<http://www.w3.org/TR/xmlschema-1>

<http://www.w3.org/TR/xmlschema-2>



## **Warum reichen DTD nicht?**



- **Attribute**
  - Keine selbständigen Objekte, nur lokal einem Objekt zugeordnet
  - Keine Gruppenbildung möglich
- **Elemente**
  - Keine Defaultbelegung möglich, Inhalt nicht validierbar
  - Keine Wiederholungsfaktoren
  - Gruppenbildung nur indirekt möglich
  - Nur global wirksame Deklarationen möglich
- **Beide**
  - Typisierung nicht ausreichend
  - Keine benutzerdefinierten Typen
  - Syntax erfordert speziellen Markup
  - Namespace-Konzept nicht integriert
  - Komplexe Strukturen, objektorientiertes Vorgehen schlecht unterstützt



## Schema - welches Schema?



- XML DTD
  - Seit langer Zeit die gemeinsame Grundlage
  - Herkunft SGML
- XDR (XML-Data Reduced)
  - Microsoft-Standard, älter als W3C XML Schema
  - z.B. in MSXML 3.0, BizTalk, SQL 200
  - wird nun zunehmend verdrängt von W3C XML Schema
- Schematron
  - Regelbasierter Ansatz, z.B. zur Abbildung komplexer Abhängigkeiten zwischen Elementen
  - Gut kombinierbar mit W3C XML Schema
  - Siehe auch:  
<http://www.ascc.net/xml/resource/schematron/>



## Schema - welches Schema?



- Examplotron
  - Einfacher, aber wirksamer Ansatz - allerdings mit nur eingeschränkten Möglichkeiten
  - ausgehend von „Beispielinstanzen mit Zusätzen“
  - siehe auch: <http://examplotron.org>
- RELAX NG
  - Zusammenfassung zweier Schema-Sprachen: RELAX und TREX
  - Große Ähnlichkeit zu W3C XML Schema, z.B. XML Syntax
  - Formaler (im math. Sinn), frei von einigen komplizierten Eigenschaften von W3C XML Schema
  - Erwartet die Definition zulässiger Elemente und Attribute in den Dokumentinstanzen
  - Datentypen von W3C XML Schema können verwendet werden
  - siehe auch: <http://relaxng.org>



# Von der DTD zum Schema

Ein beispiel-orientierter „Einstieg“



## Vorbemerkungen



- Dieser Abschnitt führt wesentliche Schema-Konstrukte anhand von Beispielen ein („induktive“ Methode).
- Diese werden erst einmal nur vorgestellt (und mündlich diskutiert), nicht systematisch abgeleitet. Der Sinn ist, einen „Vorgeschmack“ auf und ersten Eindruck von XML Schema zu erhalten.
- Da wir die Möglichkeiten der DTD kennen, erklären sich Schema-Bespiele fast von selbst, wenn sie exakt nachbilden, was ein bestimmtes DTD-Konstrukt leistet.
- Weitergehende Möglichkeiten von XML Schema sowie eine - zumindest stellenweise - vollständige, „deduktive“ Erschließung ist späteren Abschnitten vorbehalten.



## Von DTD zu Schema: Programm



- Dokumententyp-Deklaration, Verbindung zur XML-Instanz, genereller Aufbau eines Schema-Dokuments
- Elementtyp-Deklarationen
  - EMPTY (nur Attribute)
  - ANY
  - Children (Sequence, Choice)
  - Mixed
- Attributlisten-Deklarationen
  - StringType (CDATA)
  - TokenizedType (ID, IDREF, IDREFS, ENTITY ENTITIES, NMTOKEN, NMTOKENS)
  - EnumeratedType (Enumeration, NotationType)
  - Attribut-Defaults (#IMPLIED, #REQUIRED, (#FIXED) default\_val.)
- NOTATION-Deklaration (SYSTEM, PUBLIC)



## Dokumententyp-Deklaration



- Die Dokumententyp-Deklaration entfällt bzw. kann entfallen!
- Statt dessen vergibt man „*hints*“ über spezielle globale Attribute an den Schema-Validierer:
- DTD-Einbindung:  

```
<!DOCTYPE Dozent SYSTEM "dozent.dtd" [ ... ]>
```
- XML Schema-Einbindung:  

```
<Dozent  
  xmlns="http://fbi.fh-wi.de/~werntges/ns/dozent"  
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
  xsi:schemaLocation=  
    "http://fbi.fh-wi.de/~werntges/ns/dozent dozent.xsd">
```

  - Man beachte die paarweise Auflistung von Namespace-URI und URL in xsi:schemaLocation.
  - Grundsätzlich lassen sich auch mehrere solche Schema-Paare angeben - alles in einem Attributwert.



## Aufbau eines Schema-Dokuments



- Schema-Dateien sind eigenständige XML-Dokumente, und zwar Instanzen des Dokumenttyps „schema“ aus einem reservierten Namensraum.
- Sie sind KEINE externen *entities* der beschriebenen XML-Dokumentinstanzen!
- XML Schema-Rahmen:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema
  xmlns:xsd = "http://www.w3.org/2001/XMLSchema"
  targetNamespace =
    "http://fbi.fh-wi.de/~werntges/ns/dozent"
  xmlns:target = "http://fbi.fh-wi.de/~werntges/ns/dozent"
  elementFormDefault="qualified">
  <!-- <xsd:element>, <xsd:attribute>, <xsd:group> ... -->
</xsd:schema>
```



## Erläuterungen



- `<?xml ... ?>`
  - Die normale XML-Deklaration (optional)
- `xmlns:xsd = "..."`
  - Eine verbreitete Konvention zur Bezeichnung des Namensraums von XML Schema. Siehe auch vereinfachtes Beispiel weiter unten.
- `targetNamespace = "..."`
  - Der Namensraum, für den das im Folgenden definierte „Vokabular“ bestimmt ist, i.d.R. der Ihrer Dokumentinstanz
- `xmlns:target = "..."`
  - Ein lokal definiertes Namensraum-Präfix, das benötigt wird, um in der Schema-Datei auf hier deklarierte Elemente verweisen zu können.
- `elementFormDefault = "qualified"` (default wäre "unqualified")
  - Bewirkt „normales“ Namensraum-Verhalten, verhindert die gemischte Verwendung von Elementen mit und ohne Namensraum



## Schema-Dokument mit *default*-Namensraum



- Häufig wird der Namensraum von XML Schema zum *default* in Schemainstanzen erklärt, um so zahlreiche Präfixes zu vermeiden.
- Beispiel:

```
<?xml version="1.0" encoding="UTF-8"?>
<schema
  xmlns = "http://www.w3.org/2001/XMLSchema"
  targetNamespace =
    "http://fbi.fh-wi.de/~werntges/ns/dozent"
  xmlns:target =
    "http://fbi.fh-wi.de/~werntges/ns/dozent"
  elementFormDefault="qualified">
  <!-- <element>, <attribute>, <group> ... -->
</schema>
- ...
```



## Elementtyp-Deklaration



- #PCDATA  
(nur Freitext, häufiger Spezialfall von Mixed)
  - DTD:  

```
<!ELEMENT Vorname #PCDATA>
```
  - XML Schema:  

```
<xsd:element
  name="Vorname"
  type="xsd:string"/>
```
  - Bemerkungen:  
Der eingebaute Datentype „string“ kommt der Bedeutung von #PCDATA sehr nahe.



## Elementtyp-Deklaration



- ANY (beliebige Inhalte, eher „pathologisch“)

– DTD:

```
<!ELEMENT Container ANY>
```

– XML Schema:

```
<xsd:element name="Container">
  <xsd:complexType>
    <xsd:any namespace="##any"
      processContents="lax"
      minOccurs="0"
      maxOccurs="unbounded"/>
  </xsd:complexType>
</xsd:element>
```



## Elementtyp-Deklaration



- EMPTY (nur Attribute)

– DTD:

```
<!ELEMENT Beschäftigungsverhältnis EMPTY>
<!ATTLIST Beschäftigungsverhältnis Art ... >
```

– XML Schema:

```
<xsd:element name="Beschäftigungsverhältnis">
  <xsd:complexType>
    <xsd:attribute name="Art" type="..."/>
  </xsd:complexType>
</xsd:element>
```

– Bemerkungen:

Kurzschreibweise! Ausgelassen (vor *attribute*) wurde:

```
<xsd:complexContent>
  <xsd:restriction base="xsd:anyType">
```

Siehe auch: *XML Schema Tutorial*, „2.5.3 Empty Content“



## Elementtyp-Deklaration



- Children (hier nur direkte Unterelemente)

– DTD:

```
<!ELEMENT Name (Vorname, Nachname)>
```

– XML Schema:

```
<xsd:element name="Name">  
  <xsd:complexType>  
    <xsd:sequence>  
      <xsd:element ref="target:Vorname"/>  
      <xsd:element ref="target:Nachname"/>  
    </xsd:sequence>  
  </xsd:complexType>  
</xsd:element>
```



## Elementtyp-Deklaration



- Mixed, Choice

– DTD:

```
<!ELEMENT abstract (#PCDATA|emph|quot)*>
```

– XML Schema:

```
<xsd:element name="abstract">  
  <xsd:complexType mixed="true">  
    <xsd:choice minOccurs="0"  
      maxOccurs="unbounded">  
      <xsd:element ref="target:emph"/>  
      <xsd:element ref="target:quot"/>  
    </xsd:choice>  
  </xsd:complexType>  
</xsd:element>
```



- StringType
  - DTD:  
`<!ATTLIST elem attname CDATA #IMPLIED>`
  - XML Schema:  
`<xsd:attribute  
    name="attname"  
    type="xsd:string"  
    use="optional"/>`
- Bemerkungen:
  - Attribute in XML Schema können ähnlich wie Elemente lokal oder global eigenständig deklariert werden.
  - Ihre Zuordnung zu Elementen erfolgt über den Kontext ihrer Einbindung in einen `complexType`.



Erweiterung eines einfachen Datentyps um ein Attribut:

```
<xs:element name="width">  
  <xs:complexType>  
    <xs:simpleContent>  
      <xs:extension base="xs:nonNegativeInteger">  
        <xs:attribute name="unit" type="xs:NMTOKEN"/>  
      </xs:extension>  
    </xs:simpleContent>  
  </xs:complexType>  
</xs:element>
```

Anwendung:

```
<width unit="cm">25</width>
```



- TokenizedType
  - DTD:

```
<!ATTLIST elem id ID #REQUIRED>
```
  - XML Schema:

```
<xsd:attribute  
    name="id"  
    type="xsd:ID"  
    use="required"/>
```
- Bemerkungen:
  - In XML Schema sind die aus DTD bekannten einfachen „Token-artigen“ Datentypen direkt verfügbar:  
ID, IDREF, IDREFS, NMTOKEN, NMTOKENS, ENTITY, ENTITIES.



- EnumeratedType (NotationType, Enumeration)
  - DTD:

```
<!ATTLIST Today my-date-fmt NOTATION  
    (ISODATE|EUDATE) #REQUIRED>
```
  - XML Schema:

```
<xsd:attribute  
    name="my-date-fmt"  
    type="target:dateNotation"  
    use="required"/>
```
- Bemerkungen:
  - In XML Schema lassen sich eigene Datentypen aus den eingebauten ableiten und dann wie gewohnt verwenden.



## Attributtyp-Deklaration



- Ableitung eines eigenen Datentyps, hier: Auswahlliste von NOTATIONS

– DTD:

```
<!ATTLIST Today my-date-fmt NOTATION  
    (ISODATE|EUDATE) #REQUIRED>
```

– XML Schema:

```
<xsd:simpleType name="dateNotation">  
    <xsd:restriction base="xsd:NOTATION">  
        <xsd:enumeration value="ISODATE"/>  
        <xsd:enumeration value="EUDATE"/>  
    </xsd:restriction>  
</xsd:simpleType>
```



## Attributtyp-Deklaration



- Ableitung eines eigenen Datentyps, hier: Auswahlliste

– DTD:

```
<!ATTLIST Vorlesung Wochentag  
    (Montag|Dienstag|...|Sonntag) #IMPLIED>
```

– XML Schema:

```
<xsd:simpleType name="Wochentag">  
    <xsd:restriction base="xsd:string">  
        <xsd:enumeration value="Montag"/>  
        ...  
        <xsd:enumeration value="Sonntag"/>  
    </xsd:restriction>  
</xsd:simpleType>
```



- Default-Deklarationen in XML-Schema
  - a) #REQUIRED, #IMPLIED:

`<xsd:attribute>` kennt das Attribut **use**  
Zulässige Werte: "required", "optional", ferner "prohibited"  
(etwa zum gezielten Blockieren / Reservieren)
  - b) „Echte“ Defaultwert-Belegung, optional mit „#FIXED“:

`<xsd:attribute>` kennt die Attribute **default** und **fixed**  
Diese werden einfach (alternativ) mit dem gewünschten Defaultwert belegt.



- Default-Deklarationen in XML-Schema, Beispiel:
  - DTD:

```
<!ATTLIST elem atname1 CDATA "mydefaultvalue"
              atname2 CDATA #FIXED "fixedval">
```
  - XML Schema:

```
<xsd:attribute
  name="atname1"
  type="xsd:string"
  default="mydefaultvalue"/>
<xsd:attribute
  name="atname2"
  type="xsd:string"
  fixed="fixedval"/>
```



- SYSTEM vs. PUBLIC

- DTD:

```
<!NOTATION ISODATE SYSTEM
    "http://www.iso.ch/date_specification">
<!NOTATION DOCTYPE PUBLIC
    "-//OASIS//DTD DocBook V3.1//EN"
    "docbook/3.1/docbook.dtd">
```

- XML Schema:

```
<xsd:notation name="ISODATE"
    system =
        "http://www.iso.ch/date_specification"/>
<xsd:notation name="DOCTYPE"
    system="docbook/3.1/docbook.dtd"
    public="-//OASIS//DTD DocBook V3.1//EN"/>
```



- Elemente von XML-Schema, bisher:

- element
  - attribute
  - notation
  - simpleType, complexType
  - any, complexContent
  - restriction
  - enumeration, sequence, choice



# Datentypen in XML Schema

Vordefinierte Datentypen  
„DT4DTD“  
Ableitung eigener Datentypen



## Datentypen-Anatomie



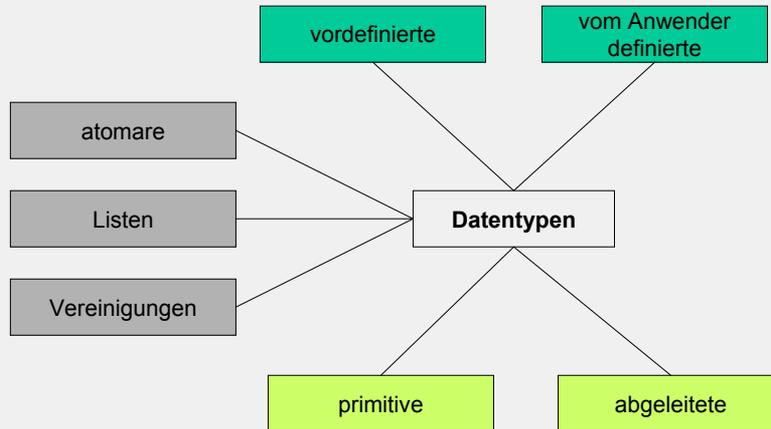
- Datentypen lassen sich gut mit mengentheoretischen Begriffen beschreiben:
  - Formal ist jeder Datentyp ein Triplet (3-Tupel)  
  
(Wertemenge, lexikalische Menge, Facettenmenge)
- Wertemenge  $W$ 
  - Die i.a. diskrete Menge der (abstrakten) Werte, die der Datentyp annehmen kann.
  - Definierbarkeit von  $W$ 
    - axiomatisch
    - durch explizite Aufzählung ihrer Elemente
    - durch Ableitung (Untermenge, Mengendifferenz) von einer gegebenen anderen Wertemenge
    - durch Kombinationen mehrerer gegebener Wertemengen (Bildung der Vereinigungsmenge sowie von Listen aus Elementen anderer Mengen)



- Lexikalische Menge L
  - Die Menge der Symbole, aus denen die Wertemenge abgeleitet wird.
  - Jedem Element der Wertemenge entspricht mindestens ein Element der lexikalischen Menge.
  - Beispiel:
    - 100, 100.00, 1e2 sind drei Elemente der lexikalischen Menge des Datentyps „float“ und meinen dasselbe Wertemengenelement 100.
  - „Kanonische Darstellung“:
    - Eine Untermenge von L, bijektiv zu W
    - Für jeden Datentypen benötigt man angepasste Regeln zur Festlegung der Kanonischen Darstellung. Einzelheiten s. *XML Schema: Datatypes*.



- Facettenmenge
  - Die Facettenmenge eines Datentypen besteht aus fundamentalen und (optionalen) einschränkenden Facetten.
  - Fundamentale Facette:
    - Eine abstrakte Eigenschaft zur semantischen Charakterisierung der Elemente der Wertemenge W. Es gibt folgende 5 Arten:
      - equal* Sei a und b aus W. Dann ist  $a=b$ ,  $a \neq b$  immer ermittelbar
      - ordered* Sei a und b aus W. Dann ist  $a < b$  immer ermittelbar, etc.
      - bounded* Es lassen sich obere/untere Grenzen von W benennen
      - cardinality* W ist „endlich“ oder „abzählbar unendlich“
      - numeric* W besteht aus numerischen Werten
    - Details: Tabelle in C.1 von *XML Schema: Datatypes* ordnet jedem vordefinierten Datentyp die Werte dieser Facetten zu.
  - Einschränkende Facette:
    - Eine optionale Einschränkung der zulässigen Wertemenge
    - W3C XML Schema kennt 12 einschränkende Facetten.
    - Beispiel: „Erste Ziffer muss ‚1‘ sein“



- W3C XML Schema besitzt zahlreiche vordefinierte **Datentypen**. Sie gliedern sich in **primitive** und **abgeleitete** Datentypen.
- Datentypen im hier vorgestellten Sinn beruhen auf Konzepten aus ISO 11404 (sprachunabhängige Datentypen), auf SQL-Datentypen, und auf Datentypen gängiger Programmiersprachen wie Java - und natürlich auf SGML/XML Attributtypen.
- Die primitiven Datentypen bilden die Grundlage aller abgeleiteten Datentypen - auch der eigenen.
- Anwendern ist es nicht möglich, die Menge der primitiven Datentypen zu vergrößern. Selbst definierte Datentypen sind stets abgeleitete.



## Vordefinierte Datentypen



- Primitive Datentypen
  - Gewöhnliche Datentypen zur Programmierung
    - `string` entspricht CDATA
    - `boolean` `true`, `false`, `1`, `0`
    - `decimal` Dezimalzahlen beliebiger Genauigkeit  
min. 18 Stellen vor dem Dezimalzeichen (.)  
opt. führendes Vorzeichen (+,-)
    - `float` *single precision* 32-bit, incl. der speziellen Werte `0`, `-0`, `INF`, `-INF`, `NaN`  
`-1E4`, `1267.43233E12`, `12.78e-2`, `12`, `INF`
    - `double` *double precision* 64-bit, analog `float`



## Vordefinierte Datentypen



- Primitive Datentypen (Forts.)
  - XML-Datentypen
    - `anyURI` Zur Spezifikation von URIs, incl. *fragment identifiers* und XPointer-Ausdrücken
    - `QName` *Namespace-qualified name*.  
Basiert auf „Name“ aus XML 1.0 und „QName“ aus XML Namespaces.
    - `NOTATION` Analog zum gleichnamigen Attributtypen in XML DTDs.
  - Binäre Datentypen
    - `hexBinary` Ziffern aus [0-9a-fA-F]. 1 byte = 2 hex-Ziffern  
„0FB7“ = 4023 dec.
    - `base64Binary` 3-nach-4 Codierung gemäß RFC 2045 Kap. 6.8 (MIME part 1). Zeichen aus [A-Za-z0-9+/] und =  
Bsp.: „000102“ (hex) = „AADB“ (base64)



- Primitive Datentypen (Forts.)
  - Zeitdauern (Basierend auf ISO 8601+Abweichungen)
    - `duration` Beginnt immer mit „P“, gefolgt von einer Liste von Paaren aus Werten und *designators*:  
 $PnYnMnDTnHnMnS$   
 P-Teil: Y=*years*, M=*months*, D=*days*;  
 T-Teil: H=*hours*, M=*minutes*, S=*seconds*

Weitere Regeln und Beispiele zu *duration*:

n = *integer*, außer bei S (dort *decimal* erlaubt)

Teile mit n=0 können entfallen

Der ganze T-Teil kann ggf. entfallen, aber nicht „P“

`P1Y3M15DT2H50M3S` 1 Jahr 3 Monate 15 Tage 2 Std ... 3 Sek.

`-P120D` -120 Tage (man beachte das Vorzeichen)

`PT2H59M5.6S` eine gute Marathon-Zeit ...



- Primitive Datentypen (Forts.)
  - Zeitpunkte (auch basierend auf ISO 8601+Abweichungen)
    - `date` Im Format CCYY-MM-DD. Beispiele:  
`2002-11-15` 15.11.2002  
`-0133-06-01` 1.6.133 v. Chr.
    - `time` Im Format hh:mm:ss (Sekunden auch *decimal*)  
`14:12:34.843` selbsterklärend

Besonderheiten

Alle Ziffern (auch führende, auch Sekunden) müssen befüllt werden, ggf. mit „0“ - auch wenn sie nicht signifikant sind.

**Zeitzoneangaben** - per Postfix wie folgt

`15:20:00Z` 15 Uhr 20 UTC / GMT

`15:20:00+01:00` 15 Uhr 20 in unserer Zeitzone

`15:20:00+03:30` 15 Uhr 20 Teheran-Zeit



## Vordefinierte Datentypen



- Primitive Datentypen (Forts.)
  - Zeitpunkte (Forts.)
    - `dateTime` Kombination aus date und time, separiert durch „T“. Beispiele:  
`2002-11-15T08:45:23`  
`2002-11-15T08:45:23-05:00`
  - Regelmäßig wiederkehrende Zeitpunkte
    - `gDay` Tag im Monat, Format: ---DD  
`---08` Jeder 8. Tag eines Monats
    - `gMonth` Monat im Jahr, Format: --MM  
`--06` Juni
    - `gMonthDay` Kombination, Format: --MM-DD  
`--12-24` Heiligabend



## Vordefinierte Datentypen



- Primitive Datentypen (Forts.)
  - Weitere, nun aber bestimmte Zeitpunkte
    - `gYear` Ein bestimmtes Jahr, Format: CCYY  
`2003` Aktuelles Jahr  
`-0333` Jahr der Schlacht bei Issos
    - `gYearMonth` Monat im Jahr, Format: CCYY-MM  
`2003-06` Juni des aktuellen Jahres
  - Abweichungen von ISO 8601
    - Minuszeichen erlaubt unmittelbar vor Werten von:
      - duration, dateTime, date, gMonth, gYear
    - Kein Jahr Null
      - Der Jahreswert „0000“ ist nicht zulässig
    - Jenseits von Jahr 9999
      - dateTime, date, gYearMonth, und gYear akzeptieren auch mehr als 4-stellige Jahreswerte, gemäß ISO 8601 *Draft Revision*.



## Vordefinierte Datentypen



- Abgeleitete Datentypen
  - Eingeschränkte numerische Datentypen
    - `integer` decimal ohne Bruch-Anteil
    - `positiveInteger` integer > 0
    - `negativeInteger` integer < 0
    - `nonPositiveInteger` integer <= 0
    - `nonNegativeInteger` integer >= 0
  - Computer-Wortlängen
    - `byte, unsignedByte` 8-bit, -128 ... 127 bzw. 0 ... 255
    - `short, unsignedShort` 16-bit, -32768 ... 32767 bzw. 0 ... 65535
    - `int, unsignedInt` 32-bit, analog (s. XML Schema-2 3.3)
    - `long, unsignedLong` 64-bit, analog



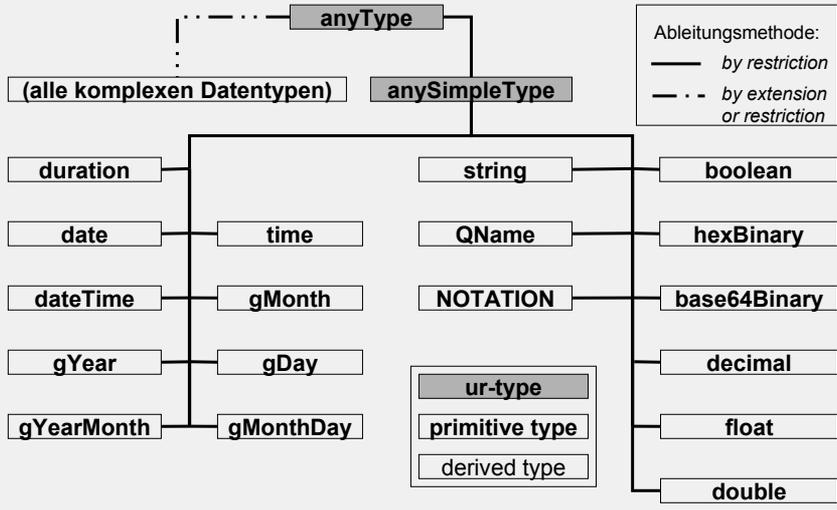
## Vordefinierte Datentypen



- Abgeleitete Datentypen (Forts.)
  - Abgeleitete XML-Attributtypen
    - `ID` Analog XML 1.0 Attributtyp ID, entsprechend:  
`IDREF, IDREFS`  
`ENTITY, ENTITIES`  
`NMTOKEN, NMTOKENS`
  - Andere XML-Konstrukte
    - `Name` Gemäß XML 1.0 „Name“-Regel
    - `language` Gleiche Werte wie XML 1.0-Attribut `xml:lang`
    - `NCName` „No colon name“ - Gegenstück zu `QName`
    - `normalizedString` weist den XML-Prozessor an, den Stringinhalt zu normieren, analog zur Normierung von CDATA-Attributwerten (*white space* zu *space*)
    - `token` strengere Normierung, analog zu `NMTOKEN`



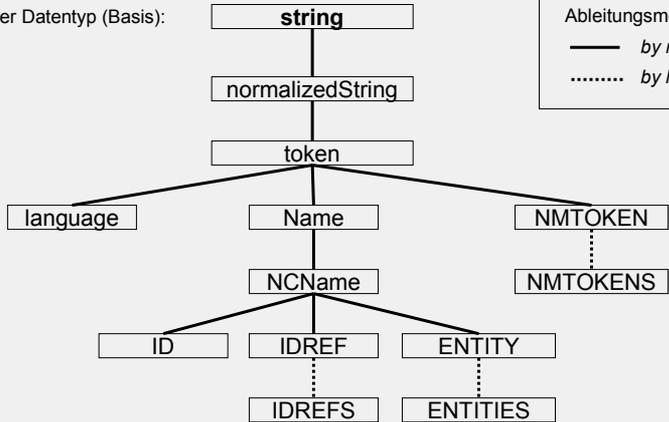
# Abstammung der primitiven Datentypen



# Vordefinierte Datentypen

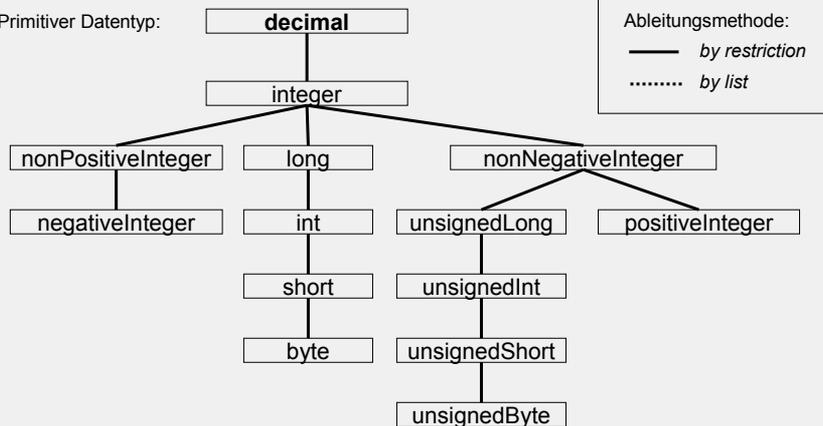


Primitiver Datentyp (Basis):





Primitiver Datentyp:



- Benutzung der vordefinierten Datentypen
  - Es ist möglich, auch ohne Verweis auf ein XML Schema die in W3C XML Schema vordefinierten Datentypen zu referenzieren.
  - Dazu verwende man das globale Attribut **type** aus dem Schemainstanz-Namensraum

<http://www.w3.org/2001/XMLSchema-instance>

sowie folgenden Namensraum für die Datentypen:

<http://www.w3.org/2001/XMLSchema-datatypes>



## Vordefinierte Datentypen



- Benutzung, Beispiel:
  - Datentypen-Information direkt aus dem Instanzdokument an die Anwendung, ohne Schema-Validierung:

```
<doc xmlns:xsi=
  "http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema-
  datatypes">
  <mynum xsi:type="xsd:decimal">409</mynum>
  <mystr xsi:type="xsd:string">
  This is my string</mystr>
</doc>
```

- Etwa beim Aufbau dynamischer Datenstrukturen auch ohne Schema, bei Verwendung der Datentypen durch andere Schemasprachen, etc.



## Datatypes for DTD (DT4DTD)



- Aufgabe: Nutzung von XML Schema-Datentypen in DTDs
- Problem: DTDs sind älter als XML-Schema und nicht änderbar.
- Lösungsansatz:
  - Per Konvention
  - Vergabe spezieller (globaler) Attributnamen
    - e-dtype** Element-Datentyp (falls Inhalt nicht gleich „string“)
    - a-dtype** Attribut-Datentyp, listenwertig
  - Ohne Namespace-Präfix (wird implizit unterstellt)
  - Per NOTATION erweiterbar auf selbst definierte Datentypen
  - Beispiel: Siehe nächste Seite
- Bemerkungen:
  - Keine offizielle W3C-Empfehlung, aber eine „Note“, siehe <http://www.w3.org/TR/dt4dtd> (bzw. die lokale Kopie).
  - Unterstützt z.B. von *Java Arch. for XML Binding (JAXB)*



- Beispiel (aus „Ch. Goldfarb’s XML Handbook“)

```
<!NOTATION pubYear SYSTEM "datatypeDefs.xml#pubYear">
<!ATTLIST poem
    a-dtype          CDATA #FIXED
                    "pubyear pubYear
                    linecount nonNegativeInteger"
    pubyear          CDATA #IMPLIED
    linecount        CDATA #IMPLIED >
```

  - e-dtype wird hier nicht verwendet
  - Dem Attribut „a-dtype“ wird #FIXED (!) ein String zugewiesen.
  - Dieser String besteht aus einer Liste von Paaren  
Jedes Paar besteht aus einem Attributnamen des vorliegenden Elements und aus dessen zugewiesenen Datentyp.
  - Der Datentyp „nonNegativeInteger“ verweist auf XML Schema
  - Der benutzerdefinierte Typ „pubYear“ verwendet eine Hilfskonstruktion, auf die eine Anwendung reagieren könnte.



- Wirklich leistungsfähig werden die Datentypen von XML Schema erst durch die Möglichkeit, eigene Datentypen abzuleiten.
- Zusammengesetzte (komplexe) Datentypen, insb. benötigt zur Deklaration von Elementen, sind Gegenstand eines späteren Abschnitts.
- Zur Ableitung von einfachen Datentypen (wie sie auch Attribute annehmen können), stellt W3C XML Schema das Element „simpleType“ zur Verfügung.
- Es nimmt Bezug auf einen Basistyp - vordefiniert oder benutzerdefiniert - und wendet eine von drei Ableitungsmethoden an.



- Die drei Methoden der Ableitung
  - **by list**  
Ein Element des Listentyps ist eine Folge (*sequence*) von Elementen der zugrundeliegenden Wertemenge des *itemType*.
  - **by union**  
Vereinigungsmenge (von W und L) bilden
  - **by restriction**  
Die 12 Facetten (in 6 Kategorien) der Einschränkung:
    - Länge: ***length, minLength, maxLength***
    - Muster: ***pattern***
    - Aufzählung: ***enumeration***
    - Whitespace: ***whitespace***
    - Intervall (*range*): ***minInclusive, minExclusive, maxExclusive, maxInclusive***
    - Dezimalstellen: ***totalDigits, fractionDigits***



- Beispiel:
  - Eine Liste von Größenangaben mit dem Basistyp `decimal`

```
<simpleType name='sizes'>  
  <list itemType='decimal' />  
</simpleType>
```
  - Anwendung dann:

```
<cerealSizes xsi:type='sizes'> 8 10.5 12  
</cerealSizes>
```
- Neu im Beispiel:
  - `<list>` mit Attribut „`itemType`“
- **Vorsicht:**
  - Listenelemente werden mit *whitespace* separiert. Enthält der *itemType* *whitespace* als zulässige Zeichen, lässt sich die Liste nicht immer in ihre korrekten Bestandteile zerlegen!



## Ableitung *by union*



- Beispiel:
  - Die Menge  $Z^*$  (alle ganzen Zahlen außer Null)

```
<simpleType name='z-asterisk'>  
  <union>  
    <simpleType>  
      <restriction base="positiveInteger"/>  
    </simpleType>  
    <simpleType>  
      <restriction base="negativeInteger"/>  
    </simpleType>  
  </union>  
</simpleType>
```

- Neu im Beispiel:
  - `<union>` unterhalb vom zu definierenden `simpleType`



## Ableitung *by restriction*



- Ableiten durch Einschränkung erfolgt durch Einwirkung der erwähnten einschränkenden Facetten auf einen Basisdatentyp.
- Man kann von vordefinierten und auch eigenen Datentypen ableiten, zunächst natürlich nur von den vordefinierten.
- Nicht jede Facette ist auf jeden vordefinierten Datentypen anwendbar.
  - Meist ergibt sich dies schon aus dem Kontext.
  - Einzelheiten: Siehe Tabelle in Kap. 4.1.5 von W3C XML Schema: Datatypes



## Ableitung *by restriction*



- Typische Konstruktion beim Ableiten:

```
<simpleType name='myRestrictedType'>  
  <restriction base='baseType'>  
    facet 1 ...  
    ...  
    facet n ...  
  </restriction>  
</simpleType>
```

- Neu im Beispiel:
  - <restriction> mit Attribut „base“
  - Diverse Facetten-Elemente



## Ableitung *by restriction*



<length>, <minLength>, <maxLength>

- Alle Listentypen: Länge der Liste (Zahl der Einträge)
- string und abgeleitete: Anzahl characters (nicht: Bytes!)
- hexBinary, base64Binary: Anzahl Oktets der Binärdarstellung

```
<simpleType name='dreiBytes'>  
  <restriction base='hexBinary'>  
    <length value='3' />  
  </restriction>  
</simpleType>  
<simpleType name='KfzKzStadt'>  
  <restriction base='token'>  
    <minLength value='1' />  
    <maxLength value='3' />  
  </restriction>  
</simpleType>
```



## Ableitung by restriction



<totalDigits>, <fractionDigits>

- decimal und abgeleitete: Gesamtzahl Stellen und Anzahl Nachkommastellen

```
<simpleType name='geldBetrag'>
  <restriction base='decimal'>
    <totalDigits value='8' />
    <fractionDigits value='2' fixed='true' />
  </restriction>
</simpleType>
```

Zulässige Werte sind dann etwa:

-123456.78, 2.70, +1.00

Unzulässig:

1234567.89, 2.7, 25



## Ableitung by restriction



<minInclusive>, <maxInclusive>, <minExclusive>, <maxExclusive>

- Alle Datentypen mit „geordneten“ Wertemengen (d.h. es gibt eine Ordnungsrelation „>“ auf W), insb. die numerischen Datentypen, Zeiten und Intervalle - aber nicht Stringtypen oder Listen.

```
<simpleType name='AlterEinerMinderjährigenPerson'>
  <restriction base='nonNegativeInteger'>
    <maxExclusive value='18' />      <!-- x < 18 -->
  </restriction>
</simpleType>
<simpleType name='wasserTempFluessig'>
  <restriction base='decimal'>
    <minInclusive value='0' />      <!-- t >= 0 -->
    <maxInclusive value='100' />    <!-- t <= 100 -->
  </restriction>
</simpleType>
```



## Ableitung *by restriction*



### <enumeration>

- Reduzierung der Wertemenge des Basistyps auf die explizit gelisteten Werte-Elemente. Praktisch immer möglich außer bei `boolean`.

```
<simpleType name='unbeweglicheFeiertage'>
  <restriction base='gMonthDay'>
    <enumeration value='--01-01' />
    <enumeration value='--05-01' />
    <enumeration value='--10-03' />
    <enumeration value='--12-24'>
      <annotation><documentation>
        Halber Tag</documentation></annotation>
    </enumeration>
    <enumeration value='--12-25' />
    <enumeration value='--12-26' />
    <enumeration value='--12-31'> ... </enumeration>
  </restriction>
</simpleType>
```



## Ableitung *by restriction*



### <whiteSpace>

- Eigentlich keine Facette zur Einengung der Wertemenge, sondern eine einengende Anweisung an den XML-Prozessor
- Nur drei gültige Werte:
  - **preserve**: *whitespace* wird nicht verändert
  - **replace**: *whitespace* wird zu *Space*, wie bei CDATA-Attributen
  - **collapse**: *whitespace*-Normierung wie bei NMTOKEN-Attributen
- Die meisten vordefinierten Datentypen verwenden *collapse*, außer *string* und einige davon abgeleitete.

```
<simpleType name='eineZeileText'>
  <restriction base='string'>
    <whiteSpace value='replace' />
    <maxLength value='135' />
  </restriction>
</simpleType>
```



### <pattern>

- Sehr flexible und mächtige Ableitungsmethode, basierend auf „regulären Ausdrücken“. Mit allen einfachen Datentypen verwendbar.

```
<simpleType name='Bankleitzahl'>
  <restriction base='nonNegativeInteger'>
    <pattern value='\d{8}' />
  </restriction>
</simpleType>
```

```
<simpleType name='KFZ-Kennzeichen'>
  <restriction base='token'>
    <pattern value='[A-ZÄÖÜ]{1,3}-[A-ZÄÖÜ]{1,3} \d{1,4}' />
    <maxLength value='8' />
  </restriction>
</simpleType>
```



- Reguläre Ausdrücke:
  - Ähnlich zu - aber nicht gleich - den regulären Ausdrücken aus den Scriptsprachen **Perl** oder **Ruby**.
  - Vollständig definiert in XML Schema Teil 2 (*Datatypes*).
  - Eine Sammlung verschiedener Beispiele für reguläre Ausdrücke findet man in XML Schema Teil 0 (*Tutorial*), Tabelle D1.
  - Reguläre Ausdrücke sind nicht Gegenstand dieser Vorlesung, sondern werden vorausgesetzt.
  - Hier werden nur Ergänzungen zu Perl aufgeführt.
- Empfehlung:
  - Prüfen Sie Ihre Kenntnisse zu Regulären Ausdrücken mittels der o.g. Tabelle D1.
  - Holen Sie Lücken in Ihrem Repertoire nach, z.B. durch Lesen der Spezifikationen in Teil 2 von XML Schema.



## Ableitung *by restriction*



- Reguläre Ausdrücke und Unicode:
  - Die bisher eingebauten Kurzschreibweisen für bestimmte Zeichenmengen wie `\d` für Ziffer, `\s` für *whitespace* etc. benötigen für Unicode einige Erweiterungen:
  - Kategorienbildung:
    - L *Letters*
    - M*Marks*
    - N *Numbers*
    - P *Punctuation*
    - S *Separators*
    - O *Other*
  - Eigenschaften, spezifisch für jede Zeichenkategorie, z.B.:
    - u *uppercase* (bei L)
    - s *space* (bei S)
  - Definition von Codeblöcken, angesprochen über Namen, z.B.:
    - Greek



## Ableitung *by restriction*



- Reguläre Ausdrücke und Unicode:
  - Mit `\p{}` lassen sich nun verschiedenste Teilmengen von Unicode selektieren. Beispiele:
    - `\p{Lu}` Ein beliebiger Großbuchstabe
    - `\p{Sc}` Ein beliebiges Währungscodezeichen, etwa ‚€‘
    - `\p{IsGreek}` Ein Zeichen aus dem Codeblock „Greek“
    - `\P{IsGreek}` Kein Zeichen aus dem Codeblock „Greek“
  - Weitere spezielle Zeichensequenzen:
    - `\s, \S` `[\#x20,\t,\n,\r], [^\s]` (aber siehe auch: XML 1.1)
    - `\i, \I` `Letter | ‘ ’ | ‘:’, [^\i]` (*initial name letter*)
    - `\c, \C` `NameChar, [^\c]` (vgl. XML 1.0)
    - `\d, \D` `\p{Nd}, [^\d]` (Dezimalziffern)
    - `\w, \W` alle außer `[\p{P}, \p{Z}, \p{C},.]`, `[\w]`  
(also keine Interpunktionszeichen, Separatoren, oder aus der Kategorie „Andere“)



# Strukturen mit XML Schema

Aufbau komplexer Elemente  
Ausgewählte Themen zu XML Schema  
(Noch unvollständig)



## XML Schema: Strukturen



Ein XML Schema besteht aus:

- 13 verschiedenen Schema-Komponenten
- gebildet aus 3 Komponentengruppen.
- **4 Primäre Komponenten**
  - einfache und komplexe Typendef.: `<simpleType>`, `<complexType>`
  - Attribut- und Elementdeklarationen: `<attribute>`, `<element>`
  - Bem.: teils mit Namen, teils „anonym“
- **4 Sekundäre Komponenten**
  - Attribut- und Modellgruppen-Def.: `<attributeGroup>`, `<group>`
  - *notation*-Deklarationen, *identity-constraint definitions* (übersetzen!)
  - Bem.: stets mit Namen versehen
- **5 Helfer-Komponenten**
  - Anmerkungen, Modellgruppen, Partikel, *wildcards*, Attribut-Verwendungen / kontextabhängig.



# Die XML-Darstellung der 13 Schemakomponenten

Kurzbeschreibung ihrer Attribute und Inhalte  
Details in *XML Schema: Structures* Kap. 3,  
Erläuterungen mündlich in der Vorlesung!



## <schema>



```
<schema
  attributeFormDefault = (qualified | unqualified) : unqualified
  blockDefault = (#all | List of (extension | restriction |
                                substitution)) : ''
  elementFormDefault = (qualified | unqualified) : unqualified
  finalDefault = (#all | List of (extension | restriction)) : ''
  id = ID
  targetNamespace = anyURI
  version = token
  xml:lang = language
  {any attributes with non-schema namespace . . .}>

Content:
  ((include | import | redefine | annotation)*,
   (((simpleType | complexType | group | attributeGroup) |
    element | attribute | notation), annotation*))

</schema>
```



## <simpleType>

10

```
<simpleType
  final = (#all | (list | union | restriction))
  id = ID
  name = QName
  {any attributes with non-schema namespace . . .}>
Content: (annotation?, (restriction | list | union))
</simpleType>
```

```
<restriction
  base = QName
  id = ID
  {any attributes with non-schema namespace . . .}>
Content: (annotation?, (simpleType?, (minExclusive |
minInclusive | maxExclusive | maxInclusive |
totalDigits | fractionDigits | length | minLength |
maxLength | enumeration | whiteSpace | pattern)*))
</restriction>
```



## <simpleType>

10

```
<list
  id = ID
  itemType = QName
  {any attributes with non-schema namespace . . .}>
Content: (annotation?, (simpleType?))
</list>
```

```
<union
  id = ID
  memberTypes = List of QName
  {any attributes with non-schema namespace . . .}>
Content: (annotation?, (simpleType*))
</union>
```



## <simpleType>



Beispiel (mit Ableitungstyp „restriction“):

```
<xs:simpleType name="celsiusWaterTemp">
  <xs:restriction base="xs:number">
    <xs:fractionDigits value="2"/>
    <xs:minExclusive value="0.00"/>
    <xs:maxExclusive value="100.00"/>
  </xs:restriction>
</xs:simpleType>
```



## <complexType>



```
<complexType
  abstract = boolean : false
  block = (#all | List of (extension | restriction))
  final = (#all | List of (extension | restriction))
  id = ID
  mixed = boolean : false
  name = NCName
  {any attributes with non-schema namespace . . .}>

Content:
  (annotation?,
   (simpleContent | complexContent |
    ((group | all | choice | sequence)?,
     ((attribute | attributeGroup)*, anyAttribute?))))
</complexType>
```



### Standardbeispiel in Kurzschreibweise:

```
<xs:complexType name="PurchaseOrderType">
  <xs:sequence>
    <xs:element name="shipTo" type="USAddress"/>
    <xs:element name="billTo" type="USAddress"/>
    <xs:element ref="comment" minOccurs="0"/>
    <xs:element name="items" type="Items"/>
  </xs:sequence>
  <xs:attribute name="orderDate"
    type="xs:date"/>
</xs:complexType>
```



### Explizite Datentyp-Ableitung:

```
<xs:complexType name="length2">
  <xs:complexContent>
    <xs:restriction base="xs:anyType">
      <xs:sequence>
        <xs:element name="size"
          type="xs:nonNegativeInteger"/>
        <xs:element name="unit" type="xs:NMTOKEN"/>
      </xs:sequence>
    </xs:restriction>
  </xs:complexContent>
</xs:complexType>

<xs:element name="depth" type="length2"/>

<depth> <size>25</size><unit>cm</unit> </depth>
```



## <complexType>



Erweiterung eines einfachen Datentyps um ein Attribut:

```
<xs:complexType name="length1">
  <xs:simpleContent>
    <xs:extension base="xs:nonNegativeInteger">
      <xs:attribute name="unit" type="xs:NMTOKEN"/>
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>

<xs:element name="width" type="length1"/>

<width unit="cm">25</width>
```



## <element>



```
<element
  abstract = boolean : false
  block = (#all | List of (extension | restriction | substitution))
  default = string
  final = (#all | List of (extension | restriction))
  fixed = string
  form = (qualified | unqualified)
  id = ID
  maxOccurs = (nonNegativeInteger | unbounded) : 1
  minOccurs = nonNegativeInteger : 1
  name = NCName
  nillable = boolean : false
  ref = QName
  substitutionGroup = QName
  type = QName
  {any attributes with non-schema namespace . . .}>
Content:
  (annotation?,
  ((simpleType | complexType)?, (unique | key | keyref)*))
</element>
```



Beispiel:

```
<xs:element name="PurchaseOrder"
  type="PurchaseOrderType" />

<xs:element name="gift">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="birthday" type="xs:date" />
      <xs:element ref="PurchaseOrder" />
    </xs:sequence>
  </xs:complexType>
</xs:element>
```



- Einige Anmerkungen

- Attribute „abstract“, „substitutionGroup“:

XML Schema ermöglicht die Bildung abstrakter (virtueller, nicht instanzierbarer) Elementtypen.

Diese fungieren als Kopf einer *substitution group* aus abgeleiteten, im Detail unterschiedlichen Elementtypen.

Spezifiziert man den abstrakten Elementtyp z.B. in einer <sequence> eines neuen Elementtyps, so kann \*anstelle\* des abstrakten Elementtyps ein beliebiges konkretes Mitglied der *substitution group* erscheinen.

- Attribute „block“, „final“

Modifiziert *substitutions* - hier nicht weiter besprochen.

- Attribut „nillable“

Erlaubt/untersagt die Verwendung des globalen Attributs xsi:nil



## <group>

10

```
<group
  name = NCName
  Content: (annotation?,
            (all | choice | sequence))
</group>
```



## <group>

10

Beispiel:

```
<xs:group name="myModelGroup">
  <xs:sequence>
    <xs:element ref="someThing"/>
    . . .
  </xs:sequence>
</xs:group>

<xs:complexType name="trivial">
  <xs:group ref="myModelGroup"/>
  <xs:attribute .../>
</xs:complexType>
```



## <group>



Beispiel (Forts.):

```
<xs:complexType name="moreSo">
  <xs:choice>
    <xs:element ref="anotherThing"/>
    <xs:group ref="myModelGroup"/>
  </xs:choice>
  <xs:attribute .../>
</xs:complexType>
```



## <attribute>



```
<attribute
  default = string
  fixed = string
  form = (qualified | unqualified)
  id = ID
  name = NCName
  ref = QName
  type = QName
  use = (optional | prohibited | required) : optional
  {any attributes with non-schema namespace ...}>
Content: (annotation?, (simpleType?))
</attribute>
```

Beispiel:

```
<xs:attribute name="age"
  type="xs:positiveInteger" use="required"/>
```



## <attribute>



- Bereits besprochen / in Beispielen erläutert:
  - name, type, use, default, fixed
- Weitere Attribute:
  - form Vgl. Attribut „elementFormDefault“ des Elements „schema“ - hier nicht besprochen
  - id stets möglich, erleichtert z.B. Suchen
  - ref bei Ableitungen, ersetzt dann name, form, type, <simpleType>
- Content
  - annotation stets möglich
  - simpleType zur Definition eines anonymen Datentyps mit nur lokaler Reichweite, z.B. als Ersatz für die Verwendung von Attribut type)



## <attributeGroup>



```
<attributeGroup
  id = ID
  name = NCName
  ref = QName
  {any attributes with non-schema namespace ...}>
Content:
(annotation?,
 ((attribute | attributeGroup)*,
  anyAttribute?))
</attributeGroup>
```



## <attributeGroup>



Beispiel:

```
<xs:attributeGroup name="myAttrGroup">
  <xs:attribute .../>
  ...
</xs:attributeGroup>

<xs:complexType name="myelement">
  ...
  <xs:attributeGroup ref="myAttrGroup"/>
</xs:complexType>
```



## <all>, <choice>, <sequence>



```
<all
  id = ID
  maxOccurs = 1 : 1
  minOccurs = (0 | 1) : 1
  {any attributes with non-schema namespace . . .}>
  Content: (annotation?, element*)
</all>

<choice
  id = ID
  maxOccurs = (nonNegativeInteger | unbounded) : 1
  minOccurs = nonNegativeInteger : 1
  {any attributes with non-schema namespace . . .}>
  Content: (annotation?,
  (element | group | choice | sequence | any)*)
</choice>
```



```
<sequence
  id = ID
  maxOccurs = (nonNegativeInteger | unbounded) : 1
  minOccurs = nonNegativeInteger : 1
  {any attributes with non-schema namespace . . .}>
Content:
  (annotation?,
   (element | group | choice |
    sequence | any)*)
</sequence>
```

<a11>: Alle gelisteten Elemente müssen in der Instanz enthalten sein, aber die Reihenfolge spielt keine Rolle.



Beispiel:

```
<xs:all>
  <xs:element ref="cats"/>
  <xs:element ref="dogs"/>
</xs:all>

<xs:sequence>
  <xs:choice>
    <xs:element ref="left"/>
    <xs:element ref="right"/>
  </xs:choice>
  <xs:element ref="landmark"/>
</xs:sequence>
```



## <any>



```
<any
  id = ID
  maxOccurs = (nonNegativeInteger | unbounded) : 1
  minOccurs = nonNegativeInteger : 1
  namespace = ((##any | ##other) | List of (anyURI |
    (##targetNamespace | ##local))) : ##any
  processContents = (lax | skip | strict) : strict
  {any attributes with non-schema namespace . . .}>
```

**Content:** (annotation?)

```
</any>
```

- Verwendbar nur innerhalb eines *content models*
- Gestattet es, \*beliebige\* Elemente z.B. aus einem fremden Namensraum im gegebenen Element zuzulassen. Daher auch „*element wildcard*“.



## <notation>



```
<notation
  id = ID
  name = NCName
  public = anyURI
  system = anyURI
  {any attributes with non-schema namespace . . .}>
Content: (annotation?)
</notation>
```

Beispiel:

```
<xs:notation name="jpeg" public="image/jpeg"
  system="viewer.exe" />
```

- Verwendung z.B. so:

```
<xs:simpleType <xs:restriction base="xs:NOTATION">
  <xs:enumeration value="jpeg"/> ...
```



# Anhang

Noch nicht besprochenes  
Material, teils noch auszuarbeiten



## Noch zu besprechen



- **<unique>, <key>, <keyref>;**  
**<selector>, <field>**
  - sogenannte *identity constraint definition*-Komponenten
  - verallgemeinern das ID-Konzept, erfordern XPath-Kenntnisse; hier zurückgestellt.
- **<annotation>** (bestehend aus **<appinfo>, <documentation>**)
  - Zu Dokumentationszwecken, an vielen Stellen verfügbar. **Rege Verwendung empfohlen!**
- **<include>, <import>, <redefine>**
  - Mechanismen zur Modularisierung und Mehrfachverwertung mehrerer Schemata.
  - **include:** Einbinden globaler Deklarationen in den aktuellen Namensraum,
  - **import:** *Verweisen auf* globale Deklarationen, auch aus anderen Namensräumen,
  - **redefine:** wie der Name schon sagt ...



- **Teilweise noch auszuarbeiten:**
  - Ableitungen von *complex types* per *extension*  
nur Anfügen (zum Urtyp *anyType*) möglich.
  - *element substitution groups* (gab es in DTD nicht)
  - *model groups: sequence, conjunction, disjunction*
  - *particle: element decl., wildcard, model group*
  - *attribute use (= default decl.?)*
  - wildcards: Identifizieren *element* bzw. *attribute info items* nach ihrem *namespace*-Namen, ignorieren *local names*.
  - *levels of conformance (of processors)*
  - *xsi:nil, xsi:schemaLocation, xsi:noNamespaceSchemaLocation*
  - Unterscheide: *Schema component, XML representation* („XSDL“)
- **Ergänzungen**
  - Schema der Schemata, DTD der Schemata
  - `<annotation>` als stets optionales erstes Element