

Kapitel WT:III

III. Dokumentensprachen

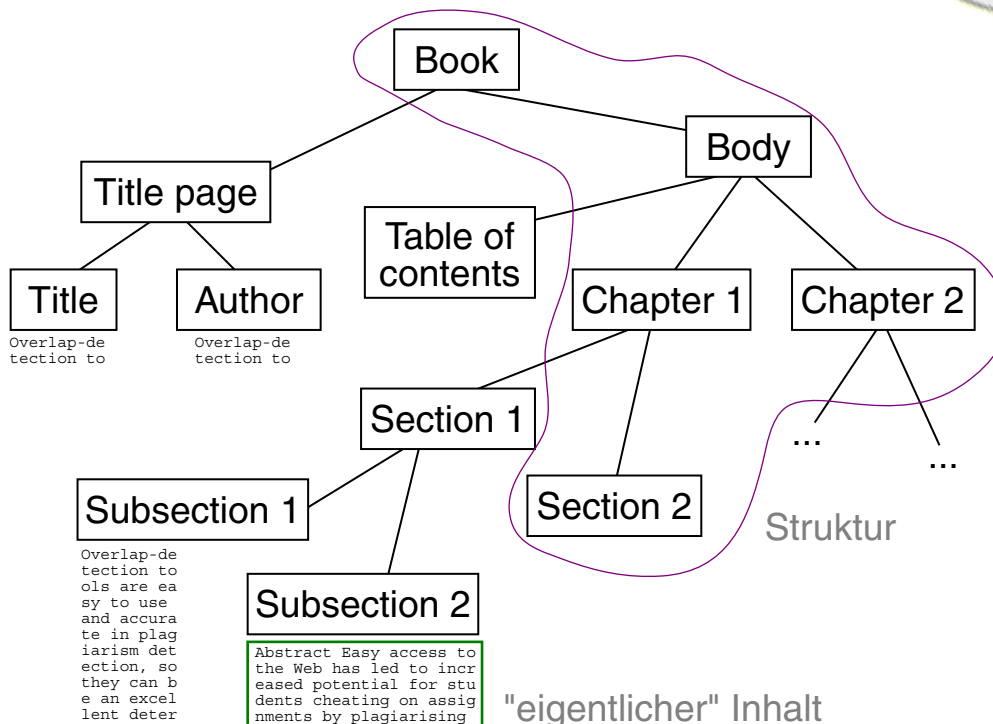
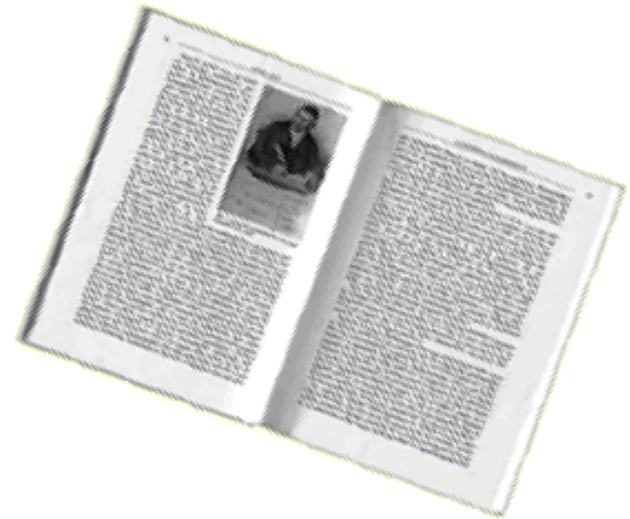
- ❑ Auszeichnungssprachen
- ❑ HTML
- ❑ Cascading Stylesheets CSS
- ❑ XML-Grundlagen
- ❑ XML-Schema
- ❑ Die XSL-Familie
- ❑ APIs für XML-Dokumente

Auszeichnungssprachen

Einführung

Trennung von

1. Dokument **struktur**
2. Dokument **inhalt**
3. Dokument **darstellung** bzw. Layout



Darstellung

<i>Blackletter ITC</i>	ACTION IS	<i>Informal Roman</i>
<i>Brush Script MT</i>	Andy	Jokerman
Bradley Hand ITC	Arial Black	Jetice ITC
Carlz MT	Bauhaus 93	Kindergarten
<i>Edwardian Script ITC</i>	Bell MT	Kristen ITC
<i>French Script ITC</i>	Broadway	Maiandra GD
<i>Freestyle Script</i>	CASTELLAR	Modern No. 20
Gigi	Comic Sans MS	Papyrus
Harrington	Cooper Black	Parade
<i>Harlow Solid Italic</i>	COPPERPLATE	Revue
<i>Monotype Corsiva</i>	Enviro	Sirona
Matruxa MT	Forté	Tempus Sans ITC
Pristina	Footlight MT Light	ThinkerToy
<i>Shelley Delante BT</i>	Irish	Verdana
<i>Strada</i>	High Tower Text	VIKING

Auszeichnungssprachen

Einführung

Beispiel \LaTeX :

```
\documentclass{llncs}
\usepackage[T1]{fontenc}
\usepackage[german,american]{babel}
\usepackage{graphicx}

\begin{document}

\title{Fuzzy Fingerprints for Near Similarity Search}
\titlerunning{Fuzzy Fingerprints\ldots}

\author{Benno Stein\inst{1}}
\institute{Faculty of Media, Media Systems}

\maketitle

\begin{abstract}
This paper introduces a particular form of fuzzy-fingerprints--their
construction, their interpretation, and their use in the field of
information retrieval.
\end{abstract}
```

...

Auszeichnungssprachen

Einführung

Beispiel \LaTeX :

```
\documentclass{llncs}
\usepackage[T1]{fontenc}
\usepackage[german,american]{babel}
\usepackage{graphicx}

\begin{document}

\title{Fuzzy Fingerprints for Near Similarity Search}
\titlerunning{Fuzzy Fingerprints for Near Similarity Search}

\author{Benno Stein\ir
\institute{Faculty of

\maketitle

\begin{abstract}
This paper introduces
construction, their in
information retrieval.
\end{abstract}

...
```

Fuzzy Fingerprints for Near Similarity Search

Benno Stein¹

Faculty of Media, Media Systems

Abstract This paper introduces a particular form of fuzzy-fingerprints—their construction, their interpretation, and their use in the field of information retrieval.

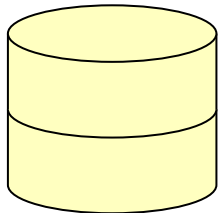
...

Auszeichnungssprachen

Einführung

Trennung von Struktur, Inhalt und Darstellung ermöglicht:

- ❑ Layout- und geräteunabhängige Archivierung
- ❑ maschinelle Analyse und Verarbeitung von Strukturinformation.
Beispiele: Indexe, Seitenzahlen, Verweise, Zitate
- ❑ **Single-Source-Prinzip**: die Änderung an *einer* Quelle wird in allen nachfolgenden Layout-Prozessen nachvollzogen
- ❑ Stichworte: *Database Publishing, Cross Media Publishing*



Bemerkungen:

- ❑ Mögliche Zielformate eines Layout-Prozesses:
 - Portable Document Format PDF
 - Postscript PS
 - Rich Text Format RTF
 - Extended Markup Language XML
 - Hypertext Markup Language HTML
 - Programm-Code
 - Hilfedateien [\[Wikipedia\]](#)

Auszeichnungssprachen

Einführung

Merkmale von Auszeichnungssprachen:

- ❑ Strukturinformation wird in den „eentlichen“ Inhalt integriert.
→ Metasprache zur Auszeichnung von Strukturinformation
- ❑ Auszeichnung = Markup
Auszeichnungssprache = Markup-Sprache
- ❑ Markup-Symbol (*Tag*) = Wort aus der Markup-Sprache;
insbesondere: Unterscheidung von Start-Tags und End-Tags

Auszeichnungssprachen

Einführung

Merkmale von Auszeichnungssprachen:

- ❑ Strukturinformation wird in den „eigentlichen“ Inhalt integriert.
→ Metasprache zur Auszeichnung von Strukturinformation
- ❑ Auszeichnung = **Markup**
Auszeichnungssprache = Markup-Sprache
- ❑ Markup-Symbol (*Tag*) = Wort aus der Markup-Sprache;
insbesondere: Unterscheidung von Start-Tags und End-Tags

Forderungen an Auszeichnungssprachen:

- ❑ Syntax und Semantik von Markup-Symbolen definierbar
- ❑ erweiterbar hinsichtlich neuer Strukturelemente und Dokumententypen
- ❑ von Menschen schreib- und lesbar
- ❑ einbettbar in Programmiersprachen
- ❑ offen für zukünftige Entwicklungen: neue Medientypen, Medienintegration

Auszeichnungssprachen

SGML

Historie:

60er einheitliches Datenformat soll Datenverarbeitung flexibler machen

70er C. Goldfarb entwickelt bei IBM die Generalized Markup Language GML

1986 Standardisierung von GML → SGML = Standard GML, ISO/IEC 8879

Auszeichnungssprachen

SGML

Historie:

60er einheitliches Datenformat soll Datenverarbeitung flexibler machen

70er C. Goldfarb entwickelt bei IBM die Generalized Markup Language GML

1986 Standardisierung von GML → SGML = Standard GML, ISO/IEC 8879

Konzepte von SGML:

1. SGML-Deklaration

Definiert Zeichenvorrat, Steuerzeichen, Auszeichnungsregeln für Parser.

2. SGML Document Type Definition, DTD (Dokumentenklasse)

Definiert die Elementtypen eines SGML-Dokuments, „gegen“ die der Parser analysiert. Die Elementtypen bilden einen Strukturbaum.

3. SGML-Dokument

Enthält eine Instanz des Strukturbaums gemäß einer DTD.
Die Blätter des Strukturbaums bilden den eigentlichen Inhalt.

Auszeichnungssprachen

SGML

Historie:

60er einheitliches Datenformat soll Datenverarbeitung flexibler machen

70er C. Goldfarb entwickelt bei IBM die Generalized Markup Language GML

1986 Standardisierung von GML → SGML = Standard GML, ISO/IEC 8879

Konzepte von SGML:

1. SGML-Deklaration

Definiert Zeichenvorrat, Steuerzeichen, Auszeichnungsregeln für Parser.

2. SGML Document Type Definition, DTD (Dokumentenklasse)

Definiert die Elementtypen eines SGML-Dokuments, „gegen“ die der Parser analysiert. Die Elementtypen bilden einen Strukturbaum.

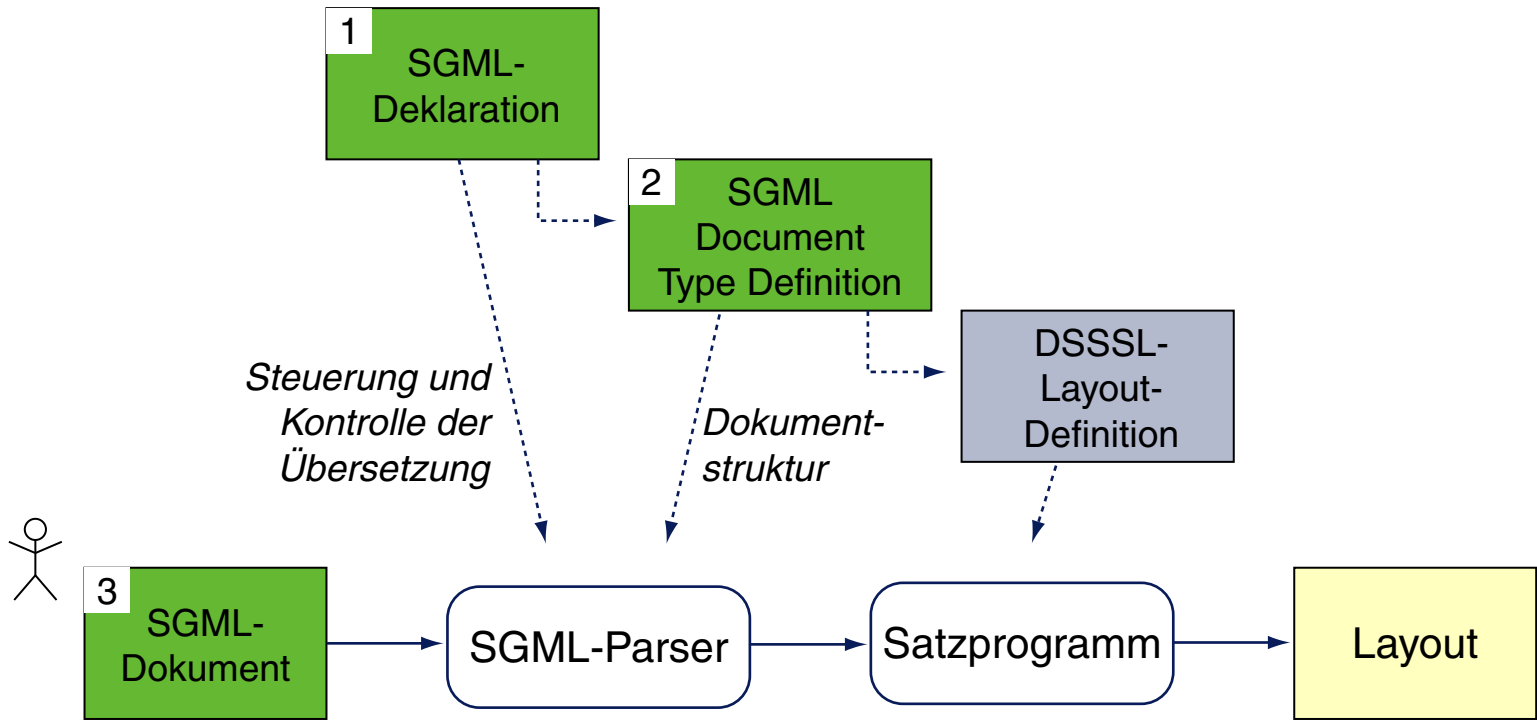
3. SGML-Dokument

Enthält eine Instanz des Strukturbaums gemäß einer DTD.

Die Blätter des Strukturbaums bilden den eigentlichen Inhalt.

Auszeichnungssprachen

SGML Dokumentenverarbeitung



Bemerkungen:

- ❑ Das Formatieren ist nicht Bestandteil von SGML.
- ❑ Für Layout-spezifische und geräteabhängige Definitionen zur Darstellung der in SGML beschriebenen Strukturelemente wurde eine spezielle Sprache, die *Document Style Semantics and Specification Language* DSSSL („Dissel“ ausgesprochen) entwickelt. [\[Wikipedia, DSSSL-Portal\]](#)
- ❑ Mittlerweile entwickelt sich Cascading Style Sheets (ab Level 3, CSS3) zu einer Alternative sowohl für DSSSL als auch für Stylesheetsprachen wie XSL-FO. [\[Wikipedia\]](#)

Auszeichnungssprachen

SGML-Dokument

Allgemeine Form einer Element**instanz** [WT:III HTML] :

```
<elementname {attribute}*> ... </elementname>
```

Beispiel:

```
<table width="100%" cellspacing="0" cellpadding="0" border="0">  
...  
</table>
```

Auszeichnungssprachen

SGML-Dokument

Allgemeine Form einer Element**instanz** [WT:III HTML] :

```
<elementname {attribute}*> ... </elementname>
```

Beispiel:

```
<table width="100%" cellspacing="0" cellpadding="0" border="0">
...
</table>
```

Abstrakte Syntax für Markup-Symbole (*Tags*):

```
start-tag ::= stag-open identifier {attribute}* tag-close
end-tag   ::= etag-open identifier tag-close
attribute ::= identifier = "value"
```

Konkrete Syntax für Markup-Symbole (festgelegt in der SGML-Deklaration):

```
stag-open   ::= <
etag-open   ::= </
tag-close   ::= >
identifier  ::= {xchar}+
value       ::= {char}+
```

Auszeichnungssprachen

SGML-Dokument

Allgemeine Form einer Element**instanz** [WT:III HTML] :

```
<elementname {attribute}*> ... </elementname>
```

Beispiel:

```
<table width="100%" cellspacing="0" cellpadding="0" border="0">  
...  
</table>
```

Abstrakte Syntax für Markup-Symbole (*Tags*):

```
start-tag ::= stag-open identifier {attribute}* tag-close  
end-tag    ::= etag-open identifier tag-close  
attribute  ::= identifier = "value"
```

Konkrete Syntax für Markup-Symbole (festgelegt in der SGML-Deklaration):

```
stag-open   ::= <  
etag-open  ::= </  
tag-close  ::= >  
identifier ::= {xchar}+  
value      ::= {char}+
```


Auszeichnungssprachen

SGML-Dokument

Allgemeine Form einer Element**instanz** [WT:III HTML] :

```
<elementname {attribute}*> ... </elementname>
```

Beispiel:

```
<table width="100%" cellspacing="0" cellpadding="0" border="0">  
...  
</table>
```

Abstrakte Syntax für Markup-Symbole (*Tags*):

```
start-tag ::= stag-open identifier {attribute}* tag-close  
end-tag   ::= etag-open identifier tag-close  
attribute ::= identifier = "value"
```

Konkrete Syntax für Markup-Symbole (festgelegt in der SGML-Deklaration):

```
stag-open   ::= <  
etag-open  ::= </  
tag-close  ::= >  
identifier ::= {xchar}+  
value      ::= {char}+
```

Auszeichnungssprachen

SGML-Dokument

Allgemeine Form einer Element**instanz** [WT:III [HTML](#)] :

```
<elementname {attribute}*> ... </elementname>
```

Beispiel:

```
<table width="100%" cellspacing="0" cellpadding="0" border="0">
...
</table>
```

Abstrakte Syntax für Markup-Symbole (*Tags*):

```
start-tag ::= stag-open identifier {attribute}* tag-close
end-tag   ::= etag-open identifier tag-close
attribute ::= identifier = "value"
```

Konkrete Syntax für Markup-Symbole (festgelegt in der [SGML-Deklaration](#)):

```
stag-open   ::= <
etag-open   ::= </
tag-close   ::= >
identifier  ::= {xchar}+
value       ::= {char}+
```

Auszeichnungssprachen

SGML-Dokument

Allgemeine Form einer Element**instanz** [WT:III HTML] :

```
<elementname {attribute}*> ... </elementname>
```

Beispiel:

```
<table width="100%" cellspacing="0" cellpadding="0" border="0">
...
</table>
```

Abstrakte Syntax für Markup-Symbole (*Tags*):

```
start-tag ::= stag-open identifier {attribute}* tag-close
end-tag   ::= etag-open identifier tag-close
attribute ::= identifier = "value"
```

Konkrete Syntax für Markup-Symbole (festgelegt in der SGML-Deklaration):

```
stag-open   ::= <
etag-open   ::= </
tag-close   ::= >
identifier  ::= {xchar}+
value       ::= {char}+
```

Auszeichnungssprachen

SGML-Dokument

Allgemeine Form einer Element**instanz** [WT:III HTML] :

```
<elementname {attribute}*> ... </elementname>
```

Beispiel:

```
<table width="100%" cellspacing="0" cellpadding="0" border="0">  
...  
</table>
```

Abstrakte Syntax für Markup-Symbole (*Tags*):

```
start-tag ::= stag-open identifier {attribute}* tag-close  
end-tag   ::= etag-open identifier tag-close  
attribute ::= identifier = "value"
```

Konkrete Syntax für Markup-Symbole (festgelegt in der SGML-Deklaration):

```
stag-open  ::= <  
etag-open  ::= </  
tag-close  ::= >  
identifier ::= {xchar}+  
value      ::= {char}+
```

Auszeichnungssprachen

SGML-Dokument

Allgemeine Form einer Element**instanz** [WT:III [HTML](#)] :

```
<elementname {attribute}*> ... </elementname>
```

Beispiel:

```
<table width="100%" cellspacing="0" cellpadding="0" border="0">
...
</table>
```

Abstrakte Syntax für Markup-Symbole (*Tags*):

```
start-tag ::= stag-open identifier {attribute}* tag-close
end-tag  ::= etag-open identifier tag-close
attribute ::= identifier = "value"
```

Konkrete Syntax für Markup-Symbole (festgelegt in der [SGML-Deklaration](#)):

```
stag-open  ::= <
etag-open  ::= </
tag-close  ::= >
identifier ::= {xchar}+
value      ::= {char}+
```

Auszeichnungssprachen

SGML Document Type Definition [WT:III XML]

Die DTD definiert:

1. Art und Aufbau von Elementtypen für eine Klasse von Dokumenten
= die Inhaltsmodelle der Elementtypen
2. die in Elementeninstanzen verwendbaren Attribute und ihre Datentypen
3. verschiedene Arten von Textkonstanten, sogenannte *Entities*

Auszeichnungssprachen

SGML Document Type Definition (Fortsetzung)

Beispiel für die Deklaration eines Element**types** in einer DTD:

```
<!ELEMENT book                (titlepage, body)>
<!ELEMENT titlepage           (title, author)>
<!ELEMENT body                 (table-of-contents, chapter+)>
<!ELEMENT chapter             (chapterhead, section+)>
<!ELEMENT title                (#PCDATA)>
```

Auszeichnungssprachen

SGML Document Type Definition (Fortsetzung)

Beispiel für die Deklaration eines Elementtyps in einer DTD:

```
<!ELEMENT book (titlepage, body)>
<!ELEMENT titlepage (title, author)>
<!ELEMENT body (table-of-contents, chapter+)>
<!ELEMENT chapter (chapterhead, section+)>
<!ELEMENT title (#PCDATA)>
```

<!ELEMENT	Beginn der Deklaration des Elementtyps
chapter	Name des Elementtyps
(Beginn des Inhalts der Deklaration
chapterhead,	genau ein Kapitelkopf muss vorkommen
section+	mindestens ein Abschnitt muss vorkommen
)	Ende des Inhalts
>	Ende der Deklaration
#PCDATA	Datentyp "Parsed Character Data" [w3schools 1 2]

Auszeichnungssprachen

SGML Document Type Definition (Fortsetzung)

Beispiel für die Deklaration eines Elementtyps in einer DTD:

```
<!ELEMENT book (titlepage, body)>
<!ELEMENT titlepage (title, author)>
<!ELEMENT body (table-of-contents, chapter+)>
<!ELEMENT chapter (chapterhead, section+)>
<!ELEMENT title (#PCDATA)>
```

<!ELEMENT	Beginn der Deklaration des Elementtyps
chapter	Name des Elementtyps
(Beginn des Inhalts der Deklaration
chapterhead,	genau ein Kapitelkopf muss vorkommen
section+	mindestens ein Abschnitt muss vorkommen
)	Ende des Inhalts
>	Ende der Deklaration
#PCDATA	Datentyp "Parsed Character Data" [w3schools 1 2]

Beispiel für die Deklaration einer Textkonstante (*Entity*):

```
<!ENTITY euro "&#8364;"> [w3schools]
```

Bemerkungen:

- ❑ Die Elemente einer DTD können in einem SGML-Dokument instantiiert werden und dienen so im eigentlichen Inhalt als Markup.
- ❑ Entities werden durch den Aufruf `&Entityname;` referenziert.
- ❑ DTDs lassen sich auf zwei Arten einsetzen:
 1. Zur Analyse, um vorgegebene Dokumente zu validieren.
 2. Zur Synthese, um neue Dokumente zu generieren.

Auszeichnungssprachen

Zusammenhang SGML, XML, HTML, XHTML

XML, *Extensible Markup Language*, ist eine Teilmenge von SGML, die speziell auf die Bedürfnisse des WWW zugeschnitten und stark vereinfacht ist:

- XML hat eine feste, nicht veränderbare SGML-Deklaration.

Auszeichnungssprachen

Zusammenhang SGML, XML, HTML, XHTML

XML, *Extensible Markup Language*, ist eine Teilmenge von SGML, die speziell auf die Bedürfnisse des WWW zugeschnitten und stark vereinfacht ist:

- XML hat eine feste, nicht veränderbare SGML-Deklaration.

HTML, *Hypertext Markup Language*, ist eine Teilmenge von SGML und ist, verglichen mit XML, noch weiter eingeschränkt:

- HTML hat eine feste, nicht veränderbare SGML-Deklaration.
- HTML verwendet eine feste Dokumentstruktur (und hat folglich nur *eine* DTD).
→ Kein Austausch von SGML-Deklaration und DTD erforderlich.

Auszeichnungssprachen

Zusammenhang SGML, XML, HTML, XHTML

XML, *Extensible Markup Language*, ist eine Teilmenge von SGML, die speziell auf die Bedürfnisse des WWW zugeschnitten und stark vereinfacht ist:

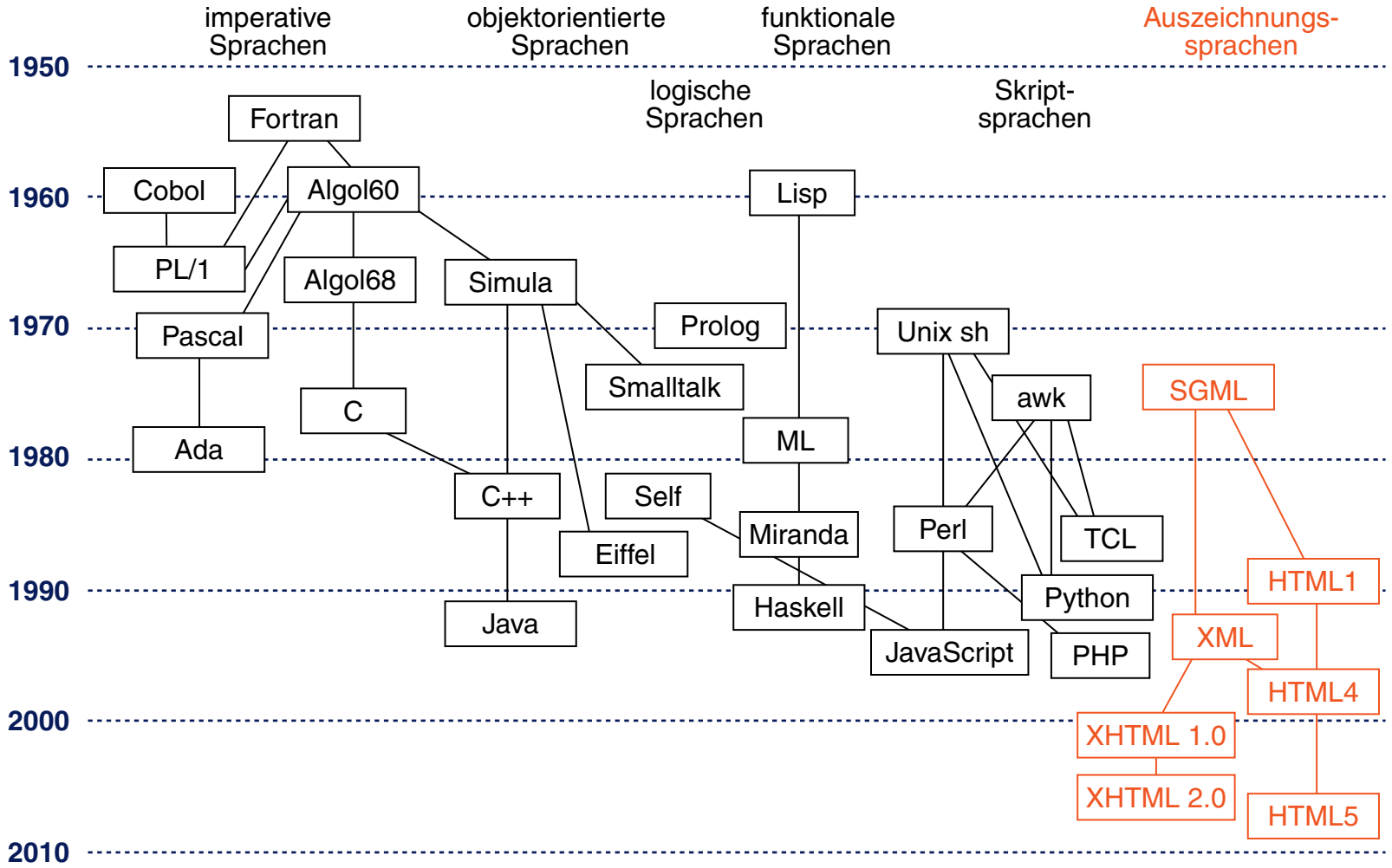
- XML hat eine feste, nicht veränderbare SGML-Deklaration.

HTML, *Hypertext Markup Language*, ist eine Teilmenge von SGML und ist, verglichen mit XML, noch weiter eingeschränkt:

- HTML hat eine feste, nicht veränderbare SGML-Deklaration.
- HTML verwendet eine feste Dokumentstruktur (und hat folglich nur *eine* DTD).
→ Kein Austausch von SGML-Deklaration und DTD erforderlich.

XHTML, *Extensible HyperText Markup Language*, ist die Definition von (Teilen von) HTML auf Basis von XML.

Auszeichnungssprachen



[Kastens 2005, www.levenez.com/lang]

III. Dokumentensprachen

- ❑ Auszeichnungssprachen
- ❑ HTML
- ❑ Cascading Stylesheets CSS
- ❑ XML-Grundlagen
- ❑ XML-Schema
- ❑ Die XSL-Familie
- ❑ APIs für XML-Dokumente

HTML

Einordnung

SGML hat alle notwendigen Konzepte.

Warum überhaupt HTML?

1. ein guter Kompromiss zwischen Einfachheit und Ausdrucksstärke
2. Verfügbarkeit und Plattformunabhängigkeit

HTML ermöglicht eine strikte Trennung
zwischen Dokumenteninhalt und Dokumentendarstellung,
erzwingt sie aber nicht.

Bemerkungen:

- ❑ HTML kompakt:
 1. Historie
 2. Prinzip der Dokumentenverarbeitung
 3. Aufbau eines HTML-Dokuments
 4. Inhaltsmodelle [W3C [REC 3.2.4](#)]
 5. Elementtypen für Inhaltsmodelle [W3C [REC 4](#)]
 6. Universalattribute [W3C [REC 3.2.5](#)]

HTML [W3C [status](#)]

Historie

- 1991 Vorstellung von ersten Versionen für URL, HTTP und HTML.
- 1994 HTML 2.0. Basiert auf standardkonformer SGML-DTD.
- 1998 HTML 4.0. Führt Cascading Stylesheets CSS ein.
- 1999 HTML 4.01. Recommendation. [W3C [REC](#)]

HTML [W3C [status](#)]

Historie

- 1991 Vorstellung von ersten Versionen für URL, HTTP und HTML.
- 1994 HTML 2.0. Basiert auf standardkonformer SGML-DTD.
- 1998 HTML 4.0. Führt Cascading Stylesheets CSS ein.
- 1999 HTML 4.01. Recommendation. [W3C [REC](#)]
- 2000 XHTML 1.0. Reformulierung von HTML4 in XML. [W3C [REC](#), [differences](#)] [\[Wikipedia\]](#)
- 2010 XHTML 2.0. Working Group Note, “back to the roots”. [W3C [NOTE](#), [1.1.3](#)]
- 2015 XHTML5. Working Group Note, Polyglot Markup. [W3C [NOTE](#)]

HTML [W3C [status](#)]

Historie

- 1991 Vorstellung von ersten Versionen für URL, HTTP und HTML.
- 1994 HTML 2.0. Basiert auf standardkonformer SGML-DTD.
- 1998 HTML 4.0. Führt Cascading Stylesheets CSS ein.
- 1999 HTML 4.01. Recommendation. [W3C [REC](#)]


- 2000 XHTML 1.0. Reformulierung von HTML4 in XML. [W3C [REC](#), [differences](#)] [\[Wikipedia\]](#)
- 2010 XHTML 2.0. Working Group Note, “back to the roots”. [W3C [NOTE](#), [1.1.3](#)]
- 2015 ~~XHTML5/Working Group Note/ Polyglot Markup~~. [W3C [NOTE](#)]

- 2008 HTML5. Recommendation. Loslösung von SGML, neue Struktur- und
2014 Multimedia-Elemente. [W3C [REC](#), [differences](#)] [\[Wikipedia \[article\]\(#\), \[figure\]\(#\)\]](#)
- 2016 HTML 5.1. Working Draft. [W3C [WD](#)]
- 2016 HTML. Living Standard. [whatwg [standard](#), [developers](#)]

Bemerkungen (HTML4):

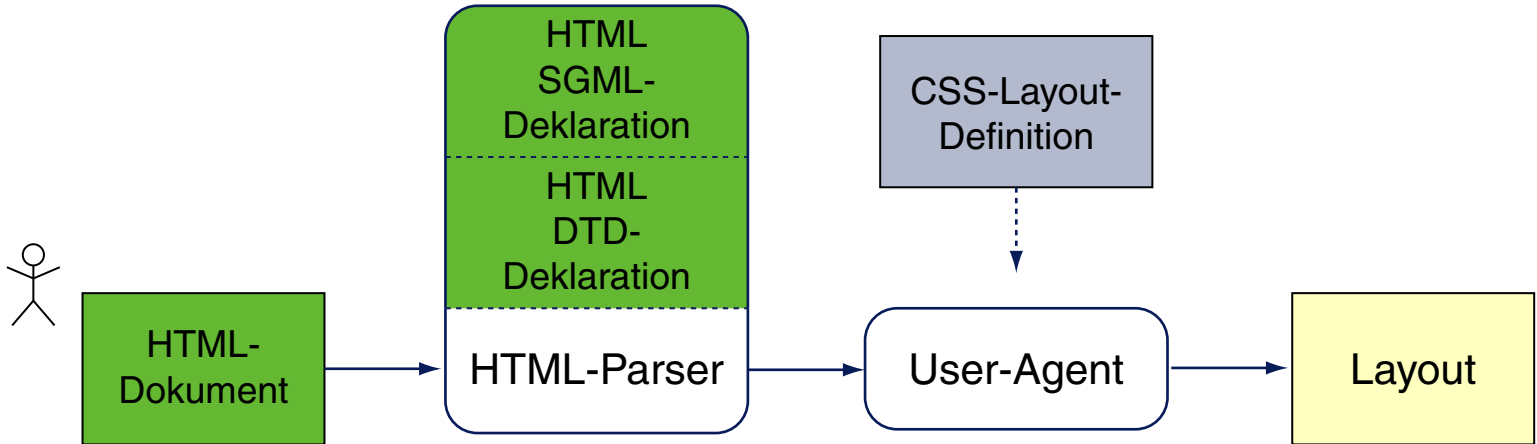
- ❑ Beispiele für die fehlende Trennung zwischen Dokumenteninhalte und Dokumentendarstellung sind Formatierungsangaben wie ``, `<center>`, etc.
- ❑ Mit der Einführung von Cascading Stylesheets in HTML 4.0 existiert ein Mechanismus, um Formatierungsangaben aus dem Dokumenteninhalte auszugliedern.
- ❑ XHTML 1.0 bringt keine neue Funktionalität gegenüber HTML 4.0, enthält aber die kleineren syntaktischen Anpassungen für den XML-Standard.
- ❑ Bei der Weiterentwicklung von XHTML 2.0 konnte keine Einigung zwischen W3C und der Industrie ([WHATWG-Konsortium](#)) erzielt werden. Mittlerweile arbeiten W3C und WHATWG gemeinsam an HTML. [W3C [REC 1.4](#)]
- ❑ Der Standardisierungsprozess der W3C ist formalisiert und spiegelt sich in den verschiedenen Leveln der veröffentlichten Reports wider. [W3C [level](#)]

Bemerkungen (HTML5):

- ❑ HTML5 führt neue Strukturelemente wie `<header>`, `<footer>` oder `<nav>` ein, um die Semantik eines Elements im Dokument explizit zu machen und die Lesbarkeit (auch für Maschinen) zu erhöhen.
- ❑ HTML5 zielt in besonderem Maße darauf ab, sogenannten *Rich Content* darstellen zu können. Beispielsweise ermöglichen `<canvas>`, `<video>` und `<audio>` die native Medieneinbindung und sollen Plugin-Technologien wie Flash überflüssig machen.
- ❑ HTML5 reagiert auf die große Menge nicht valider Dokumente im Web (Stichwort: *tag soup*), die bislang jeder Browser auf eigene Weise behandelt, mit einer standardisierten Fehlerbehandlung (Stichwort: *quirks mode*). [[W3C wiki](#)] [[Wikipedia](#)]
- ❑ HTML5-Logo: . Schreibweise: HTML5 oder HTML 5? [[whatwg](#)]

HTML

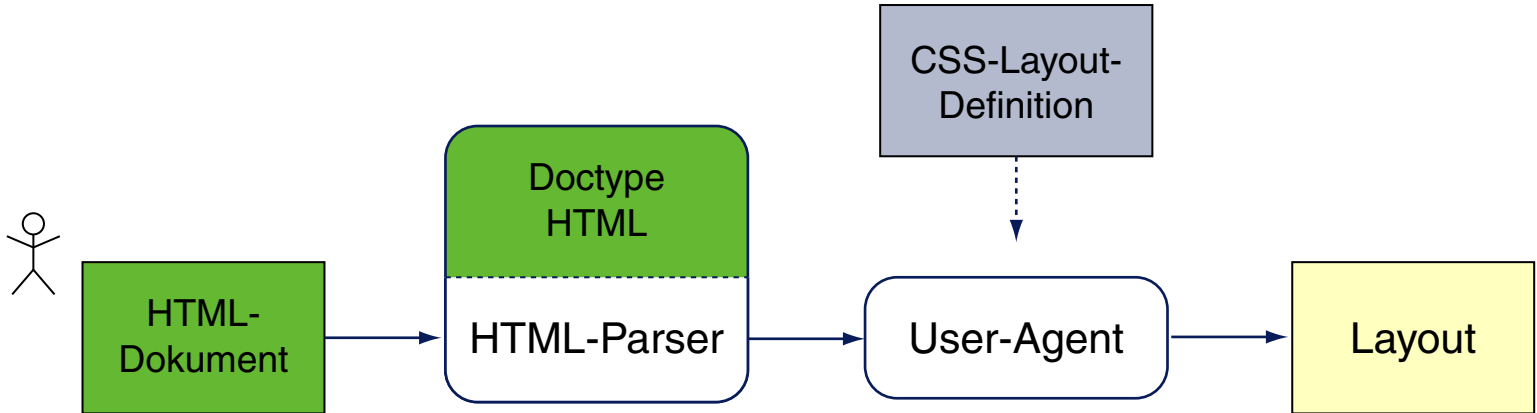
HTML Dokumentenverarbeitung (HTML4)



Vergleiche hierzu die SGML Dokumentenverarbeitung.

HTML

HTML Dokumentenverarbeitung (HTML5)

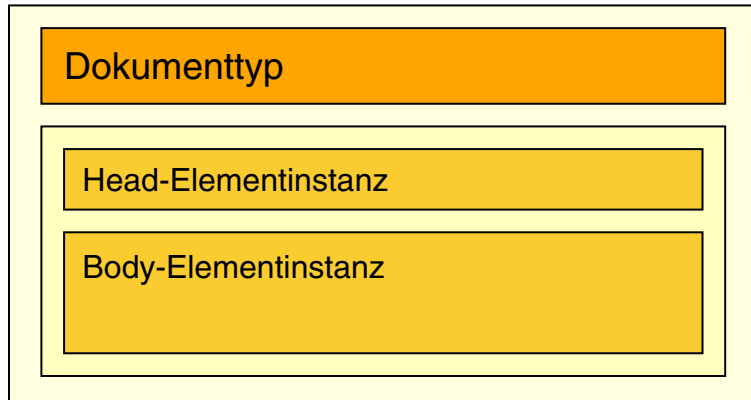


Vergleiche hierzu die SGML Dokumentenverarbeitung.

HTML

HTML-Dokument

HTML-
Dokument



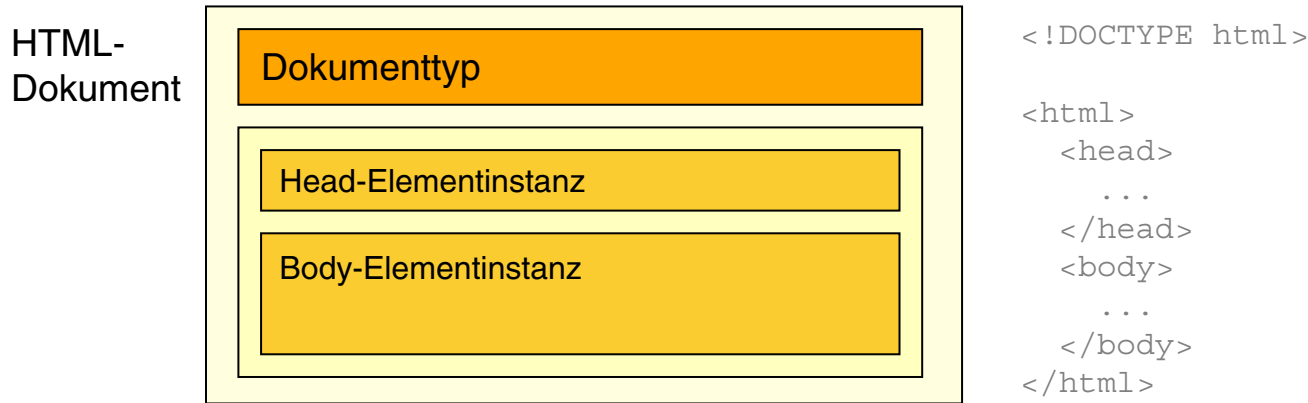
```
<!DOCTYPE html>
```

```
<html>  
  <head>  
    ...  
  </head>  
  <body>  
    ...  
  </body>  
</html>
```

- ❑ Das `<html>`-Element repräsentiert die Dokument-Wurzel. [W3C [REC 4.1](#)]
- ❑ Das `<head>`-Element repräsentiert die Meta-Daten. [W3C [REC 4.2](#)]
- ❑ Das `<body>`-Element repräsentiert den Dokumentinhalt. [W3C [REC 4.3](#)]
- ❑ Vergleiche hierzu die [XML-Dokumentstruktur](#).

HTML

HTML-Dokument



- ❑ Das `<html>`-Element repräsentiert die Dokument-Wurzel. [W3C [REC 4.1](#)]
- ❑ Das `<head>`-Element repräsentiert die Meta-Daten. [W3C [REC 4.2](#)]
- ❑ Das `<body>`-Element repräsentiert den Dokumentinhalt. [W3C [REC 4.3](#)]
- ❑ Vergleiche hierzu die [XML-Dokumentstruktur](#).

Allgemeine Form einer HTML-Element**instanz** [WT:III [SGML](#)] :

```
<elementname {attribute}*> ... </elementname>
```

HTML

DTD-Deklaration (HTML4)

HTML verwendet eine feste Dokumentstruktur, die bei HTML4 als DTD (Document Type Definition) spezifiziert ist. Unterscheidung von drei DTD-Varianten [\[W3C\]](#) [\[SELFHTML\]](#) :

1. Strict

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"  
    "https://www.w3.org/TR/html4/strict.dtd">
```

Trennung zwischen Inhalt und Darstellung: keine Formatierungsangaben erlaubt; strenge Verschachtelungsregeln; kein Inhalt ohne Block-Auszeichnung.

2. Transitional

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"  
    "https://www.w3.org/TR/html4/loose.dtd">
```

Hat nicht die Beschränkungen der Strict-DTD.

3. Frameset

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Frameset//EN"  
    "https://www.w3.org/TR/html4/frameset.dtd">
```

Für HTML-Dokumente mit Framesets.

HTML

DTD-Deklaration (HTML4)

HTML verwendet eine feste Dokumentstruktur, die bei HTML4 als DTD (Document Type Definition) spezifiziert ist. Unterscheidung von drei DTD-Varianten [\[W3C\]](#) [\[SELFHTML\]](#) :

1. Strict

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"  
  "https://www.w3.org/TR/html4/strict.dtd">
```

Trennung zwischen Inhalt und Darstellung: keine Formatierungsangaben erlaubt; strenge Verschachtelungsregeln; kein Inhalt ohne Block-Auszeichnung.

2. Transitional

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"  
  "https://www.w3.org/TR/html4/loose.dtd">
```

Hat nicht die Beschränkungen der Strict-DTD.

3. Frameset

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Frameset//EN"  
  "https://www.w3.org/TR/html4/frameset.dtd">
```

Für HTML-Dokumente mit Framesets.

Bemerkungen:

- ❑ Ein HTML4-Dokument ohne DTD-Deklaration wird nach den Regeln der Transitional-DTD für HTML 4.01 verarbeitet.
- ❑ Ein HTML4-Dokument darf nur *eine* DTD besitzen. Bei der Verwendung von Frames ermöglicht die Frameset-DTD für jedes Frame die Einbindung einer DTD.
- ❑ HTML5 ist weitgehend kompatibel zu HTML 4.01 und XHTML 1.0, basiert aber nicht mehr auf SGML. Folglich ist die Dokumentstruktur nicht mehr in Form einer DTD spezifiziert.
[\[Wikipedia\]](#) [\[CoreLangs\]](#) [\[W3C REC 8.1\]](#)

HTML

Inhaltsmodelle (HTML4)

Elementinstanzen innerhalb einer `<body>`-Elementinstanz gehören zu genau einer der folgenden zwei Kategorien [\[MDN 1 2\]](#) :

1. Block-Elemente

2. Inline-Elemente

HTML

Inhaltsmodelle (HTML4)

Elementinstanzen innerhalb einer `<body>`-Elementinstanz gehören zu genau einer der folgenden zwei Kategorien [\[MDN 1 2\]](#) :

1. Block-Elemente

Instanzen von Block-Elementen erzeugen einen eigenen Absatz im Textfluss; sie können normalen Text und Instanzen von Inline-Elementen enthalten; einige dürfen auch Instanzen anderer Block-Elemente enthalten.

Beispiele für Block-Elemente:

```
<center>, <div>, <form>, <h1>, <noframes>, <p>, <table>, <ul>
```

2. Inline-Elemente

Instanzen von Inline-Elementen werden in derselben Zeile wie der vorhergehende Text gesetzt; sie können normalen Text und Instanzen weiterer Inline-Elemente enthalten.

Beispiele für Inline-Elemente:

```
<a>, <br>, <cite>, <em>, <font>, <img>, <small>, <span>
```

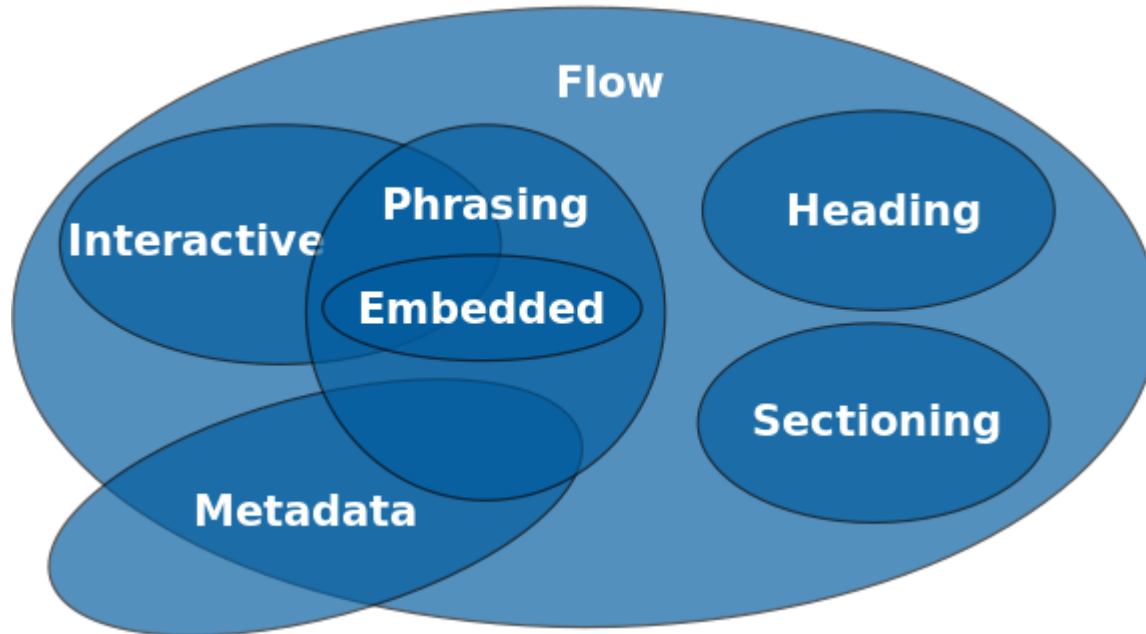
Bemerkungen:

- ❑ Die Verarbeitung von Block-Elementen aus Sicht des Layout-Programms (beispielsweise mit einem Web-Browser) ist mit dem Verhalten von \LaTeX im `\vmode` vergleichbar. Die Verarbeitung von Inline-Elementen ist mit dem Verhalten von \LaTeX im `\hmode` vergleichbar.

HTML

Inhaltsmodelle (HTML5) [W3C [REC 3.2.4](#)] [MDN] [whatwg]

Elementinstanzen innerhalb einer `<body>`-Elementinstanz fallen in mindestens eine der folgenden sieben Inhaltskategorien [W3C [REC 3.2.4.1](#)] :



Bemerkungen:

- ❑ Bei HTML5 ist die syntaktische Aufteilung in Block- und Inline-Elemente durch eine an semantischen Überlegungen orientierte Aufteilung abgelöst bzw. ergänzt worden. Aus Sicht des Layout-Programms (beispielsweise des Web-Browsers) gilt für die beiden Philosophien in etwa die folgende Entsprechung [\[MDN\]](#) :
 - Block-Elemente (HTML4) ~ Flow Content
 - Inline-Elemente (HTML4) ~ Phrasing Content

- ❑ HTML5 verzichtet auf eine Reihe von (Block-)Elementen, die bei HTML4 vordringlich zur Layout-Gestaltung dienen [\[w3schools\]](#) : `<center>`, `<frame>`, `<frameset>`, `<noframes>`

HTML

Metadaten (*Document Metadata*) [W3C [REC 4.2](#)]

□ Titel [[SELFHTML](#)]

```
<head>  
  <title>Lemmy Caution's Strange Adventures</title>  
</head>
```

Bei HTML4 ist der Titel obligatorisch, bei HTML5 kann er fehlen, falls er ableitbar ist. Der Titel erscheint nicht im dargestellten HTML-Dokument, wird aber als Fenstertitel, Lesezeichen, von Robots etc. ausgewertet.

□ Adressbasis [[SELFHTML](#)]

```
<head>  
  <title>...</title>  
  
  <base href="https://www.my-webserver.de/absolute/path">  
  
</head>
```

Definiert einen absoluten Bezugspfad und ermöglicht so die Verwendung von relativen Pfaden im Dokument.

HTML

Metadaten (*Document Metadata*) [W3C [REC 4.2](#)]

□ Links [[SELFHTML](#)]

```
<head>
  <title>...</title>

  <link rel="stylesheet" href="../share/bib.css" type="text/css">

</head>
```

Ermöglicht die Referenzierung (keine Hyperlinks) von Dokumenten; wird meist zur Angabe von externen Stylesheets verwendet.

□ Meta-Tags [[SELFHTML](#)]

```
<head>
  <title>...</title>

  <meta name="author" content="Judea Pearl">
  <meta name="keywords" content="Heuristics, Search, Bayes">
  <meta http-equiv="Content-Script-Type" content="text/javascript">

</head>
```

Meta-Tags haben meist zwei Attribute „**Eigenschaft** = **Wert**“ (name bzw. http-equiv = content); sie dienen zur Information von Web-Browsern, Robots und Web-Servern.

Bemerkungen:

- ❑ Zur Standardisierung von Meta-Tags hat das W3C die Sprache RDF (*Resource Description Framework*) entworfen.
- ❑ Meta-Tags, die mit `http-equiv` definiert sind, werden vom Client-Programm wie ein HTTP-Entity-Header einer HTTP-Response-Message interpretiert. Ein gleichnamiger HTTP-Header in der Response-Message hat Vorrang gegenüber einer Meta-Angabe im HTML-Dokument.

HTML

Dokumentaufteilung (*Sections*) [W3C [REC 4.3](#)]

□ Strukturelemente (HTML5) [W3C [wiki](#)]

<code><article></code>	eigenständiger Inhalt, ggf. mit eigenem <code><header></code> und <code><footer></code>
<code><section></code>	(1) Gruppierung verschiedener <code><article></code> in Themen, (2) Einteilung <i>eines</i> <code><article></code> in Abschnitte. Typisch mit Überschrift für Inhalte
<code><nav></code>	Navigationsmenü oder andere Navigationsmöglichkeiten
<code><aside></code>	Gruppierung von verwandter Information mit Bezug zum Hauptinhalt
<code><header></code>	Kopfinformationen einer Website oder eines Artikels
<code><footer></code>	Fußzeile einer Website oder eines Artikels

HTML

Dokumentaufteilung (*Sections*) [W3C [REC 4.3](#)]

□ Strukturelemente (HTML5) [W3C [wiki](#)]

<code><article></code>	eigenständiger Inhalt, ggf. mit eigenem <code><header></code> und <code><footer></code>
<code><section></code>	(1) Gruppierung verschiedener <code><article></code> in Themen, (2) Einteilung <i>eines</i> <code><article></code> in Abschnitte. Typisch mit Überschrift für Inhalte
<code><nav></code>	Navigationsmenü oder andere Navigationsmöglichkeiten
<code><aside></code>	Gruppierung von verwandter Information mit Bezug zum Hauptinhalt
<code><header></code>	Kopfinformationen einer Website oder eines Artikels
<code><footer></code>	Fußzeile einer Website oder eines Artikels

□ Überschriftselemente [W3C [REC 4.3.6](#)] [[SELFHTML](#)]

```
<h1>&Uuml;berschrift 1. Ordnung</h1>  
...  
<h6>&Uuml;berschrift 6. Ordnung</h6>
```

HTML

Inhaltsgruppierung (*Grouping Content*) [W3C [REC 4.4](#)]

□ Listen [[SELFHTML](#)]

<code></code>	geordnete Liste
<code></code>	ungeordnete Liste
<code></code>	Listeneintrag
<code><dl></code>	Definitionsliste
<code><dt></code>	Definitionsüberschrift
<code><dd></code>	Definitionseintrag

HTML

Textauszeichnungen (*Text-Level Semantics*) [W3C [REC 4.5](#)]

Unterscheidung von Textauszeichnungen hinsichtlich ihrer Konkretheit [[MDN](#)] :

1. Physische Auszeichnungen (HTML4)

<code><i></code>	zeichnet einen Text als kursiv aus
<code></code>	zeichnet einen Text als fett aus
<code><u></code>	zeichnet einen Text als unterstrichen aus
<code><strike></code>	zeichnet einen Text als durchgestrichen aus
<code><tt></code>	zeichnet einen Text in Schreibmaschinenschrift aus

2. Logische Auszeichnungen (HTML4 und HTML5)

<code></code>	zeichnet einen Text als betonten, wichtigen Text aus
<code></code>	zeichnet einen Text als stark betont aus (Steigerung von <code></code>)
<code><cite></code>	zeichnet einen Text als Zitat aus
<code><code></code>	zeichnet einen Text als Quelltext aus
<code><samp></code>	zeichnet einen Text als Beispiel aus

Bemerkungen:

- ❑ *“Logical states separate presentation from the content, and by doing so allow for it to be expressed in many different ways.”* [\[MDN\]](#)
- ❑ HTML5 verzichtet auf eine Reihe von Elementen, die bei HTML4 vordringlich zur physischen Auszeichnungen dienen [\[w3schools\]](#): `<basefont>`, `<big>`, `<dir>`, `<strike>`, `<tt>`
Die weiteren HTML4-Elemente zur physischen Auszeichnung haben bei HTML5 eine explizite Semantik erhalten. Beispiele: `<i>`, ``, `<u>` [\[W3C REC 4.5\]](#)
- ❑ Bei HTML5 dienen die Elemente `<ins>`, `` zur Auszeichnung von sogenannten *Edits*. [\[W3C REC 4.6\]](#) [\[w3schools\]](#)
- ❑ Beispiele für physische Auszeichnungen in \LaTeX sind die Schriftschnitte und -gewichte:
`\itshape`, `\bfseries`, `\fontfamily{phv}` `\fontsize{8}{0}` `\selectfont`
Beispiele für logische Auszeichnungen in \LaTeX :
`\em`, `\begin{quote} ... \end{quote}`

HTML

Medieneinbindung (*Embedded Content*) [W3C [REC 4.7](#)]

Wichtige Elemente:

- ❑ ``
[\[SELFHTML\]](#)
- ❑ `<iframe src="https://www.w3c.org"></iframe>`
[\[SELFHTML\]](#)
- ❑ `<object data="animation.swf" width="400" height="200"></object>`
[\[SELFHTML\]](#)
- ❑ `<embed src="animation.swf">`

HTML

Hyperlinks (*Links*) [W3C [REC 4.8](#)] [[SELFHTML](#)]

Zur Definition von Hyperlinks dient das `<a>`-Element (*Anchor*). Als Inline-Element (HTML4) kann es keine Instanzen von Block-Elementen auszeichnen; der erlaubte Kontext (HTML5) ist Phrasing Content [W3C [REC 4.5.1](#)].

- Hyperlink

- Hyperlink-Ziel

HTML

Hyperlinks (*Links*) [W3C REC 4.8] [SELFHTML]

Zur Definition von Hyperlinks dient das `<a>`-Element (*Anchor*). Als Inline-Element (HTML4) kann es keine Instanzen von Block-Elementen auszeichnen; der erlaubte Kontext (HTML5) ist Phrasing Content [W3C REC 4.5.1].

□ Hyperlink

``

Ziel ist durch *URI* definiert

Bsp.: ``

Ziel ist benannter Anker im aktuellen Dokument

Bsp.: ``

Ziel ist Anker in relativ referenziertem Dokument

Optionale Attribute des Anchor-Elements:

<code>title</code>	definiert den Mouse-Over-Text
--------------------	-------------------------------

<code>type</code>	MIME-Type des Zieldokuments
-------------------	-----------------------------

<code>download</code>	spezifiziert, dass lokal gespeichert werden soll
-----------------------	--

□ Hyperlink-Ziel

HTML

Hyperlinks (*Links*) [W3C REC 4.8] [SELFHTML]

Zur Definition von Hyperlinks dient das `<a>`-Element (*Anchor*). Als Inline-Element (HTML4) kann es keine Instanzen von Block-Elementen auszeichnen; der erlaubte Kontext (HTML5) ist Phrasing Content [W3C REC 4.5.1].

□ Hyperlink

``

Ziel ist durch *URI* definiert

Bsp.: ``

Ziel ist benannter Anker im aktuellen Dokument

Bsp.: ``

Ziel ist Anker in relativ referenziertem Dokument

Optionale Attribute des Anchor-Elements:

<code>title</code>	definiert den Mouse-Over-Text
--------------------	-------------------------------

<code>type</code>	MIME-Type des Zieldokuments
-------------------	-----------------------------

<code>download</code>	spezifiziert, dass lokal gespeichert werden soll
-----------------------	--

□ Hyperlink-Ziel

``

Zieldefinition im selben Dokument

Bemerkungen:

- ❑ Die Syntax von Hyperlinks ist unabhängig von dem angegebenen Ziel.
- ❑ URIs, die mit einem Dokumentanker # *Identifizier* abschließen, werden auch als Fragment-Identifizier bezeichnet, weil sie ein Dokument nicht als Ganzes, sondern abschnittsgenau adressieren.
- ❑ Bei HTML4 kann das a-Element in Kombination mit dem name-Attribut als Anker verwendet werden. Bei HTML5 ist im a-Element das name-Attribut nicht mehr erlaubt; das id-Attribut kann verwendet werden. [w3schools [a](#), [name](#)]

HTML

Tabellen (*Tabular Data*) [W3C REC 4.9] [SELFHTML]

□ Elemente

<code><table></code>	Tabelle
<code><caption></code>	Tabellenüberschrift
<code><colgroup></code>	Spaltengruppe
<code><col></code>	Tabellenspalte
<code><tbody></code>	Tabellenkörper
<code><thead></code>	Tabellenkopf
<code><tfoot></code>	Tabellenfuß
<code><tr></code>	Tabellenzeile
<code><td></code>	einzelne Zelle
<code><th></code>	Zelle mit Überschrift

Spalte 1	Spalte 2	Spalte 3
Zelle 1.1	Zelle 1.2	Zelle 1.3
Zelle 2.1	Zelle 2.2	Zelle 2.3

[\[html-table.html\]](#)

Bemerkungen:

- ❑ Die Attribute zur Tabellengestaltung wie `align`, `bgcolor`, etc., werden bei HTML5 nicht mehr unterstützt. [\[w3schools\]](#)

HTML

Formulare (*Forms*) [W3C [REC 4.10](#)] [[SELFHTML](#)]

Alles, was zwischen dem einleitenden `<form>`-Tag und dem abschließenden `</form>`-Tag steht, gehört zum Formular.

- Attribute des `<form>`-Elements [W3C [REC 4.10.3](#)]

- Kindelemente des `<form>`-Elements [[SELFHTML](#)]

HTML

Formulare (*Forms*) [W3C [REC 4.10](#)] [[SELFHTML](#)]

Alles, was zwischen dem einleitenden `<form>`-Tag und dem abschließenden `</form>`-Tag steht, gehört zum Formular.

□ Attribute des `<form>`-Elements [W3C [REC 4.10.3](#)]

<code>action</code>	definiert URI vom Server-Anwendungsprogramm oder <code>mailto:</code>
<code>method</code>	spezifiziert die <code>get</code> oder <code>post</code> -Methode des HTTP-Protokolls
<code>enctype</code>	Angabe eines MIME-Typs

□ Kindelemente des `<form>`-Elements [[SELFHTML](#)]

<code><input></code>	Definition von Eingabefeld
<code><label></code>	Beschreibungstext zu Eingabefeld
<code><fieldset></code>	Gruppierung von Formularelementen

HTML

Formulare (*Forms*) [W3C [REC 4.10](#)] [[SELFHTML](#)]

Alles, was zwischen dem einleitenden `<form>`-Tag und dem abschließenden `</form>`-Tag steht, gehört zum Formular.

□ Attribute des `<form>`-Elements [W3C [REC 4.10.3](#)]

<code>action</code>	definiert URI vom Server-Anwendungsprogramm oder <code>mailto:</code>
<code>method</code>	spezifiziert die <code>get</code> oder <code>post</code> -Methode des HTTP-Protokolls
<code>enctype</code>	Angabe eines MIME-Typs

□ Kindelemente des `<form>`-Elements [[SELFHTML](#)]

<code><input></code>	Definition von Eingabefeld
<code><label></code>	Beschreibungstext zu Eingabefeld
<code><fieldset></code>	Gruppierung von Formularelementen

Attribute des `<input>`-Elements

<code>type</code>	Typ des Eingabefelds: <code>text</code> , <code>radio</code> , <code>submit</code> [SELFHTML]
<code>size</code>	definiert die Characteranzahl des Eingabefelds
<code>value</code>	definiert einen Default-Wert
<code>name</code>	definiert Variablennamen im <code><form></code> -Element

Bemerkungen:

- ❑ HTML5 erweitert die Attribute des `<input>`-Elements. So ermöglicht `type` zusätzliche Datentypen mit den passenden Eingaben, `placeholder` eine adäquatere Gestaltung und `autofocus`, `pattern`, `required` eine leistungsfähigere Validierung. [W3C [wiki](#)]

HTML

Formulare: Beispiel [\[html-form.html\]](#)

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>html-form</title>
  </head>
  <body>
    <form name="Webis" action="mailto:benno.stein@uni-weimar.de"
      method="post" enctype="text/plain">
      <fieldset>
        <legend>Formular 1</legend>
        <label><input type="radio" name="x" value="1">Radio-Text 1</label>
        <label><input type="radio" name="x" value="2" checked="checked">Radio-Text 2</label>
        <input type="submit" name="z" value="Email schreiben">
      </fieldset>
    </form>

  </body>
</html>
```

HTML

Formulare: Beispiel [\[html-form.html\]](#)

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>html-form</title>
  </head>
  <body>
    <form name="Webis" action="mailto:benno.stein@uni-weimar.de"
          method="post" enctype="text/plain">
      <fieldset>
        <legend>Formular 1</legend>
        <label><input type="radio" name="x" value="1">Radio-Text 1</label>
        <label><input type="radio" name="x" value="2" checked="checked">Radio-Text 2</label>
        <input type="submit" name="z" value="Email schreiben">
      </fieldset>
    </form>

    </body>
</html>
```

Test - Mozilla Firefox

Formular 1

Radio-Text 1 Radio-Text 2

Email schreiben

Formular 2

Vorname:

Nachname:

Daten schicken

HTML

Formulare: Beispiel [\[html-form.html\]](#)

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>html-form</title>
  </head>
  <body>
    <form name="Webis" action="mailto:benno.stein@uni-
      method="post" enctype="text/plain">
      <fieldset>
        <legend>Formular 1</legend>
        <label><input type="radio" name="x" value="1">
        <label><input type="radio" name="x" value="2">
        <input type="submit" name="z" value="Email s
      </fieldset>
    </form>

    <form action="http://webis16.medien.uni-weimar.de/cgi-bin/cgi-sample1.cgi">
      <fieldset>
        <legend>Formular 2</legend>
        <label for="field1">Vorname:</label>
        <input id="field1" name="vorname" type="text" value="Stefan">
        <label for="field2">Nachname:</label>
        <input id="field2" name="nachname" type="text" placeholder="Nachnamen eingeben">
        <input type="submit" name="z" value="Daten schicken">
      </fieldset>
    </form>
  </body>
</html>
```

Test - Mozilla Firefox

Formular 1

Radio-Text 1 Radio-Text 2

Email schreiben

Formular 2

Vorname:

Nachname:

Daten schicken

HTML

Formulare: Beispiel (Fortsetzung)

Erzeugung der HTTP-Response-Message via Shell-Script cgi-sample1.cgi :

```
#!/bin/bash

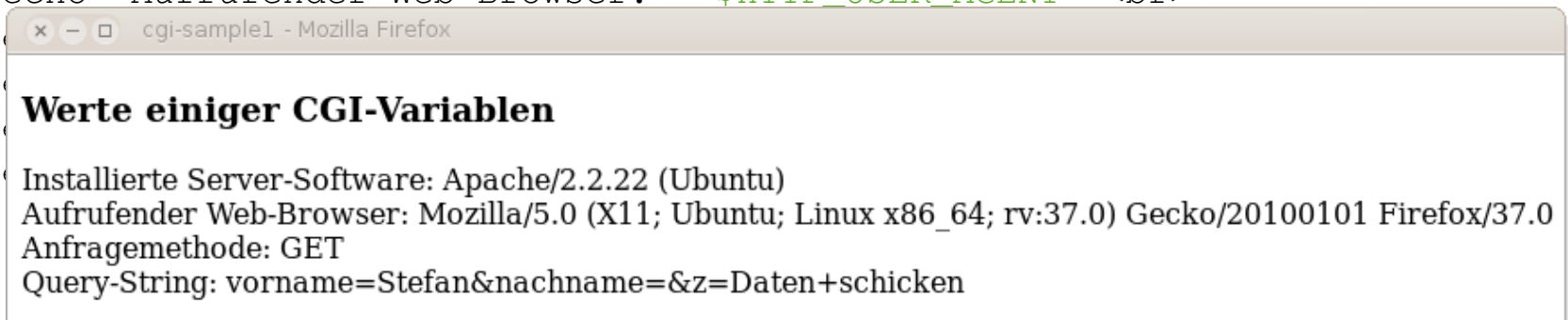
echo "content-type: text/html"
echo "" #Leerzeile gemäß HTTP-Protokoll.
echo "<!DOCTYPE html>"
echo "<html>"
echo "<head>"
echo "<meta http-equiv=\" \"content-type\" \" \"content=\" \"text/html; ...
echo "<title>cgi-sample1</title>"
echo "</head>"
echo "<body>"
echo "<h3>Werte einiger CGI-Variablen</h3>"
echo "Installierte Server-Software: " $SERVER_SOFTWARE "<br>"
echo "Aufrufender Web-Browser: " $HTTP_USER_AGENT "<br>"
echo "Anfragemethode: " $REQUEST_METHOD "<br>"
echo "Query-String: " $QUERY_STRING "<br>"
echo "</body>"
echo "</html>"
```

HTML

Formulare: Beispiel (Fortsetzung)

Erzeugung der HTTP-Response-Message via Shell-Script cgi-sample1.cgi :

```
#!/bin/bash
echo "content-type: text/html"
echo "" # Leerzeile gemäß HTTP-Protokoll.
echo "<!DOCTYPE html>"
echo "<html>"
echo "<head>"
echo "<meta http-equiv=\" \"content-type\" \" \"content=\" \"text/html; ...
echo "<title>cgi-sample1</title>"
echo "</head>"
echo "<body>"
echo "<h3>Werte einiger CGI-Variablen</h3>"
echo "Installierte Server-Software: " $SERVER_SOFTWARE "<br>"
echo "Aufrufender Web-Browser: " $HTTP_USER_AGENT "<br>"
```



```
x - □ cgi-sample1 - Mozilla Firefox

Werte einiger CGI-Variablen

Installierte Server-Software: Apache/2.2.22 (Ubuntu)
Aufrufender Web-Browser: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:37.0) Gecko/20100101 Firefox/37.0
Anfragemethode: GET
Query-String: vorname=Stefan&nachname=&z=Daten+schicken
```

HTML

Interaktivität (*Scripting*) [W3C [REC 4.11](#)]

Wichtige Elemente:

- ❑

```
<script>
  document.write("Hello world.")
</script>
```
- ❑

```
<noscript>
  Browser does not support JavaScript.
</noscript>
```
- ❑

```
<canvas id="Demo"></canvas>
<script>
  var canvas = document.getElementById("Demo");
  var canvasCtx = canvas.getContext("2d");
  ...
</script>
```

HTML

Universalattribute (*Global Attributes*) [[W3C REC 3.2.5](#)] [[w3schools](#)] [[SELFHTML](#)]

Universalattribute sind in allen HTML-Elementen verwendbar. Einteilung und Beispiele:

1. Allgemeine

2. Zur Internationalisierung

HTML

Universalattribute (*Global Attributes*) [[W3C REC 3.2.5](#)] [[w3schools](#)] [[SELFHTML](#)]

Universalattribute sind in allen HTML-Elementen verwendbar. Einteilung und Beispiele:

1. Allgemeine

<code>id</code>	ordnet der Elementinstanz einen individuellen Namen zu
<code>title</code>	definiert den Mouse-Over-Text
<code>class</code>	ordnet der Elementinstanz eine Stylesheet-Klasse zu
<code>style</code>	definiert CSS-Angaben zur Formatierung der Elementinstanz

2. Zur Internationalisierung

<code>dir</code>	gibt die Schreibrichtung für Text in der Elementinstanz an
<code>lang</code>	gibt die verwendete Landessprache (nach RFC 1766) an
<code>translate</code>	spezifiziert, ob Inhalte bei Lokalisierung zu übersetzen sind

HTML

Universalattribute (*Global Attributes*) (Fortsetzung)

Universalattribute sind in allen HTML-Elementen verwendbar. Einteilung und Beispiele:

3. Zum Event-Handling [[W3C REC 6.1.5.2](#)]

4. Für eigene Daten (*Custom Data Attributes*) [[W3C REC 3.2.5.9](#)] [[w3schools](#)]

HTML

Universalattribute (*Global Attributes*) (Fortsetzung)

Universalattribute sind in allen HTML-Elementen verwendbar. Einteilung und Beispiele:

3. Zum Event-Handling [[W3C REC 6.1.5.2](#)]

<code>onclick</code>	Ausführen von Script-Code beim Anklicken der Elementinstanz
<code>onkeydown</code>	Ausführen von Script-Code beim Herunterdrücken einer Taste
<code>onmouseover</code>	Ausführen von Script-Code beim Überfahren der Elementinstanz

4. Für eigene Daten (*Custom Data Attributes*) [[W3C REC 3.2.5.9](#)] [[w3schools](#)]

<code>data-*</code>	Semantik definiert durch Programmierer der Web-Site
---------------------	---

Attributnamen müssen mit “data-” beginnen, XML-kompatibel sein und dürfen keine Großbuchstaben enthalten.

Cascading Stylesheets CSS

Einordnung

Ziele von CSS [\[W3C\]](#) :

1. leistungsfähige Layout-Definition für HTML-Dokumente
2. Anpassung an verschiedene Ausgabegeräte/-medien
3. zentrales Layout-Management

Cascading Stylesheets ermöglichen für HTML-Dokumente eine Trennung zwischen Inhalt und Darstellung.

Bemerkungen:

- ❑ CSS kompakt:
 1. Historie
 2. Einbindung in HTML
 3. Stylesheet-Regeln
 4. Selektoren
 5. Deklarationen
 6. Layout
 7. Verarbeitungsstrategie

Cascading Stylesheets CSS

Historie

1996 CSS Level 1. Recommendation.

2011 CSS Level 2. Recommendation. [W3C [REC](#), [css home](#)]

2016 CSS Level 3. (einige Module als Recommendation)

2016 CSS Level 4. (einige Module als Working Draft) [W3C [status](#)]

Bemerkungen:

- ❑ Das Wort „Cascading“ bezieht sich auf die kombinierte Anwendung mehrerer Stylesheets. Konflikte zwischen anwendbaren Layout-Vorgaben (d.h. durch die Layout-Vorgaben werden einer Elementinstanz für eine Property unverträgliche Werte zugewiesen) werden mit Rücksicht auf Ursprung, Gewichtung und Spezialisierungsgrad gelöst. [\[W3C\]](#)
- ❑ Die Entwicklung der Cascading Style Sheets geschieht in „Leveln“ (nicht Versionen), die aufeinander aufbauen. Dabei stellen die Features eines höheren Levels eine Übermenge der Features eines niedrigeren Levels dar. Das erlaubte Verhalten für ein Feature in einem höheren Level muss jedoch präziser definiert bzw. weniger tolerant als in einem niedrigeren Level definiert sein. [\[W3C\]](#)
- ❑ CSS 3 Demos. [\[MDN overview, 1, 2\]](#)

Cascading Stylesheets CSS

Einbindung von Stylesheet-Information [\[SELFHTML\]](#)

Die Einbindung bzw. Deklaration von Stylesheet-Information kann auf folgende, miteinander kombinierbare Arten geschehen:

1. Stylesheet-Deklaration in eigener CSS-Datei
2. Stylesheet-Deklaration zentral im HTML-Dokument
3. *Styleattribut*-Deklaration im Start-Tag einer Elementinstanz

Cascading Stylesheets CSS

Einbindung von Stylesheet-Information [\[SELFHTML\]](#)

Die Einbindung bzw. Deklaration von Stylesheet-Information kann auf folgende, miteinander kombinierbare Arten geschehen:

1. Stylesheet-Deklaration in eigener CSS-Datei

```
<link rel="stylesheet" type="text/css" href="../share/bib.css">
```

Das `<link>`-Element darf nur im `<head>`-Element verwendet werden.

2. Stylesheet-Deklaration zentral im HTML-Dokument

```
<style type="text/css">  
  h3 {color: red; font: arial}  
</style>
```

Das `<style>`-Element darf in dieser Form nur im `<head>`-Element verwendet werden.

3. *Styleattribut*-Deklaration im Start-Tag einer Elementinstanz

Cascading Stylesheets CSS

Einbindung von Stylesheet-Information [\[SELFHTML\]](#)

Die Einbindung bzw. Deklaration von Stylesheet-Information kann auf folgende, miteinander kombinierbare Arten geschehen:

1. Stylesheet-Deklaration in eigener CSS-Datei

```
<link rel="stylesheet" type="text/css" href="../share/bib.css">
```

Das `<link>`-Element darf nur im `<head>`-Element verwendet werden.

2. Stylesheet-Deklaration zentral im HTML-Dokument

```
<style type="text/css">  
  h3 {color: red; font: arial}  
</style>
```

Das `<style>`-Element darf in dieser Form nur im `<head>`-Element verwendet werden.

3. *Styleattribut*-Deklaration im Start-Tag einer Elementinstanz

```
<h3 style="color: red; font: arial">Neues Kapitel</h3>
```

Die Syntax des **Attributwertes** entspricht dem Deklarationsteil einer CSS-Regel.

Bemerkungen:

- ❑ Der Geltungsbereich von Stylesheet-Deklarationen, die aus einer CSS-Datei eingebunden werden, ist global für das HTML-Dokument. Die CSS-Datei muss die Namensendung `.css` haben.
- ❑ Der Geltungsbereich von Stylesheet-Deklarationen, die im `<head>`-Element gemacht werden, ist global: die Deklarationen beziehen sich auf das gesamte HTML-Dokument.
- ❑ HTML5 erlaubt Stylesheet-Deklarationen im `<body>`-Element, wenn das `scoped`-Attribut gesetzt ist.
- ❑ Der Geltungsbereich von *Styleattribut*-Deklarationen ist die Elementinstanz selbst einschließlich ihrer Kindelemente.
- ❑ Im Konfliktfall haben lokale Deklarationen Vorrang vor globalen.
- ❑ Stylesheet-Deklarationen innerhalb eines HTML-Dokuments widersprechen dem Paradigma der Trennung von Inhalt und Darstellung.

Bemerkungen (Medientypen) :

- ❑ Durch Angabe eines `media`-Attributs im `<link>`-Tag lassen sich Medientypen wie `screen`, `print`, `aural`, `braille`, `handheld`, `tv`, `tty` oder `all` spezifizieren. Je nach Endgerät werden vom Browser passende Stylesheets ausgewählt.
- ❑ Stylesheet-Dateien lassen sich kombinieren. Beispiel:

```
<link rel="stylesheet" href="base.css">
<link rel="stylesheet" href="print.css" media="print">
<link rel="stylesheet" href="screen1a.css" media="screen">
<link rel="stylesheet" href="screen1b.css" media="screen">
<link rel="alternate stylesheet" href="screen2.css" media="screen" title=...>
```



Im Beispiel gilt

- `base.css` für alle Medientypen.
 - `print.css` zusätzlich für den Medientyp `print`. Die enthaltenen CSS-Regeln werden beispielsweise bei der Erzeugung der Druckvorschau berücksichtigt.
 - `screen1a.css` und `screen1b.css` oder **alternativ** `screen2.css` zusätzlich für den Medientyp `screen`.
- ❑ Alternative Stylesheets für den Medientyp `screen` können meist im Browsermenü ausgewählt werden. Als Menüeintrag wird der Text des `title`-Attributs verwendet.

Cascading Stylesheets CSS

Stylesheet-Regeln [\[SELFHTML\]](#)

Ein Stylesheet ist eine Sammlung von Layout-Regeln. Eine Layout-Regel ist wie folgt aufgebaut:

<code>h1, h2, table</code>	<code>{ color: red; font: arial }</code>
	
(drei) Selektoren	Deklaration

- Mittels des Selektors werden passende Elementinstanzen ausgewählt. Mehrere Selektoren können – durch Kommata getrennt – in einer Liste angegeben sein.
- Der Deklarationsteil enthält – durch Semikola getrennt – die Layout-Vorgaben jeweils in Form von Property-Value-Paaren.

Cascading Stylesheets CSS

Selektoren [[W3C selectors level 3](#), [level 4](#)] [[MDN](#)]

1. Elementtypselektoren [[W3C](#)]
2. Attributselektoren [[W3C](#)]
3. Klassenselektoren [[W3C](#)]
4. ID-Selektoren [[W3C](#)]

Cascading Stylesheets CSS

Selektoren [[W3C selectors level 3](#), [level 4](#)] [[MDN](#)]

1. Elementtypselektoren [[W3C](#)]

```
h1 {color: red}
```

```
<h1> ... </h1>
```

2. Attributselektoren [[W3C](#)]

3. Klassenselektoren [[W3C](#)]

4. ID-Selektoren [[W3C](#)]

Cascading Stylesheets CSS

Selektoren [\[W3C selectors level 3, level 4\]](#) [\[MDN\]](#)

1. Elementtypselektoren [\[W3C\]](#)

```
h1 {color: red} <h1> ... </h1>
```

2. Attributselektoren [\[W3C\]](#)

```
h1[hreflang=fr] {color: red} <h1 hreflang="fr"> ... </h1>
```

3. Klassenselektoren [\[W3C\]](#)

4. ID-Selektoren [\[W3C\]](#)

Cascading Stylesheets CSS

Selektoren [\[W3C selectors level 3, level 4\]](#) [\[MDN\]](#)

1. Elementtypselektoren [\[W3C\]](#)

```
h1 {color: red} <h1> ... </h1>
```

2. Attributselektoren [\[W3C\]](#)

```
h1[hreflang=fr] {color: red} <h1 hreflang="fr"> ... </h1>
```

3. Klassenselektoren [\[W3C\]](#)

```
.Classname {color: red} <div class="Classname"> ... </div>  
h1.Classname {color: red} <h1 class="Classname"> ... </h1>
```

4. ID-Selektoren [\[W3C\]](#)

Cascading Stylesheets CSS

Selektoren [\[W3C selectors level 3\]](#), [\[level 4\]](#) [\[MDN\]](#)

1. Elementtypselektoren [\[W3C\]](#)

```
h1 {color: red} <h1> ... </h1>
```

2. Attributselektoren [\[W3C\]](#)

```
h1[hreflang=fr] {color: red} <h1 hreflang="fr"> ... </h1>
```

3. Klassenselektoren [\[W3C\]](#)

```
.Classname {color: red} <div class="Classname"> ... </div>  
h1.Classname {color: red} <h1 class="Classname"> ... </h1>
```

4. ID-Selektoren [\[W3C\]](#)

```
#Identifier {color: red} <h1 id="Identifier"> ... </h1>
```

Cascading Stylesheets CSS

Selektoren (Fortsetzung)

5. Pseudo-Klassenselektoren [\[W3C\]](#)
6. Pseudo-Elementselektoren [\[W3C\]](#)
7. Kombinierte Selektoren [\[W3C\]](#)

Cascading Stylesheets CSS

Selektoren (Fortsetzung)

5. Pseudo-Klassenselektoren [\[W3C\]](#)

```
a:visited {color: red}
```

```
<a> ... </a>
```

6. Pseudo-Elementselektoren [\[W3C\]](#)

7. Kombinierte Selektoren [\[W3C\]](#)

Cascading Stylesheets CSS

Selektoren (Fortsetzung)

5. Pseudo-Klassenselektoren [\[W3C\]](#)

```
a:visited {color: red}      <a> ... </a>
```

6. Pseudo-Elementselektoren [\[W3C\]](#)

```
p::first-line {color: red} <p> ... </p>
```

7. Kombinierte Selektoren [\[W3C\]](#)

Cascading Stylesheets CSS

Selektoren (Fortsetzung)

5. Pseudo-Klassenselektoren [\[W3C\]](#)

```
a:visited {color: red} <a> ... </a>
```

6. Pseudo-Elementselektoren [\[W3C\]](#)

```
p::first-line {color: red} <p> ... </p>
```

7. Kombinierte Selektoren [\[W3C\]](#)

```
h1 em {color: red} <h1>This is <em>very</em> ... </h1>  
thead > tr {color: orange} <thead> <tr> ... </thead> \[Beispiel\]
```

Bemerkungen:

- ❑ Pseudoklassen ermöglichen die Selektion auf Basis von Information, die nicht oder nur versteckt im Dokumentenbaum abgebildet ist. Diese Information kann sich auf folgende Aspekte beziehen:
 - Links und Lokationen [\[W3C\]](#)
 - Benutzeraktionen [\[W3C\]](#)
 - Zeitverläufe [\[W3C\]](#)
 - linguistische Konzepte [\[W3C\]](#)
 - Eingabezustände [\[W3C\]](#)
 - Dokumentbaumstruktur [\[W3C\]](#)

- ❑ Pseudo*elemente* ermöglichen die Selektion von Dokumenteninhalten, die nicht durch die Markup-Sprache ausgezeichnet werden kann. Beispiele: der erste Buchstabe eines Abschnitts, die letzte Zeile eines Abschnitts. [\[W3C\]](#)

- ❑ Unter die Rubrik der kombinierten Selektoren fällt insbesondere die Spezifikation von Nachbarschaftsbeziehungen im Dokumentenbaum. Beispiele: > (ist Kindknoten von), ~ (ist Geschwisterknoten von), _ (ist Nachfolger von).

Cascading Stylesheets CSS

Selektoren (Fortsetzung)

Mit Hilfe von Klassenselektoren kann die Einführung neuer Elementtypen nachempfunden werden.

Besonders geeignet sind die **funktionslosen** HTML-Elementtypen [W3C wiki 1, 2]

1. Block-Elementtyp `<div>`
2. Inline-Elementtyp ``

Cascading Stylesheets CSS

Selektoren (Fortsetzung)

Mit Hilfe von Klassenselektoren kann die Einführung neuer Elementtypen nachempfunden werden.

Besonders geeignet sind die **funktionslosen** HTML-Elementtypen [W3C wiki [1](#), [2](#)]

1. Block-Elementtyp `<div>`
2. Inline-Elementtyp ``

Beispiel:

```
<head>
  <title>Example</title>
  <style type="text/css">
    .myheading {font-family: sans-serif; color: blue}
  </style>
</head>
<body>
  <div class="myheading">Ein eigener Heading-Stil</div>
  ...
```

Bemerkungen:

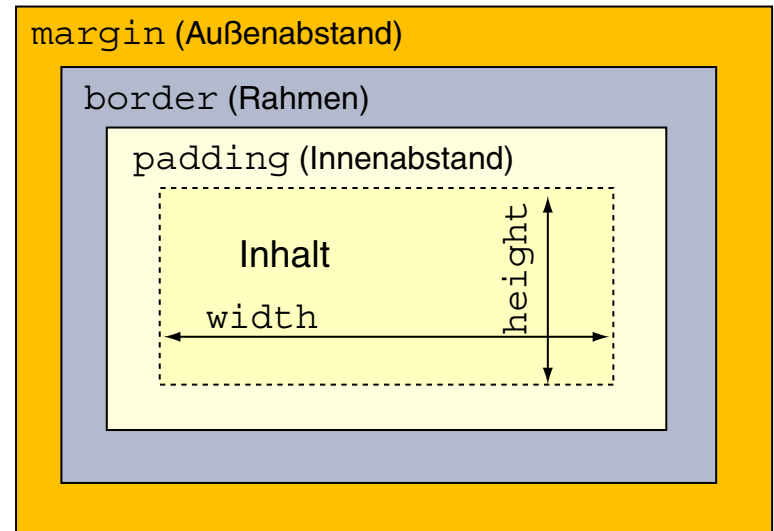
- ❑ HTML verwendet eine feste Dokumentstruktur und somit sind alle Elementtypen vorgegeben. Diese HTML-Elementtypen besitzen elementtypspezifische Vorgaben für ihre Darstellung.
- ❑ Die Schaffung von Elementinstanzen, die zu einer gemeinsamen Stylesheet-Klasse gehören und darüberhinaus ohne weitere Funktion (= ohne intendierte Semantik) sind, geschieht in zwei Schritten:
 1. Definition einer neuen Stylesheet-Klasse mittels `.Classname { ... }`.
 2. Verwendung der neuen Stylesheet-Klasse mittels `class="Classname"` in Elementinstanzen des Typs `<div>` oder ``.
- ❑ Das W3C warnt vor der übertriebenen Nutzung dieser Möglichkeit, weil die intendierte Semantik selbstdefinierter Klassen für Außenstehende oft nicht erkennbar ist. [\[W3C\]](#)

Cascading Stylesheets CSS

Deklarationen

CSS-Deklarationen können sich auf nahezu alle Aspekte der Dokumentgestaltung beziehen. [[SELFHTML](#)]

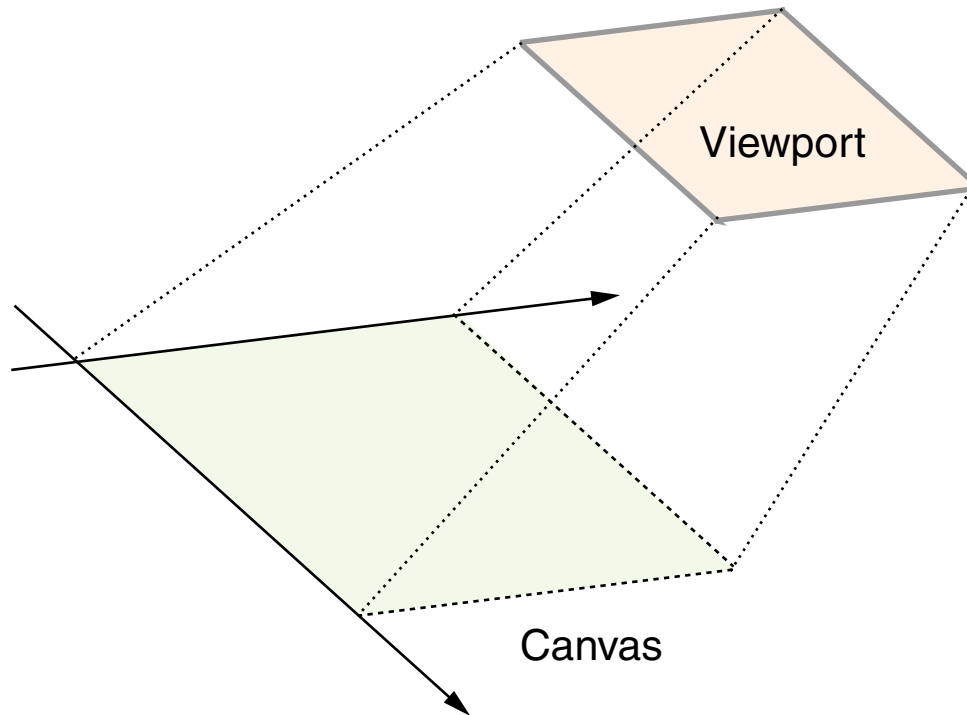
- ❑ Maßeinheiten [[SELFHTML](#)]
- ❑ Schrift [[MDN](#)]
- ❑ Farben [[MDN](#)]
- ❑ Listen [[MDN](#)]
- ❑ Tabellen [[MDN](#)] [[webis aitoools](#), [teaching](#)]
- ❑ Box-Modell [[W3C](#)] [[MDN](#)] [[SELFHTML](#)]



Cascading Stylesheets CSS

Layout (*Visual Formatting*) [W3C]

Das Rendering eines Dokuments erfolgt auf einer unendlich großen Zeichenfläche (*Canvas*). Der Viewport zeigt einen Ausschnitt der Zeichenfläche.



Cascading Stylesheets CSS

Layout (*Visual Formatting*) (Fortsetzung)

❑ display-Property [\[W3C\]](#)

<code>block</code>	Element erzeugt eine Block-Box.
<code>inline</code>	Element erzeugt eine oder mehrere Inline-Boxen.
<code>inline-block</code>	Element erzeugt eine nicht teilbare Inline-Box, der Inhalt wird im Block-Kontext formatiert.

❑ position-Property [\[W3C\]](#)

<code>static</code>	Box wird im vorliegenden Kontext (Block/Inline) gesetzt.
<code>relative</code>	Box wird mit Offset relativ zur normalen Position gesetzt.
<code>absolute</code>	Box wird an absoluter Position im enthaltenen Block gesetzt.
<code>fixed</code>	Box wird an absoluter Position im enthaltenen Block gesetzt, aber in der Viewport-Ebene. (Position fest im Viewport)

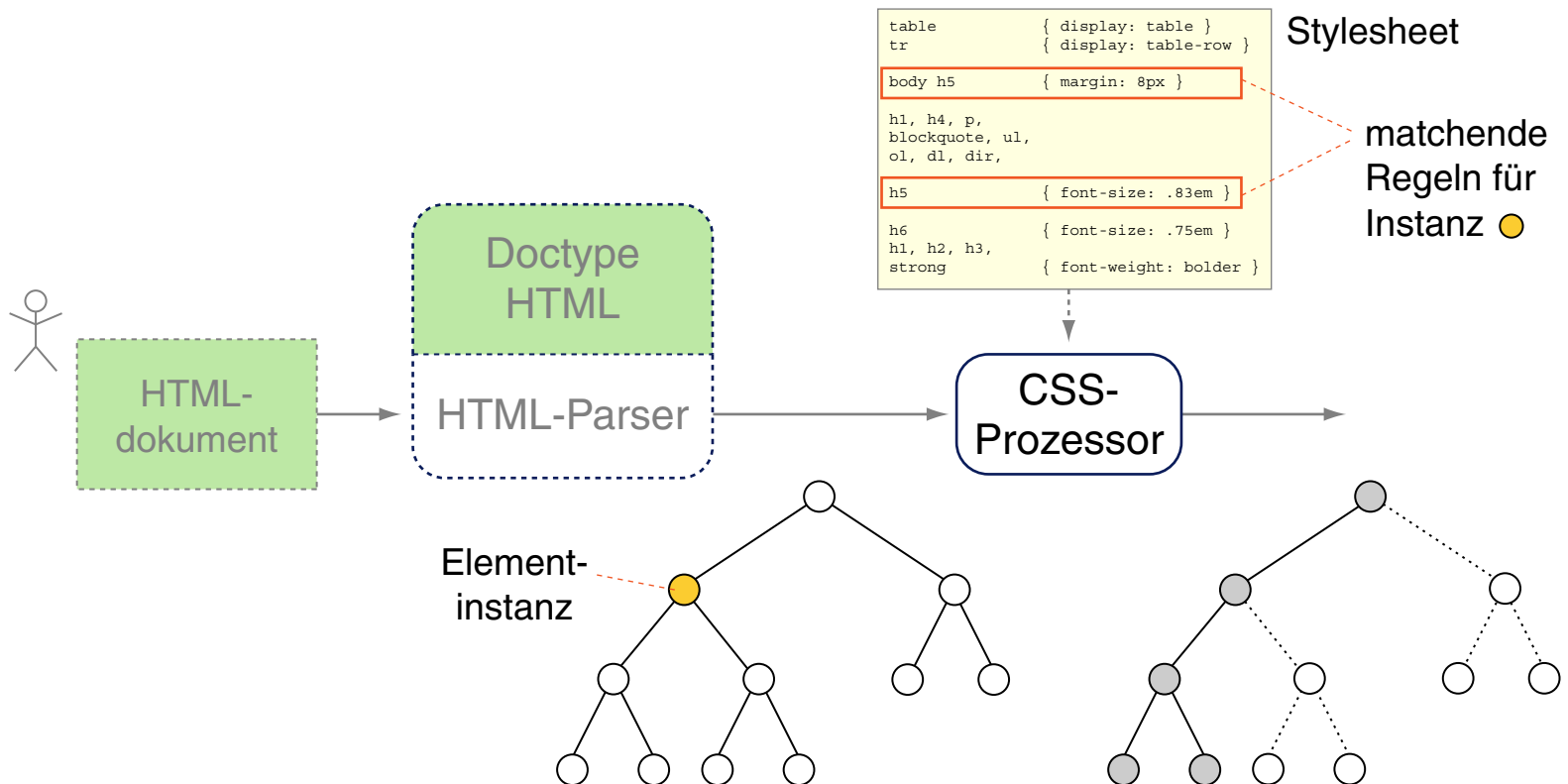
❑ **Offsets** `top`, `right`, `bottom`, `left` [\[W3C\]](#)

Angabe der Position als Abstand vom jeweiligen Rand der enthaltenden Box (`absolute/fixed`) oder von der vorgesehenen Position der Box (`relative`).

Cascading Stylesheets CSS

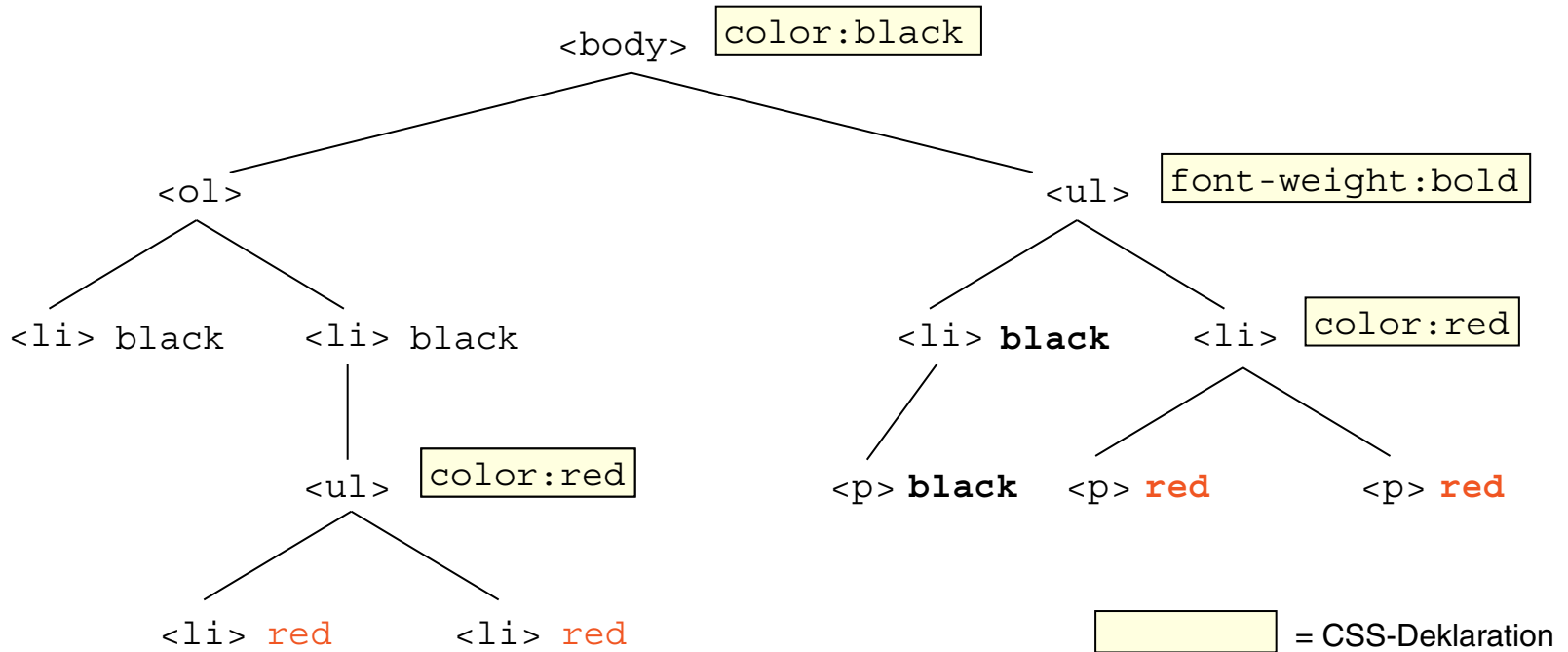
Verarbeitungsstrategie [WT:III [XSL-Verarbeitung](#)]

Bei der Darstellung einer Elementinstanz werden in den zugehörigen Stylesheets anhand der Selektoren die matchenden Regeln identifiziert. Ihre Anwendung geschieht in der Reihenfolge von den weniger zu den mehr spezifischen.



Cascading Stylesheets CSS

Verarbeitungsstrategie (Fortsetzung)



□ Deklarationen werden an eingebettete Elementinstanzen vererbt.

□ Lokale Vorgaben überschreiben vererbte Werte und Defaults:

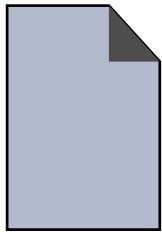
`color:black` → `red`

`font-weight:normal` → `bold`

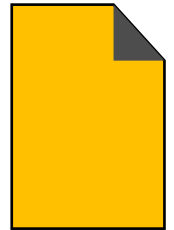
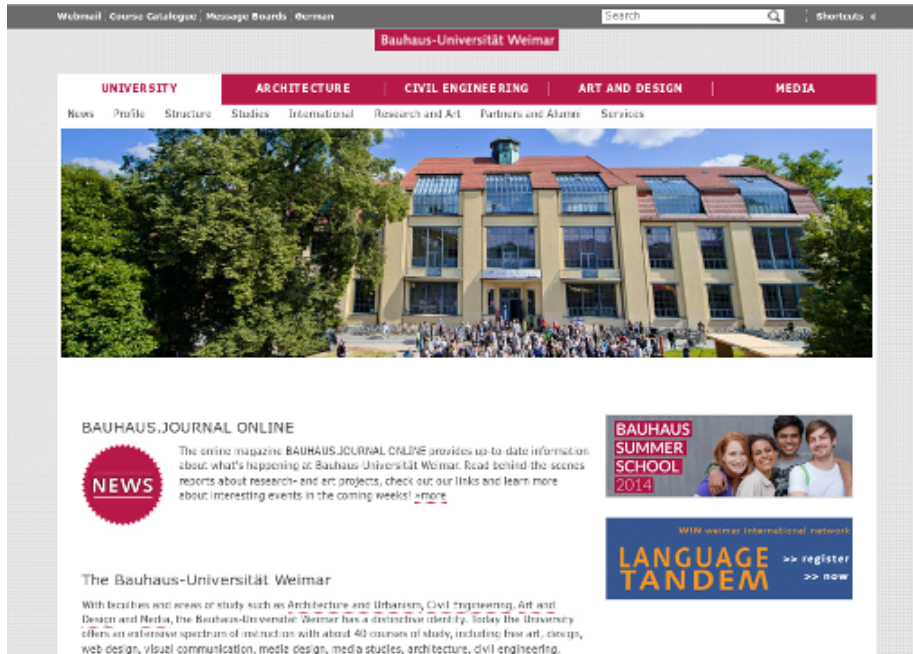
Cascading Stylesheets CSS

Verarbeitungsstrategie (Fortsetzung) [W3C] [SELFHTML]

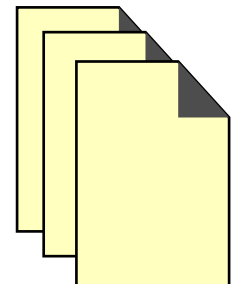
Für die Darstellung von HTML-Dokumenten werden drei Arten von Stylesheets ausgewertet:



Browser-Stylesheet



Benutzer-Stylesheet



Autoren-Stylesheets

Cascading Stylesheets CSS

Verarbeitungsstrategie (Fortsetzung)

Für die Darstellung von HTML-Dokumenten werden drei Arten von Stylesheets ausgewertet:

1. Browser-Stylesheet

Definiert das Standard-Layout für die Elementinstanzen; dieses Stylesheet ist Browser-spezifisch und wird vom Browser-Hersteller entwickelt. Ein entsprechender W3C-Vorschlag befindet hier [\[W3C\]](#).

2. Benutzer-Stylesheet

Definiert die Präferenzen eines Benutzers. Die Spezifikation des Benutzer-Stylesheets geschieht über einen Browser-Dialog.

3. Autoren-Stylesheet(s)

Die „eigentlichen“ (sichtbaren) Stylesheets, die ein Autor eines HTML-Dokuments zur Realisierung seiner Layout-Ziele entwickelt hat.

Bemerkungen:

- ❑ Regeln lassen sich durch Angabe von `!important` stärker gewichten. Im Beispiel ist das Setzen der Property `font-style` stärker gewichtet:

```
p {  
    font-style: italic !important  
    color: red;  
}
```

- ❑ Eine Gewichtung ist nur für Autoren- und Benutzer-Stylesheets spezifizierbar.
- ❑ Konflikte zwischen anwendbaren Layout-Vorgaben werden zunächst mit Rücksicht auf Ursprung und Gewichtung gelöst [\[W3C\]](#) :
 1. `!important`-Regeln aus Benutzer-Stylesheet
 2. `!important`-Regeln aus Autoren-Stylesheets
 3. normale Regeln aus Autoren-Stylesheets
 4. normale Regeln aus Benutzer-Stylesheet
 5. Regeln aus Browser-Stylesheet

Bestehen immer noch Konflikte, so werden diese mit Rücksicht auf

6. Spezialisierungsgrad (speziellere Regeln vor allgemeineren) und
7. Reihenfolge (spätere Regeln vor früheren) gelöst.

HTML, CSS

Quellen zum Nachlernen und Nachschlagen im Web: HTML

- ❑ MDN. *HTML*.
developer.mozilla.org/en-US/docs/Web/HTML
- ❑ SELFHTML e.V. *SELFHTML*.
www.selfhtml.org, wiki.selfhtml.org
- ❑ WHATWG. *HTML, Living Standard*.
www.whatwg.org
- ❑ W3C. *HTML5, Recommendation*.
www.w3.org/TR/html5/
- ❑ W3C. *HTML Wiki*.
www.w3.org/wiki/Category:HTML
- ❑ W3 Schools. *HTML Reference*.
www.w3schools.com/tags

HTML, CSS

Quellen zum Nachlernen und Nachschlagen im Web: CSS

- ❑ MDN. *CSS*.
developer.mozilla.org/en-US/docs/Web/CSS
- ❑ MDN. *CSS Tutorial*.
developer.mozilla.org/en-US/docs/Web/Guide/CSS/Getting_started
- ❑ W3C. *CSS Level 2*.
www.w3.org/TR/CSS2
- ❑ W3C. *CSS Home*.
www.w3.org/Style/CSS
- ❑ W3 Schools. *CSS*.
www.w3schools.com/css

HTML, CSS

Quellen zum Nachlernen und Nachschlagen im Web: Werkzeuge

- ❑ Flanders. *Web Pages That Suck*. (Web-Design)
www.webpagesthatsuck.com
- ❑ W3C. *HTML Tidy*. (standardisieren und säubern von HTML-Code)
<https://www.htacg.org/tidy-html5>, www.w3.org/People/Raggett/tidy
- ❑ W3C. *Markup Validation Service*.
validator.w3.org

III. Dokumentensprachen

- ❑ Auszeichnungssprachen
- ❑ HTML
- ❑ Cascading Stylesheets CSS
- ❑ XML-Grundlagen
- ❑ XML-Schema
- ❑ Die XSL-Familie
- ❑ APIs für XML-Dokumente

XML-Grundlagen

```
<!DOCTYPE html>
```

```
<html>
  <head>
    <meta http-equiv="content-type" content="text/html; charset=ISO-8859-1">
    <title>Person</title>
  </head>

  <body>
    <h3>Alan Turing</h3>
    <p>
      23. Juni 1912* <br>
      Mathematiker, Informatiker
    </p>
  </body>
</html>
```

XML-Grundlagen

```
<!DOCTYPE html>
```

```
<html>
  <head>
    <meta http-equiv="content-type" content="text/html; charset=ISO-8859-1">
    <title>Person</title>
  </head>
  <body>
    <h3>Alan Turing</h3>
    <p>
      23. Juni 1912* <br>
      Mathematiker, Informatiker
    </p>
  </body>
</html>
```

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

```
<person>
  <name>
    <vorname>Alan</vorname>
    <nachname>Turing</nachname>
  </name>
  <geburtstag>23. Juni 1912</geburtstag>
  <beruf>Mathematiker</beruf>
  <beruf>Informatiker</beruf>
</person>
```

XML-Grundlagen

```
<!DOCTYPE html>
```

```
<html>
  <head>
    <meta http-equiv="content-type" content="text/html; charset=ISO-8859-1">
    <title>Person</title>
  </head>

  <body>
    <h3>Alan Turing</h3>
    <p>
      23. Juni 1912* <br>
      Mathematiker, Informatiker
    </p>
  </body>
</html>
```

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

```
<person>
  <name>
    <vorname>Alan</vorname>
    <nachname>Turing</nachname>
  </name>

  <geburtstag>23. Juni 1912</geburtstag>

  <beruf>Mathematiker</beruf>
  <beruf>Informatiker</beruf>
</person>
```

Keine operationale Semantik:

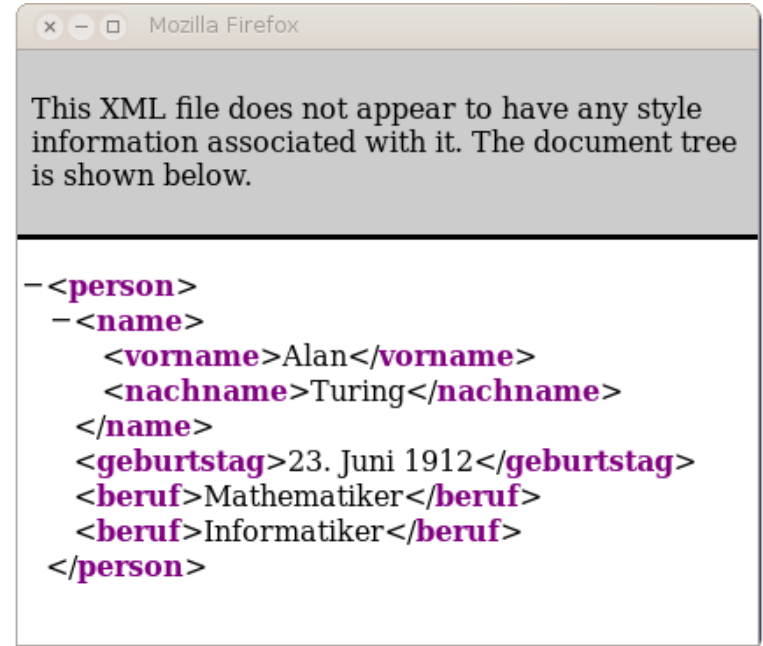
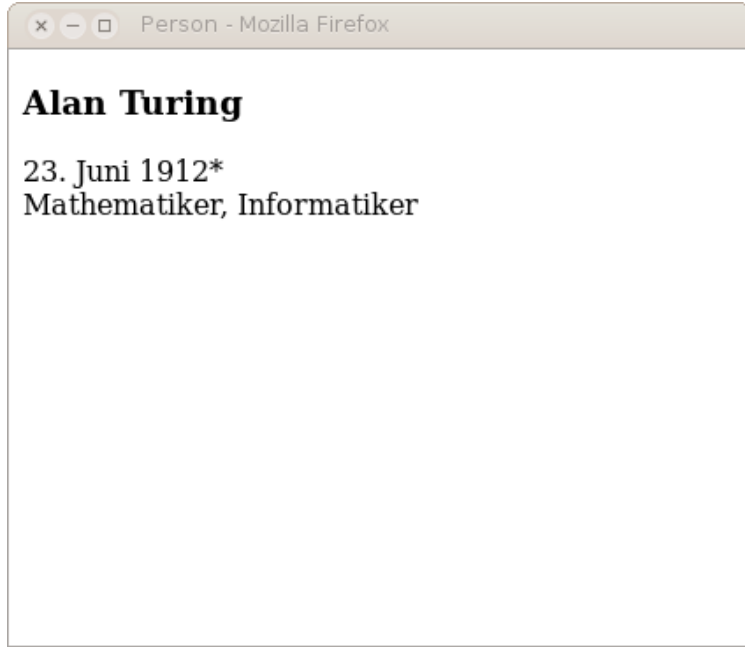
```
<?xml version="1.0" standalone="no" ?>
<!DOCTYPE person SYSTEM "person.dtd">
```

```
<person>
  <address>
    <zip>Alan</zip>
    <city>Turing</city>
  </address>

  <weight>23. Juni 1912</weight>

  <name>Mathematiker</name>
  <name>Informatiker</name>
</person>
```

XML-Grundlagen



In XML können beliebige Elementtypen definiert und deklariert werden. Mittels einer DTD (*Document Type Definition*) lassen sich strukturelle Constraints (= Inhaltsmodelle) für die Verwendung von Elementinstanzen vorschreiben.

Bemerkungen:

- ❑ XML kompakt:
 1. Historie
 2. Prinzip der Dokumentenverarbeitung
 3. Aufbau eines XML-Dokuments
 4. Weitere Regeln zur Syntax
 5. Wohlgeformtheit und Validität
 6. XML Document Type Definition, DTD
 7. Internationalisierung
 8. Namensräume
 9. XML Information Set

XML-Grundlagen

Historie: zentrale XML-Spezifikationen

- 2008 XML 1.0, Recommendation. [W3C [REC](#)]
- 2006 XML 1.1, Recommendation. [W3C [REC](#), [status](#)]

- 2004 XML-Schema Part 0: Primer, Recommendation. [W3C [REC](#)]
- 2012 XML-Schema (XSD) 1.1 Part 1: Structures. [W3C [REC](#)]
- 2012 XML-Schema (XSD) 1.1 Part 2: Datatypes. [W3C [REC](#), [status](#)]

- 2014 XSL Transformations (XSLT) 3.0. Latest Working Draft. [W3C [WD](#), [status](#)]
- 2014 XML Path Language (XPath) 3.0. Recommendation. [W3C [REC](#), [status](#)]
- 2014 XML Query Language (XQuery) 3.0. Recommendation. [W3C [REC](#), [status](#)]
- 2012 XSL Formatting Objects (XSL-FO) 2.0. Working Draft. [W3C [WD](#), [status](#)]

XML-Grundlagen

Historie: weitere bekannte XML-Spezifikationen

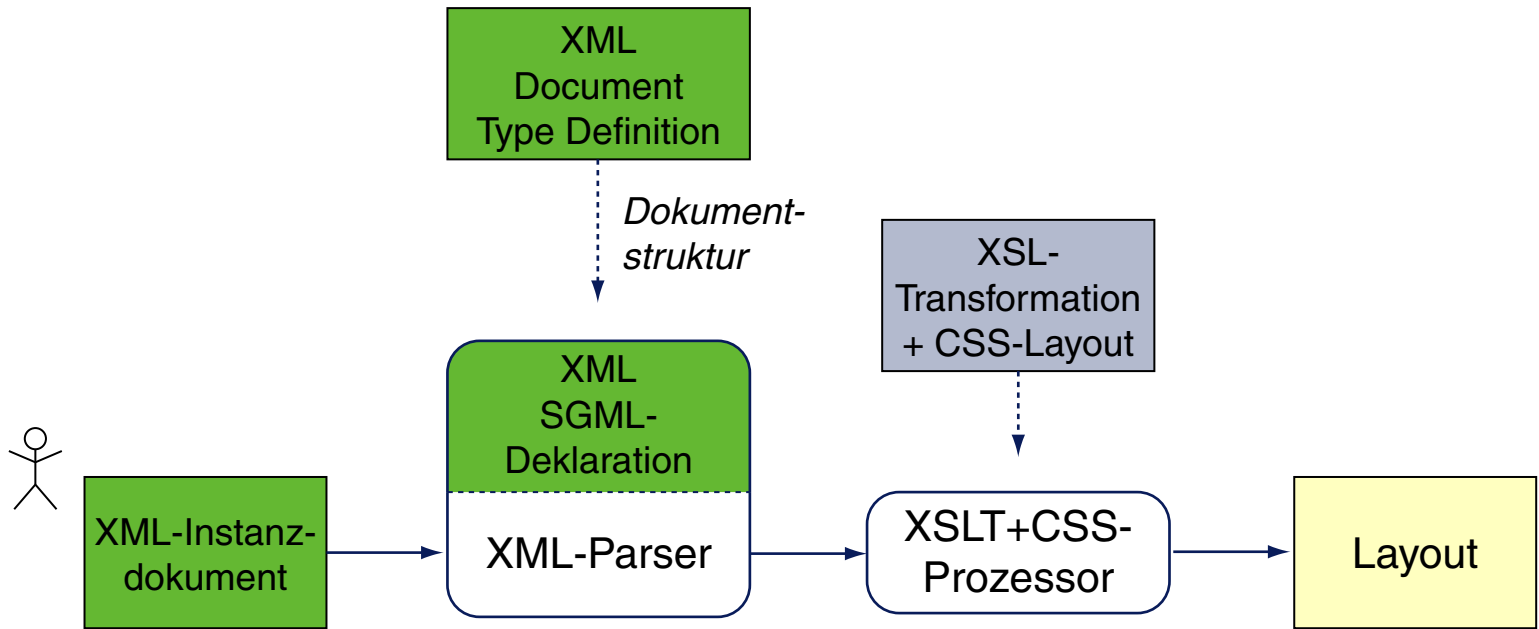
- 2014 MathML 3.0. Markup-Sprache für mathematische Notation. [W3C [REC](#), [status](#)]
- 2015 SVG 2. Markup-Sprache für zweidimensionale Vektorgrafik.
[W3C [WD](#), [status](#), [home](#)]
- 2014 RDF 1.1. Generisches Beschreibungs-Framework. [W3C [REC](#), [status](#)]
- 2007 WSDL 2.0. Beschreibungssprache für Web-Services. [W3C [REC](#), [status](#)]

Bemerkungen:

- ❑ Viele Entwickler forderten eine erweiterbare Markup-Sprache, die weniger kompliziert als SGML ist. Tatsächlich ist die Spezifikation von XML deutlich einfacher und kürzer als die von SGML.
- ❑ XML-Dokumente werden nicht nur in Web-Anwendungen genutzt. Aufgrund seiner Flexibilität kann XML den Publishing-Anforderungen von Büchern, Zeitschriften, Katalogen, Postern etc. gerecht werden. Darüberhinaus hat XML als generelles Austauschformat große Verbreitung erlangt und stellt die wichtigste Sprache zur Kodierung von RDF-Graphen dar.
- ❑ Übersicht über die Ziele von XML: [\[W3C\]](#)
- ❑ Wiederholung: Der Standardisierungsprozess der W3C ist formalisiert und spiegelt sich in den verschiedenen Levels der veröffentlichten Reports wider. [\[W3C level\]](#)

XML-Grundlagen

XML Dokumentenverarbeitung

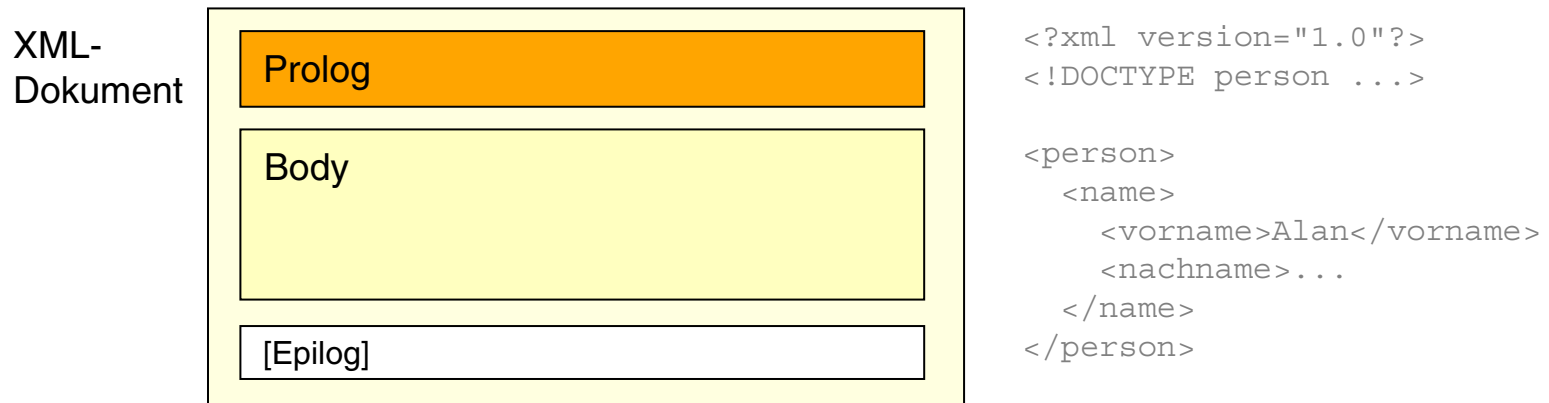


Vergleiche hierzu

- ❑ die SGML Dokumentenverarbeitung
- ❑ und die HTML Dokumentenverarbeitung.

XML-Grundlagen

XML-Dokument



- ❑ Zum Prolog zählt alles vor dem Start des XML-Wurzelementes; der Prolog enthält Meta-Informationen über das Dokument.
- ❑ Der Body besteht aus ineinander geschachtelten XML-Elementen.
- ❑ Der Epilog enthält Kommentare und Verarbeitungsanweisungen für das Dokument; er ist optional.
- ❑ Vergleiche hierzu die [HTML-Dokumentstruktur](#).

XML-Grundlagen

Dokumenten-Prolog

1. XML-Deklaration.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
```

Jeder XML-Prolog startet mit einer XML-Deklaration.

XML-Grundlagen

Dokumenten-Prolog

1. XML-Deklaration.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
```

Jeder XML-Prolog startet mit einer XML-Deklaration.

2. XML-Dokumenttyp-Deklaration.

```
<!DOCTYPE Wurzelelementname SYSTEM "URI">
```

Anstelle oder kombiniert mit der Referenz auf eine Datei lassen sich DTD-Befehle auch im Dokument einbinden. Man spricht von der **externen** bzw. der **internen** DTD-Teilmenge (*external subset, internal subset*); zusammen bilden sie die vollständige DTD.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
```

```
<!DOCTYPE poem [  
  <!ELEMENT poem (#PCDATA)>  
  <!ENTITY author "William Shakespeare">
```

```
]>
```

```
<poem>
```

```
  Dieses Gedicht stammt von &author;
```

```
</poem>
```

XML-Grundlagen

Dokumenten-Prolog

1. XML-Deklaration.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
```

Jeder XML-Prolog startet mit einer XML-Deklaration.

2. XML-Dokumenttyp-Deklaration.

```
<!DOCTYPE Wurzelelementname SYSTEM "URI">
```

Anstelle oder kombiniert mit der Referenz auf eine Datei lassen sich DTD-Befehle auch im Dokument einbinden. Man spricht von der **externen** bzw. der **internen** DTD-Teilmenge (*external subset, internal subset*); zusammen bilden sie die vollständige DTD.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
```

```
<!DOCTYPE poem [  
  <!ELEMENT poem (#PCDATA)>  
  <!ENTITY author "William Shakespeare">
```

```
]>
```

```
<poem>
```

```
  Dieses Gedicht stammt von &author;
```

```
</poem>
```

3. Verarbeitungsanweisung für XML-Stylesheets.

```
<?xml-stylesheet type="text/css" href="person.css"?>
```


Bemerkungen:

- ❑ `standalone="no"` bedeutet, dass der Rückgriff auf eine externe DTD erforderlich ist.
- ❑ Die Dokumenttyp-*Deklaration* enthält eine Referenz auf (= deklariert) eine DTD (*Document Type Definition*), die wiederum Definitionen und Deklarationen für Elemente und Attribute enthält.
- ❑ Die Dateiendung einer DTD-Datei ist `.dtd`.
- ❑ Die Dokumenttyp-Deklaration kann entfallen.
- ❑ Bei einer weltweit bekannten DTD kann anstelle des Schlüsselwortes `SYSTEM` das Schlüsselwort `PUBLIC` zusammen mit dem Public-Identifizier dieser DTD verwendet werden; der Public-Identifizier wird durch einen lokalen Katalog-Server auf eine URL abgebildet. Sicherheitshalber ist zusätzlich noch eine URI anzugeben. In der Praxis werden Public-Identifizier kaum verwendet.

XML-Grundlagen

Dokumenten-Body

Allgemeine Form einer XML-Elementinstanz:

```
<elementname {attribute}*> ... </elementname>
```

Tags müssen balanciert sein – Ausnahme sind *Empty-Element-Tags*:

```
<elementName/>      ≡      <elementName></elementName>
```

XML-Grundlagen

Dokumenten-Body

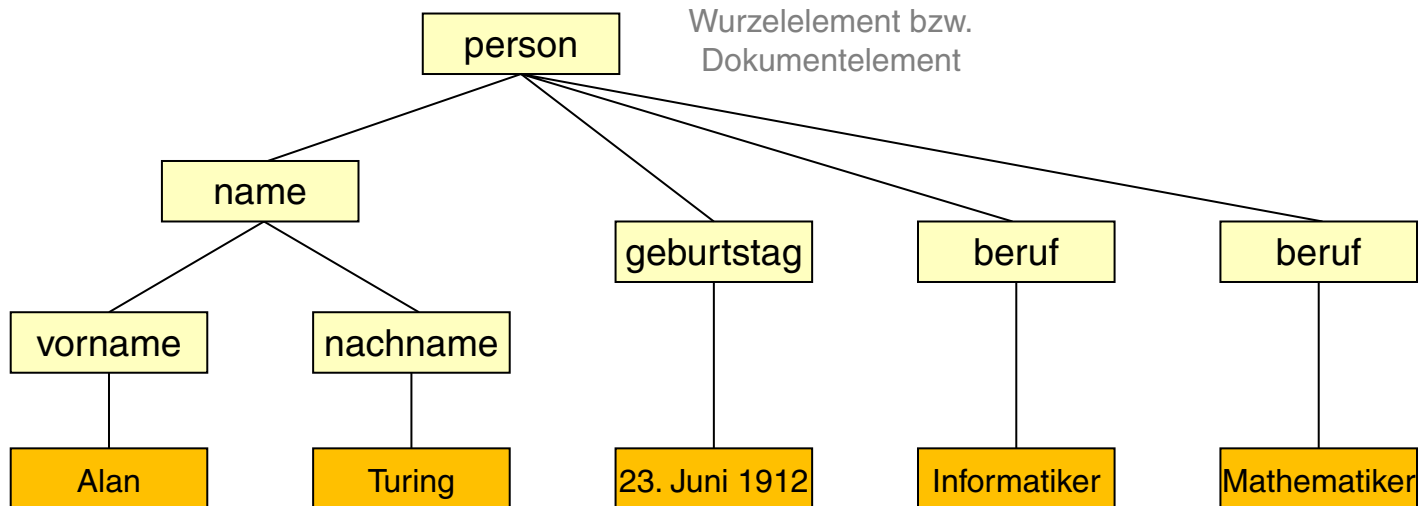
Allgemeine Form einer XML-Elementinstanz:

```
<elementname {attribute}*> ... </elementname>
```

Tags müssen balanciert sein – Ausnahme sind *Empty-Element-Tags*:

```
<elementName/> ≡ <elementName></elementName>
```

Die Elementstruktur des Bodies entspricht einem Baum. [Beispiel:](#)



XML-Grundlagen

Attribute

Verwendung von Attributen wie in SGML [WT:III SGML] :

```
<person geboren="1912-06-23" gestorben="1954-06-07">
  Alan Turing
</person>
```

Frage des Stils: **Elementmodellierung** (a) oder **Attributmodellierung** (b)

(a)

```
<person>
  <name>
    <vorname>Alan</vorname>
    <nachname>Turing</nachname>
  </name>
  <beruf>Mathematiker</beruf>
  <beruf>Informatiker</beruf>
</person>
```

(b)

```
<person>
  <name vorname="Alan" nachname="Turing"/>
  <beruf wert="Mathematiker"/>
  <beruf wert="Informatiker"/>
</person>
```

Bemerkungen:

- ❑ Notiert man Empty-Element-Tags anstatt `<elementName .../>` als `<elementName ...></elementName>`, so darf kein Whitespace (Leerzeichen, Tabulatorzeichen, Zeilenvorschub) zwischen dem Start-Tag und dem Ende-Tag stehen.
- ❑ Zum Modellierungsstil: mit Hilfe von Attributen sollten nur Meta-Daten über eine Elementinstanz spezifiziert werden. Deshalb ist im vorigen Beispiel die Elementmodellierung (a) vorzuziehen.
- ❑ Eine eindeutige Trennung zwischen Objektdaten und Meta-Daten ist nicht immer möglich.

XML-Grundlagen

Weitere Regeln zur Syntax

- ❑ XML-Namen dürfen aus beliebigen alphanumerischen, ideographischen sowie den drei Interpunktionszeichen „_“ , „-“ und „.“ bestehen.
- ❑ Entity-Referenzen wie in SGML: `&Entity-Name;` [WT:III [SGML](#)]
- ❑ **Kommentare:** `<!-- Dies ist ein Kommentar -->`

XML-Grundlagen

Weitere Regeln zur Syntax

- XML-Namen dürfen aus beliebigen alphanumerischen, ideographischen sowie den drei Interpunktionszeichen „_“ , „-“ und „.“ bestehen.
- Entity-Referenzen wie in SGML: `&Entity-Name;` [WT:III [SGML](#)]
- **Kommentare:** `<!-- Dies ist ein Kommentar -->`
- Die CDATA-Deklaration ermöglicht die literale Verwendung aller Zeichen:

```
<![CDATA [  
  <svg xmlns="https://www.w3.org/2000/svg"  
    width="12cm" height="10cm">  
    <ellipse rx="110" ry="130" />  
    ...  
  ]]>
```

- **Verarbeitungsanweisungen** werden mit `<? und ?>` eingeschlossen:

```
<?php  
  mysql_connect("database.unc.edu", "clerk", "password");  
  ...  
?>
```

Bemerkungen:

- ❑ Ideographische Zeichen, auch Bildzeichen genannt, sind Zeichen, die eine unmittelbare Interpretation besitzen. Sie können sprachunabhängig als auch sprachspezifisch sein. Beispiele sind mathematische Zeichen, chinesische Schriftzeichen oder Logogramme.
- ❑ Kommentare und Verarbeitungsanweisungen sind Markup, aber keine Elementinstanzen. Sie dürfen überall im Dokument – jedoch nicht *in* einem Tag stehen.
- ❑ Das Schlüsselwort `#CDATA` bezeichnet den Datentyp *Character Data*. Abschnitte diesen Datentyps werden durch den Parser nicht analysiert. Innerhalb eines CDATA-Abschnitts wird nur die Zeichenkette „]]>“ als Markup interpretiert; sie markiert das CDATA-Ende-Tag. [\[w3schools\]](#)
- ❑ Das Schlüsselwort `#PCDATA` bezeichnet den Datentyp *Parsed Character Data*. Abschnitte diesen Datentyps werden durch den Parser analysiert. Innerhalb eines PCDATA-Abschnitts müssen deshalb Zeichen, die Bestandteil der HTML-Markup-Syntax sind (<, >, etc.) maskiert werden, wenn sie nicht als Markup interpretiert werden sollen. Es sind alle Arten von Zeichen, Entity-Referenzen, CDATA-Abschnitte, Kommentare und Verarbeitungsanweisungen zugelassen, jedoch keine Elementinstanzen. Bei PCDATA handelt es sich üblicherweise um Text, der zwischen dem Anfang- und dem Ende-Tag einer Elementinstanz notiert wird. [\[w3schools\]](#)

XML-Grundlagen

Wohlgeformtheit und Validität

XML-Dokumente **müssen** wohlgeformt sein:

1. balancierte und geschachtelte (unverschränkte) Tags
2. genau ein Wurzelement
3. Attributnamen eindeutig pro Element, Wertzuweisungen müssen in Anführungszeichen stehen
4. keine Kommentare und Verarbeitungsanweisungen in Tags

(es gibt noch mehr)

XML-Grundlagen

Wohlgeformtheit und Validität

XML-Dokumente **müssen** wohlgeformt sein:

1. balancierte und geschachtelte (unverschränkte) Tags
2. genau ein Wurzelement
3. Attributnamen eindeutig pro Element, Wertzuweisungen müssen in Anführungszeichen stehen
4. keine Kommentare und Verarbeitungsanweisungen in Tags

(es gibt noch mehr)

XML-Dokumente **können** valide (gültig) sein. Das heißt,

1. das Dokument ist wohlgeformt,
2. das Dokument enthält eine DTD oder eine Referenz darauf, und
3. der gesamte Dokumenteninhalte ist konform mit der DTD.

XML-Grundlagen

XML Document Type Definition

XML-Dokument:

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE person SYSTEM "person.dtd">
<person>
  <name>
    <vorname>Alan</vorname>
    <nachname>Turing</nachname>
  </name>
  <geburtstag>23. Juni 1912</geburtstag>
  <beruf>Mathematiker</beruf>
  <beruf>Informatiker</beruf>
</person>
```

Zugehörige XML Document Type Definition:

```
<!ELEMENT person (name, geburtstag, beruf+)>
<!ELEMENT name (vorname, nachname)>
<!ELEMENT vorname (#PCDATA)>
<!ELEMENT nachname (#PCDATA)>
<!ELEMENT geburtstag (#PCDATA)>
<!ELEMENT beruf (#PCDATA)>
```

XML-Grundlagen

XML Document Type Definition

XML-Dokument:

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

<person>
  <name>
    <vorname>Alan</vorname>
    <nachname>Turing</nachname>
  </name>
  <geburtstag>23. Juni 1912</geburtstag>
  <beruf>Mathematiker</beruf>
  <beruf>Informatiker</beruf>
</person>
```

Zugehörige XML Document Type Definition:

```
<!ELEMENT person (name, geburtstag, beruf+)>
<!ELEMENT name (vorname, nachname)>
<!ELEMENT vorname (#PCDATA)>
<!ELEMENT nachname (#PCDATA)>
<!ELEMENT geburtstag (#PCDATA)>
<!ELEMENT beruf (#PCDATA)>
```

XML-Grundlagen

XML Document Type Definition

XML-Dokument:

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

<person>
  <name>
    <vorname>Alan</vorname>
    <nachname>Turing</nachname>
  </name>
  <geburtstag>23. Juni 1912</geburtstag>
  <beruf>Mathematiker</beruf>
  <beruf>Informatiker</beruf>
</person>
```

Zugehörige XML Document Type Definition:

```
<!ELEMENT person (name, geburtstag, beruf+)>
<!ELEMENT name (vorname, nachname)>
<!ELEMENT vorname (#PCDATA)>
<!ELEMENT nachname (#PCDATA)>
<!ELEMENT geburtstag (#PCDATA)>
<!ELEMENT beruf (#PCDATA)>
```

XML-Grundlagen

XML Document Type Definition

XML-Dokument:

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

<person>
  <name>
    <vorname>Alan</vorname>
    <nachname>Turing</nachname>
  </name>
  <geburtstag>23. Juni 1912</geburtstag>
  <beruf>Mathematiker</beruf>
  <beruf>Informatiker</beruf>
</person>
```

Zugehörige XML Document Type Definition:

```
<!ELEMENT person (name, geburtstag, beruf+)>
<!ELEMENT name (vorname, nachname)>
<!ELEMENT vorname (#PCDATA)>
<!ELEMENT nachname (#PCDATA)>
<!ELEMENT geburtstag (#PCDATA)>
<!ELEMENT beruf (#PCDATA)>
```

XML-Grundlagen

XML Document Type Definition (Fortsetzung) [WT:III SGML]

Die DTD definiert:

1. Art und Aufbau von Elementtypen für eine Klasse von Dokumenten
= die Inhaltsmodelle der Elementtypen
2. die in Elementinstanzen verwendbaren Attribute und ihre Datentypen
3. verschiedene Arten von Textkonstanten, sogenannte *Entities*

Closed-World-Semantik: Was in der DTD nicht deklariert ist, ist verboten.

XML-Grundlagen

XML Document Type Definition (Fortsetzung) [WT:III SGML]

Die DTD definiert:

1. Art und Aufbau von Elementtypen für eine Klasse von Dokumenten
= die Inhaltsmodelle der Elementtypen
2. die in Elementinstanzen verwendbaren Attribute und ihre Datentypen
3. verschiedene Arten von Textkonstanten, sogenannte *Entities*

Closed-World-Semantik: Was in der DTD nicht deklariert ist, ist verboten.

Syntax der Sätze in der DTD (Ausschnitt):

1. `<!ELEMENT Elementname Inhaltsmodell >`
2. `<!ATTLIST Elementname { Attributname Attributtyp Default-Wert }* >`
3. `<!ENTITY Entity-Name Zeichenkette >`

XML-Grundlagen

XML Document Type Definition (Fortsetzung)

Beispiel einer Elementtypdefinition:

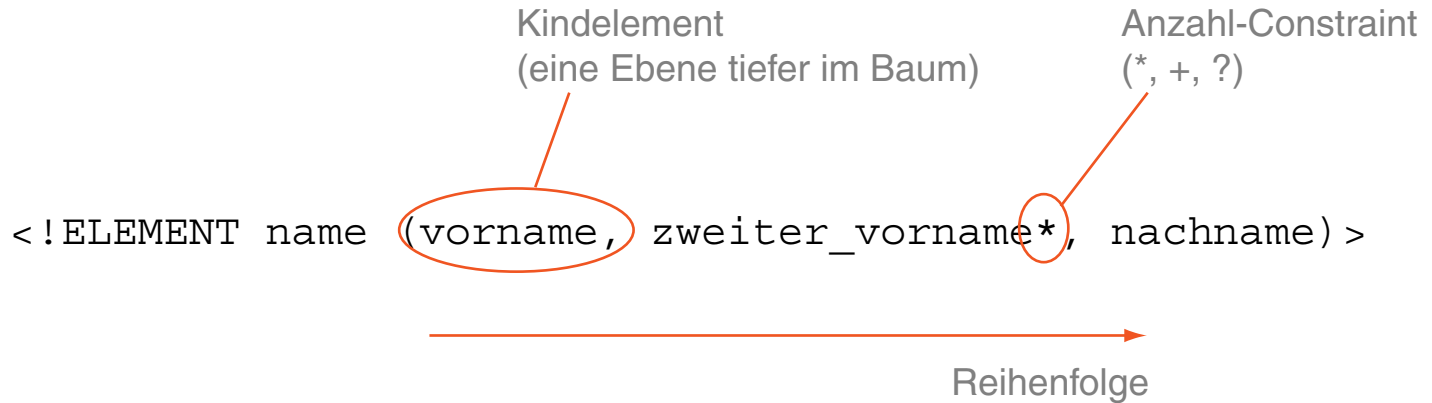
Elementname Inhaltsmodell

<!ELEMENT name (vorname, zweiter_vorname*, nachname)>

XML-Grundlagen

XML Document Type Definition (Fortsetzung)

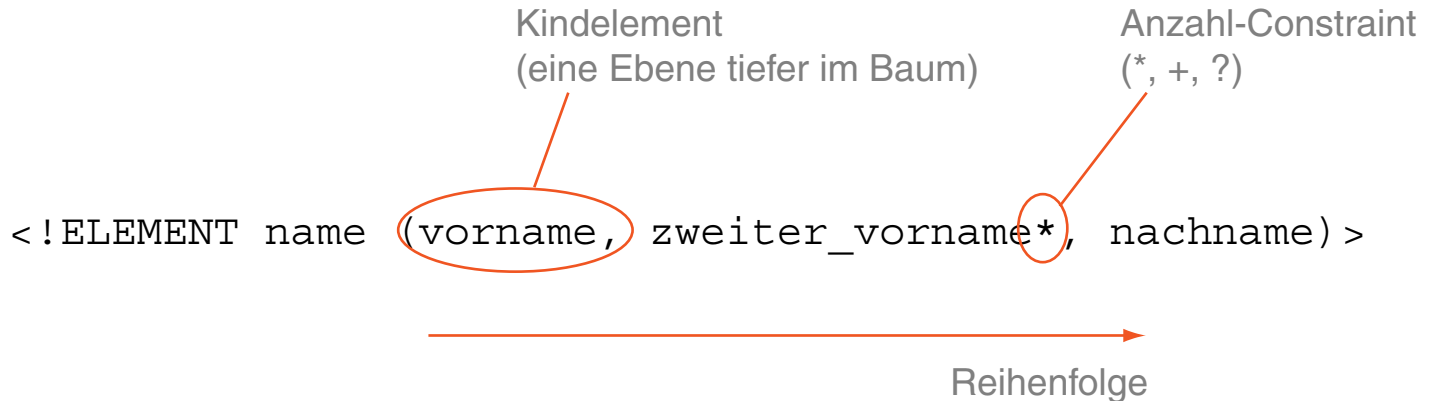
Beispiel einer Elementtypdefinition:



XML-Grundlagen

XML Document Type Definition (Fortsetzung)

Beispiel einer Elementtypdefinition:



gültige Elementinstanz:

```
<name>  
  <vorname>Alan</vorname>  
  <nachname>Turing</nachname>  
</name>
```

ungültige Elementinstanzen:

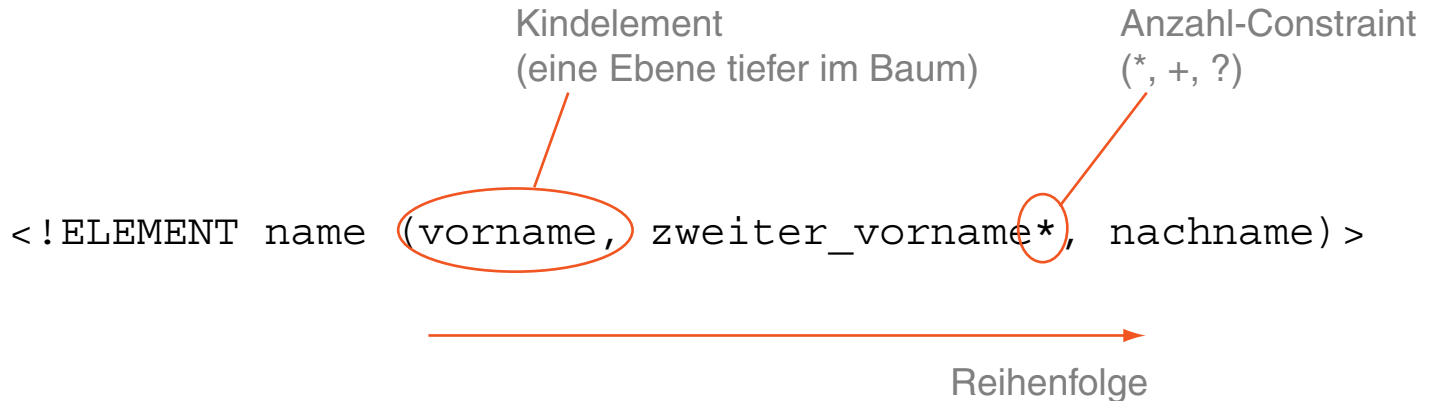
```
<name>  
  <nachname>Turing</nachname>  
  <vorname>Alan</vorname>  
</name>
```

```
<name>  
  <nachname>Turing</nachname>  
</name>
```

XML-Grundlagen

XML Document Type Definition (Fortsetzung)

Beispiel einer Elementtypdefinition:



gültige Elementinstanz:

```
<name>
  <vorname>Alan</vorname>
  <nachname>Turing</nachname>
</name>
```

ungültige Elementinstanzen:

```
<name>
  <nachname>Turing</nachname>
  <vorname>Alan</vorname>
</name>
```

```
<name>
  <nachname>Turing</nachname>
</name>
```

XML-Grundlagen

XML Document Type Definition (Fortsetzung)

Deklaration von Alternativen in einer Elementtypdefinition:

```
<!ELEMENT ziffer (null | eins | zwei | drei | vier | fünf)>
```

Klammerung und Kombination der syntaktischen Einheiten:

```
<!ELEMENT kreis (zentrum, (radius | durchmesser))>
```

XML-Grundlagen

XML Document Type Definition (Fortsetzung)

Deklaration von Alternativen in einer Elementtypdefinition:

```
<!ELEMENT ziffer (null | eins | zwei | drei | vier | fünf)>
```

Klammerung und Kombination der syntaktischen Einheiten:

```
<!ELEMENT kreis (zentrum, (radius | durchmesser))>
```

Wichtige Inhaltsmodelle für Elementtypen:

Inhaltsmodell	Typischer Aufbau
einfacher Inhalt	<code><!ELEMENT <i>Elementname</i> (#PCDATA) ></code>
explizite Kindelemente	<code><!ELEMENT <i>Elementname</i> (<i>Elementname</i>, . . . , <i>Elementname</i>) ></code>
gemischter Inhalt	<code><!ELEMENT <i>Elementname</i> (#PCDATA <i>Elementname</i>) * ></code>
beliebiger Inhalt	<code><!ELEMENT <i>Elementname</i> ANY ></code>
leerer Inhalt	<code><!ELEMENT <i>Elementname</i> EMPTY ></code>

XML-Grundlagen

XML Document Type Definition (Fortsetzung)

Deklaration von Alternativen in einer Elementtypdefinition:

```
<!ELEMENT ziffer (null | eins | zwei | drei | vier | fünf)>
```

Klammerung und Kombination der syntaktischen Einheiten:

```
<!ELEMENT kreis (zentrum, (radius | durchmesser))>
```

Wichtige Inhaltsmodelle für Elementtypen:

Inhaltsmodell	Typischer Aufbau
einfacher Inhalt	<code><!ELEMENT <i>Elementname</i> (#PCDATA) ></code>
explizite Kindelemente	<code><!ELEMENT <i>Elementname</i> (<i>Elementname</i>, . . . , <i>Elementname</i>) ></code>
gemischter Inhalt	<code><!ELEMENT <i>Elementname</i> (#PCDATA <i>Elementname</i>) * ></code>
beliebiger Inhalt	<code><!ELEMENT <i>Elementname</i> ANY ></code>
leerer Inhalt	<code><!ELEMENT <i>Elementname</i> EMPTY ></code>

XML-Grundlagen

XML Document Type Definition (Fortsetzung)

Deklaration der erlaubten Attribute für einen Elementtyp:

Elementname	Attributname	Attributtyp	Default-Wert
<!ATTLIST	bild	quelle	CDATA #REQUIRED
	breite	CDATA	"100%"
	hoehe	CDATA	#FIXED "1m"
	info	CDATA	#IMPLIED >

XML-Grundlagen

XML Document Type Definition (Fortsetzung)

Deklaration der erlaubten Attribute für einen Elementtyp:

Elementname	Attributname	Attributtyp	Default-Wert
<!ATTLIST	bild	quelle	CDATA #REQUIRED
	breite	CDATA	"100%"
	hoehe	CDATA	#FIXED "1m"
	info	CDATA	#IMPLIED >

Wichtige Attributtypen:

CDATA (Zeichenkette), ID (eindeutiger Name), IDREF (Verweis auf ein ID-Attribut),
NMTOKEN (Symbol), ENTITY

Default-Wert für Attribut

Semantik

#IMPLIED

das Attribut ist optional

#REQUIRED

das Attribut ist obligatorisch

#FIXED *Wert*

der Attributwert ist fest und kann angegeben sein

Wert

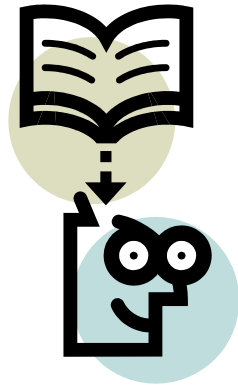
Default, falls kein Attributwert im Dokument angegeben ist

Bemerkungen:

- ❑ Attributwerte dürfen keine Elemente sein oder enthalten.
- ❑ In XML gibt es 10 Attributtypen.

XML-Grundlagen

Quiz [\[w3schools\]](#)



XML-Grundlagen

Internationalisierung



XML-Grundlagen

Internationalisierung (Fortsetzung)

Unterscheidung folgender Konzepte:

1. **abstraktes Zeichen** (*Abstract character, Character*) [[unicode](#)]
Element zur Konstruktion oder Repräsentation textueller Daten.
2. Zeichendarstellung (*Glyph, Glyph image, Character shape*) [[unicode](#) [1](#), [2](#)]
3. Zeichenvorrat, Zeichensatz (*Character repertoire*) [[unicode](#)]
Menge von abstrakten Zeichen (1) zur Repräsentation von Text.

XML-Grundlagen

Internationalisierung (Fortsetzung)

Unterscheidung folgender Konzepte:

1. **abstraktes Zeichen** (*Abstract character, Character*) [[unicode](#)]
Element zur Konstruktion oder Repräsentation textueller Daten.
2. Zeichendarstellung (*Glyph, Glyph image, Character shape*) [[unicode](#) [1](#), [2](#)]
3. Zeichenvorrat, Zeichensatz (*Character repertoire*) [[unicode](#)]
Menge von abstrakten Zeichen (1) zur Repräsentation von Text.
4. Code-Raum (*Codespace*) [[unicode](#)]
Menge von Zahlen, die abstrakten Zeichen zugeordnet werden können. Zahlen des Code-Raums heißen Zeichen-Codes (*Code points, Character codes, Character numbers*).
5. **Code-Tabelle**, codierter Zeichensatz (*Coded character set, Charset*) [[unicode](#)]
Abbildung eines Zeichenvorrats bzw. Zeichensatzes (3) auf Code-Points (4).
6. **Codierungsformat** (*Encoding form, Encoding scheme, Encoding*) [[unicode](#) [1](#), [2](#)]
Format der Byte-Repräsentation eines Code-Points (4).

Bemerkungen:

- ❑ Die Namen der meisten Zeichen, die irgendwo auf der Welt benutzt werden, findet man unter [unicode](#).
- ❑ Unterschied zwischen abstrakten Zeichen und Zeichendarstellung:

“The difference between identifying a code point and rendering it on screen or paper is crucial to understanding. The character identified by a code point is an abstract entity, such as «LATIN CHARACTER CAPITAL A» or «BENGALI DIGIT 5». The mark made on screen or paper – called a glyph – is a visual representation of the character.

The Unicode Standard does not define glyph images. The standard defines how characters are interpreted, not how glyphs are rendered. The software or hardware-rendering engine of a computer is responsible for the appearance of the characters on the screen. The Unicode Standard does not specify the size, shape, nor style of on-screen characters.” [\[unicode\]](#)

“In Unicode, the letter A is a platonic ideal. It’s just floating in heaven.”
[\[www.joelonsoftware.com\]](http://www.joelonsoftware.com)
- ❑ Das Konzept des Zeichenvorrats (3) ist an das Konzept einer Sprache geknüpft und folglich unabhängig von einem Computer. Ein Beispiel für einen Zeichenvorrat (3) ist Latein 1 (*Latin 1*): es ist die Menge der Zeichen, die in ISO/IEC 8859-1 aufgeführt sind und die zum Schreiben der Sprachen Westeuropas benötigt werden.

XML-Grundlagen

Internationalisierung (Fortsetzung)

Ein Zeichenvorrat (3) für eine bestimmte Sprache heißt Alphabet oder Schrift.

Beispiele: lateinische Schrift, chinesische Schrift.

Anspruch von Unicode:

“The Unicode Standard provides the capacity to encode all of the characters used for the written languages of the world. To keep character coding simple and efficient, the Unicode Standard assigns each character a unique numeric value and name.” [\[unicode\]](#)

Sprache	(4) Code-Raum	(5) Code-Tabelle bzw. Charset	(6) Codierungsformat bzw. Encoding	Code- Einheit	Code- Länge
Englisch	0-7F	US-ASCII	<i>kanonisch</i>	1 Byte	1 Byte
Westeuropa	0-FF	ISO 8859-1	<i>kanonisch</i>	1 Byte	1 Byte
Chinesisch	0-FFFF	Big 5	<i>kanonisch</i>	2 Byte	2 Byte
alle	0-10FFFF	Unicode 3.0	UTF-8 UTF-16	1 Byte 2 Byte	1-4 Byte 2-4 Byte

XML-Grundlagen

Internationalisierung (Fortsetzung)

Ein Zeichenvorrat (3) für eine bestimmte Sprache heißt Alphabet oder Schrift.

Beispiele: lateinische Schrift, chinesische Schrift.

Anspruch von Unicode:

“The Unicode Standard provides the capacity to encode all of the characters used for the written languages of the world. To keep character coding simple and efficient, the Unicode Standard assigns each character a unique numeric value and name.” [\[unicode\]](#)

Sprache	(4) Code-Raum	(5) Code-Tabelle bzw. Charset	(6) Codierungsformat bzw. Encoding	Code- Einheit	Code- Länge
Englisch	0-7F	US-ASCII	<i>kanonisch</i>	1 Byte	1 Byte
Westeuropa	0-FF	ISO 8859-1	<i>kanonisch</i>	1 Byte	1 Byte
Chinesisch	0-FFFF	Big 5	<i>kanonisch</i>	2 Byte	2 Byte
alle	0-10FFFF	Unicode 3.0	UTF-8 UTF-16	1 Byte 2 Byte	1-4 Byte 2-4 Byte

Bemerkungen:

- ❑ Viele Code-Tabellen liegen in nur *einem* Codierungsformat / Encoding (6) vor, das in der Tabelle hier als „kanonisch“ bezeichnet ist. In der Praxis wird oft – aber fälschlicherweise – der Name der Code-Tabelle / Charset (5) für das Codierungsformat / Encoding (6) verwendet.
- ❑ Um die Verwirrung komplett zu machen: Tatsächlich spezifiziert man in der Meta-Information von HTML-Dokumenten das Attribut `charset` und in der Meta-Information von XML-Dokumenten das Attribut `encoding`. Als Werte hierfür können die Namen der Code-Tabelle (ISO 8859-1, US-ASCII, etc.) oder des Codierungsformats (UTF-8, UTF-16, etc.) auftreten.
- ❑ UTF-8 und UTF-16 verwenden variable Code-Längen. UTF steht für Unicode Transformation Format. [\[unicode\]](#)
- ❑ Damit ein XML-Parser ein Dokument lesen kann, muss er die verwendete Code-Tabelle / Charset (5) und dessen Codierungsformat / Encoding (6) kennen.

XML-Grundlagen

Internationalisierung (Fortsetzung)

Die Code-Tabellen US-ASCII und GER-ASCII: druckbare Zeichen erhalten die Zeichen-Codes von 32 bis 126, Steuerzeichen von 0 bis 31.

US-ASCII.

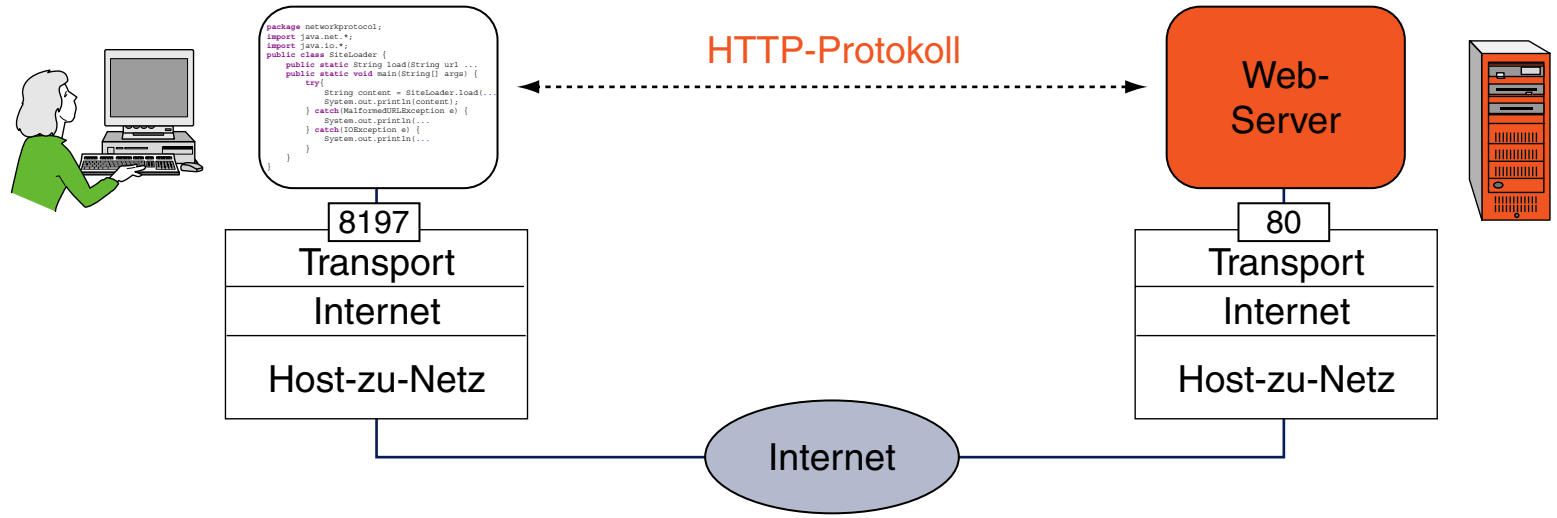
20	21	22	23	24	25	26	27	28	29	2A	2B	2C	2D	2E	2F
	!	"	#	\$	%	&	'	()	*	+	,	-	.	/
30	31	32	33	34	35	36	37	38	39	3A	3B	3C	3D	3E	3F
0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F
@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
50	51	52	53	54	55	56	57	58	59	5A	5B	5C	5D	5E	5F
P	Q	R	S	T	U	V	W	X	Y	Z	[\]	^	_
60	61	62	63	64	65	66	67	68	69	6A	6B	6C	6D	6E	6F
'	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
70	71	72	73	74	75	76	77	78	79	7A	7B	7C	7D	7E	7F
p	q	r	s	t	u	v	w	x	y	z	{		}	~	

GER-ASCII.

20	21	22	23	24	25	26	27	28	29	2A	2B	2C	2D	2E	2F
	!	"	#	\$	%	&	'	()	*	+	,	-	.	/
30	31	32	33	34	35	36	37	38	39	3A	3B	3C	3D	3E	3F
0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F
§	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
50	51	52	53	54	55	56	57	58	59	5A	5B	5C	5D	5E	5F
P	Q	R	S	T	U	V	W	X	Y	Z	Ä	Ö	Ü	^	_
60	61	62	63	64	65	66	67	68	69	6A	6B	6C	6D	6E	6F
'	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
70	71	72	73	74	75	76	77	78	79	7A	7B	7C	7D	7E	7F
p	q	r	s	t	u	v	w	x	y	z	ä	ö	ü	ß	

XML-Grundlagen

Internationalisierung: HTTP-Kommunikation mit Java [WT:II SiteLoader]



XML-Grundlagen

Internationalisierung: HTTP-Kommunikation mit Java [WT:II SiteLoader]

```
public static String load(String urlString) throws IOException {
    URL url = new URL(urlString);
    HttpURLConnection con = (HttpURLConnection) url.openConnection();

    String contentType = con.getContentType();
    System.out.println("Content-Type: " + contentType);
    String encoding = extractCharset(contentType, "utf-8");
    System.out.println("Charset encoding: " + encoding);

    InputStream in = con.getInputStream();
    BufferedReader br = new BufferedReader(new InputStreamReader(in, encoding));
    String curline;
    StringBuilder content = new StringBuilder();

    while ((curline = br.readLine()) != null) {
        content.append(curline + '\n');
    }

    br.close();
    con.disconnect();
    return content.toString();
}
```

XML-Grundlagen

Internationalisierung: HTTP-Kommunikation mit Java [WT:II SiteLoader]

```
public static String load(String urlString) throws IOException {
    URL url = new URL(urlString);
    HttpURLConnection con = (HttpURLConnection) url.openConnection();

    String contentType = con.getContentType();
    System.out.println("Content-Type: " + contentType);
    String encoding = extractCharset(contentType, "utf-8");
    System.out.println("Charset encoding: " + encoding);

    InputStream in = con.getInputStream();
    BufferedReader br = new BufferedReader(new InputStreamReader(in, encoding));
    String curline;
    StringBuilder content = new StringBuilder();

    while ((curline = br.readLine()) != null) {
        content.append(curline + '\n');
    }

    br.close();
    con.disconnect();
    return content.toString();
}
```

XML-Grundlagen

Internationalisierung: HTTP-Kommunikation mit Java [WT:II SiteLoader]

```
public static String extractCharset
(String contentType, String defaultCharset) {
    // Extracts the charset value from the Content-Type header.
    // If no charset value is specified, the given default is returned.
    // Example. Content-Type: "text/html; charset=UTF-8"
    // Background: The *charset* value corresponds to the *encoding* if
    // for the charset only a single (canonical) encoding exists.

    String charset = defaultCharset;
    for (String param : contentType.replace(" ", "").split(";")) {
        if (param.toLowerCase().startsWith("charset=")) {
            charset = param.split("=")[1];
            break;
        }
    }
    return charset;
}
```

XML-Grundlagen

Internationalisierung: HTTP-Kommunikation mit Java [WT:II SiteLoader]

```
package documentlanguages.webcrawler;
import java.net.*;
import java.io.*;

public class SiteLoader2 {

    public static String load(String urlString) ...
    public static String extractCharset(String contentType, ...

    public static void main(String[] args) {
        try {
            String content = SiteLoader2.load("http://www.heise.de");
            System.out.println(content);
        }
        catch (MalformedURLException e) {
            System.out.println("MalformedURLException:" + e.getMessage());
        }
        catch (IOException e) {
            System.out.println("IOException:" + e.getMessage());
        }
    }
}
```


XML-Grundlagen

Internationalisierung: HTTP-Kommunikation mit Java [WT:II Response-Message]

```
[stein@webis bin]$ java documentlanguages.webcrawler.SiteLoader2 | less
```

```
Content-Type: text/html; charset=utf-8
```

```
Charset encoding: utf-8
```

```
<!DOCTYPE html>
```

```
<html lang="de">
```

```
<head>
```

```
<title>heise online - IT-News, Nachrichten und Hintergründe</title>
```

```
<meta name="description" content="News und Foren zu Computer, IT, Wissenschaft, ...
```

```
<meta name="keywords" content="heise online, c't, iX, Technology Review, ...
```

```
<meta name="publisher" content="Heise Zeitschriften Verlag" />
```

```
<meta name="viewport" content="width=1175" />
```

```
<link rel="home" type="text/html" title="Startseite" href="/" />
```

```
<link rel="copyright" title="Copyright" href="/impressum.html" />
```

```
<meta http-equiv="PICS-Label" content="(PICS-1.1 &quot;http://www.rsac.org/...
```

```
<meta name="generator" content="InterRed V14.0, http://www.interred.de/, ...
```

```
<script type="text/javascript" src="/js/jquery/jquery-1.7.1.min.js"></script>
```

```
<script type="text/javascript" src="/js/ho/link_inline_images.min.js"></script>
```

```
<script type="text/javascript" src="/js/ho/bilderstrecke-1.1.min.js"></script>
```

```
...
```

```
</head>
```

```
<body>
```

```
...
```

Bemerkungen:

- ❑ Zur Bezeichnung des Encodings ist der Entity-Header „Content-Encoding“ vorgesehen, aber dieser wird vom Web-Server meist nicht gesetzt: die Information zum Encoding ist oft hinter dem Mime-Type im Entity-Header „Content-Type“ angegeben. Deshalb wird in dem Java-Beispiel auch der Entity-Header „Content-Type“ abgefragt, der eigentlich den Mime-Type spezifiziert.
- ❑ Beachte weiter, dass in dem Beispiel auch nicht das Encoding-Attribut, sondern das Charset-Attribut abgefragt wird. Das macht dann keinen Unterschied, wenn das Encoding kanonisch, also eindeutig ist, wie z.B. für den Charset „ISO 8859-1“.

XML-Grundlagen

Namensräume

Namensräume (*Namespaces*) in XML haben folgende Aufgaben:

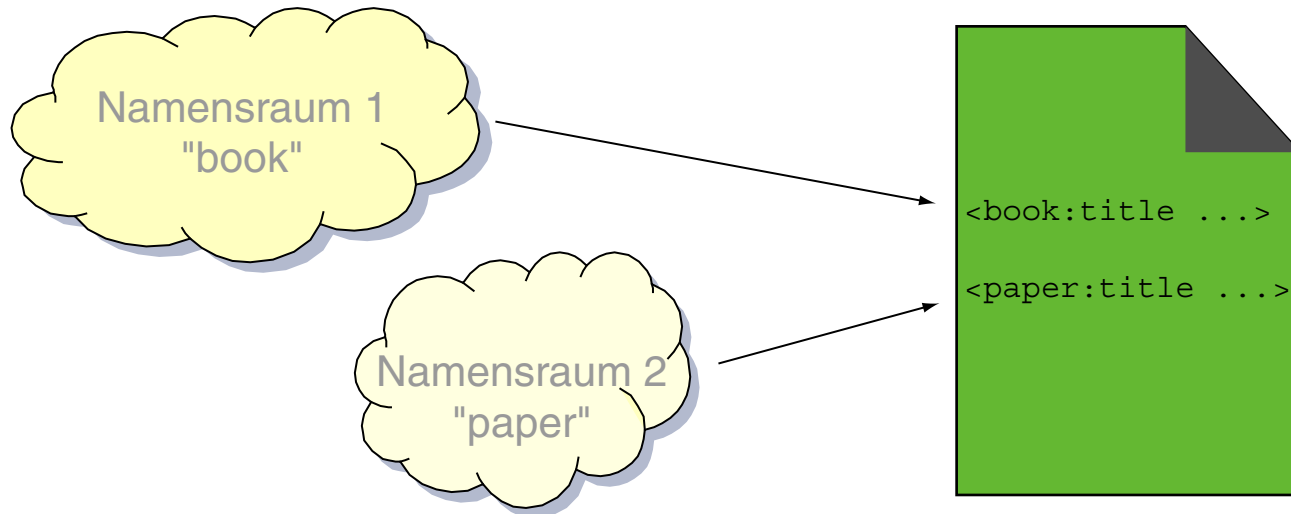
- ❑ Unterscheidung von Elementen und Attributen, die den gleichen Namen haben, aber in verschiedenen Zusammenhängen verwendet werden.
Stichwort: Vermeidung von Namenskonflikten
- ❑ Logische Gruppierung aller Elemente und Attribute einer Anwendung.

XML-Grundlagen

Namensräume

Namensräume (*Namespaces*) in XML haben folgende Aufgaben:

- ❑ Unterscheidung von Elementen und Attributen, die den gleichen Namen haben, aber in verschiedenen Zusammenhängen verwendet werden.
Stichwort: Vermeidung von Namenskonflikten
- ❑ Logische Gruppierung aller Elemente und Attribute einer Anwendung.



Namensräume sind Bezeichner – sie definieren keine Umgebung (*Scope*).

XML-Grundlagen

Namensräume (Fortsetzung)

Namensräume (*Namespaces*) in XML haben folgende Aufgaben:

- Unterscheidung von Elementen und Attributen, die den gleichen Namen haben, aber in verschiedenen Zusammenhängen verwendet werden.
Stichwort: Vermeidung von Namenskonflikten
- Logische Gruppierung aller Elemente und Attribute einer Anwendung.

Verwendung von Namensräumen in zwei Schritten:

1. **Deklaration des Namensraums** durch Bindung einer URI an einen Präfix mit `xmlns`.

```
xmlns:book="https://www.books.com"
```

2. **Qualifizierung des Vokabulars**

book :title	qualifizierender Name
book :	Präfix
title	lokaler Teil

Jede URI ist als Namensraum verwendbar.

XML-Grundlagen

Namensräume (Fortsetzung)

Gültigkeit der Namensraumdeklaration:

- Innerhalb des Elements (einschließlich), in dem die URI gebunden wird.

```
<book: Buch xmlns:book="https://www.books.com">  
  <book: Titel>Heuristics</book: Titel>  
  <Autor>Judea Pearl</Autor>  
</book: Buch>
```

- Überschreiben der Deklaration innerhalb der Elementhierarchie möglich.

XML-Grundlagen

Namensräume (Fortsetzung)

Gültigkeit der Namensraumdeklaration:

- Innerhalb des Elements (einschließlich), in dem die URI gebunden wird.

```
<book: Buch xmlns:book="https://www.books.com">  
  <book: Titel>Heuristics</book: Titel>  
  <Autor>Judea Pearl</Autor>  
</book: Buch>
```

- Überschreiben der Deklaration innerhalb der Elementhierarchie möglich.

Einrichtung eines Default-Namensraums:

- Bindung einer URI an den leeren Präfix.

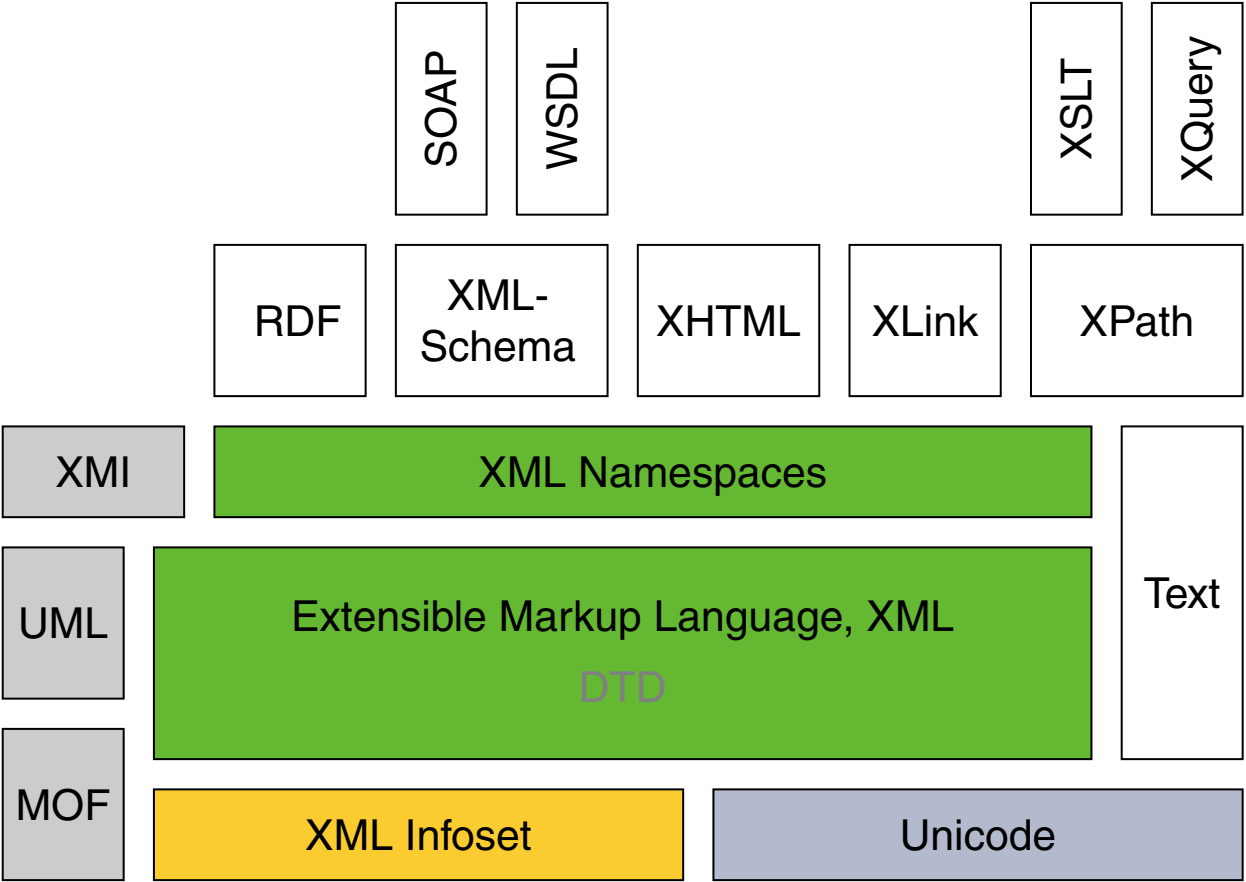
```
<Buch xmlns="https://www.books.com">  
  <Titel>Heuristics</Titel>  
  <Autor>Judea Pearl</Autor>  
</Buch>
```

Bemerkungen:

- ❑ Man bezeichnet Elementnamen, Attributnamen etc. die zu einem Namensraum gehören, als „qualifiziert“. Diese Qualifizierung kann explizit über ein Präfix, oder implizit über die Deklaration eines Default-Namensraums geschehen.
- ❑ Nicht qualifizierte Namen gehören zu dem anonymen bzw. universellen Namensraum. Im ersten Beispiel gehört `<Author>` zum anonymen Namensraum.
- ❑ Die Konzepte Default-Namensraum (= implizite Qualifizierung ohne Präfix) und anonymer Namensraum (= keine Qualifizierung) sind sorgfältig zu unterscheiden.
- ❑ *Attribute* ohne Präfix gehören nicht zum Default-Namensraum. D.h., auch wenn ein Element zu einem bestimmten (Default-)Namensraum gehört, so gehören seine *Attribute* ohne Präfix zu dem anonymen Namensraum.
- ❑ Eine Namensraumdeklaration mit Präfix besitzt Präferenz gegenüber dem Default-Namensraum.
- ❑ Durch Bindung einer leeren URI an einen Präfix wird eine bestehende Namensraumdeklaration aufgehoben. So entsteht eine Situation identisch zu einem Dokument ohne Namensraum; d.h., die Elemente gehören zum anonymen Namensraum.
- ❑ Eine Elementinstanz kann mehrere Namensraumdeklarationen aufnehmen. In der Praxis hat es sich aus Übersichtlichkeitsgründen durchgesetzt, alle in einem XML-Dokument verwendeten Namensräume zu Beginn des Dokuments im Wurzelement zu deklarieren.

XML-Grundlagen

XML Information Set [W3C] [Jeckle 2004]



XML-Grundlagen

XML Information Set (Fortsetzung)

Das XML Information Set (Infoset) definiert das XML-Dokumenten unterliegende Datenmodell. Es dient zur Beantwortung der Frage:

Welche Informationen sind in einem XML-Dokument codiert?

Ein Parser, der ein XML-Dokument analysiert, orientiert sich an der Definition des XML Information Set und stellt die entsprechenden aus dem XML-Dokument ableitbaren Informationen in einer Datenstruktur bereit.

XML-Grundlagen

XML Information Set (Fortsetzung)

Das XML Information Set (Infoset) definiert das XML-Dokumenten unterliegende Datenmodell. Es dient zur Beantwortung der Frage:

Welche Informationen sind in einem XML-Dokument codiert?

Ein Parser, der ein XML-Dokument analysiert, orientiert sich an der Definition des XML Information Set und stellt die entsprechenden aus dem XML-Dokument ableitbaren Informationen in einer Datenstruktur bereit. Beispiele:

- ❑ wie ein Element heißt
- ❑ zu welchem Namensraum ein Element gehört
- ❑ Reihenfolge der Elementinstanzen
- ❑ Code-Tabelle

Beispiele für nicht ableitbare Information:

- ❑ Größe des Leerraums zwischen Attributen
- ❑ Reihenfolge der Attribute eines Elementtyps

Bemerkungen:

- ❑ Das XML Information Set ist keine Sprache wie andere W3C-Spezifikationen, sondern ein Datenmodell. Die XML-Syntax ist eine Serialisierung dieses Datenmodells.
- ❑ Das W3C hat mittlerweile drei Datenmodelle für XML-Dokumente definiert: XML Information Set, XPath, Document Object Model (DOM). Das XPath-Datenmodell basiert auf einer Baumstruktur, die bei der Abfrage eines XML-Dokuments durchlaufen wird und ist dem XML Information Set ähnlich; DOM ist der Vorläufer beider Datenmodelle. DOM und das XPath-Datenmodell können als Interpretationen des XML Information Sets betrachtet werden. [[MSDN](#)]

XML-Grundlagen

XML Information Set (Fortsetzung)

Das XML Information Set eines XML-Dokuments wird als Baum repräsentiert.

Die Elemente des Baums heißen Informationseinheiten (*Information Items*) und sind von einem der folgenden Typen:

1. Document Information Item
2. Element Information Item
3. Attribute Information Item
4. Processing Instruction Information Item
5. Unexpanded Entity Reference Information Item
6. Character Information Item
7. Comment Information Item
8. Document Type Declaration Information Item
9. Unparsed Entity Information Item
10. Notation Information Item
11. Namespace Information Item

XML-Grundlagen

XML Information Set: Beispiel

```
<?xml version="1.0"
  encoding="ISO-8859-1"
  standalone="yes">

<person>
  <name geburtstag="23-06-1912">
    <vorname>Alan</vorname>
    <nachname>Turing</nachname>
  </name>
</person>
```

XML-Grundlagen

XML Information Set: Beispiel (Fortsetzung)

Document Information Item

version="1.0"

Encoding Scheme="ISO-8859-1"

standalone="yes"

```
<?xml version="1.0"  
encoding="ISO-8859-1"  
standalone="yes">
```

```
<person>  
  <name geburtstag="23-06-1912">  
    <vorname>Alan</vorname>  
    <nachname>Turing</nachname>  
  </name>  
</person>
```

XML-Grundlagen

XML Information Set: Beispiel (Fortsetzung)

Document Information Item

version="1.0"
Encoding Scheme="ISO-8859-1"
standalone="yes"

Element Information Item

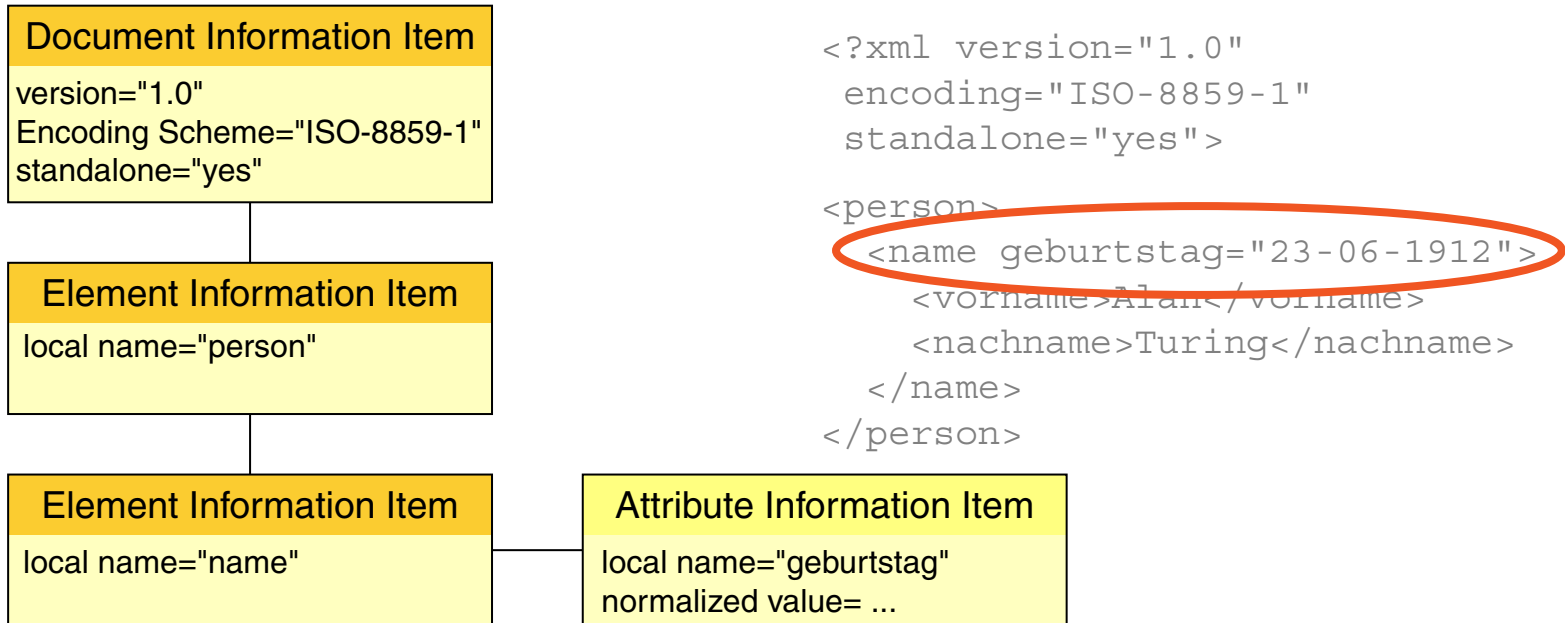
local name="person"

```
<?xml version="1.0"  
encoding="ISO-8859-1"  
standalone="yes">
```

```
<person>  
  <name geburtstag="23-06-1912">  
    <vorname>Alan</vorname>  
    <nachname>Turing</nachname>  
  </name>  
</person>
```

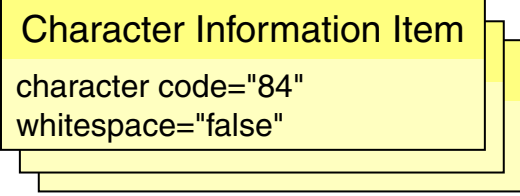
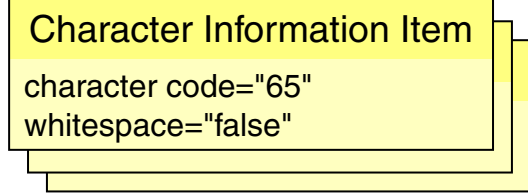
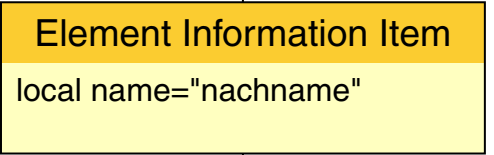
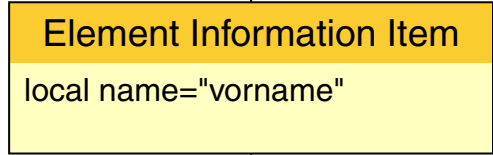
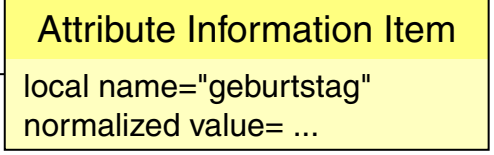
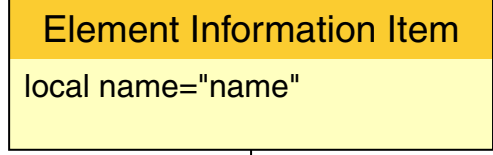
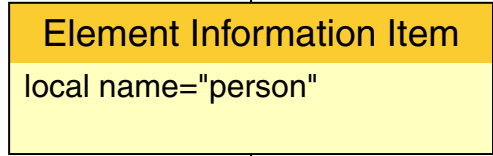
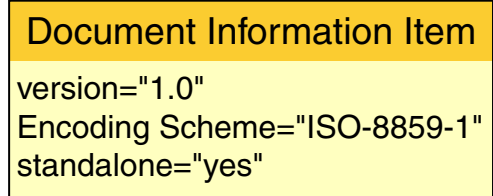

XML-Grundlagen

XML Information Set: Beispiel (Fortsetzung)



XML-Grundlagen

XML Information Set: Beispiel (Fortsetzung)



```
<?xml version="1.0"
encoding="ISO-8859-1"
standalone="yes">

<person>
  <name geburtstag="23-06-1912">
    <vorname>Alan</vorname>
    <nachname>Turing</nachname>
  </name>
</person>
```

XML-Grundlagen

Quellen zum Nachlernen und Nachschlagen im Web

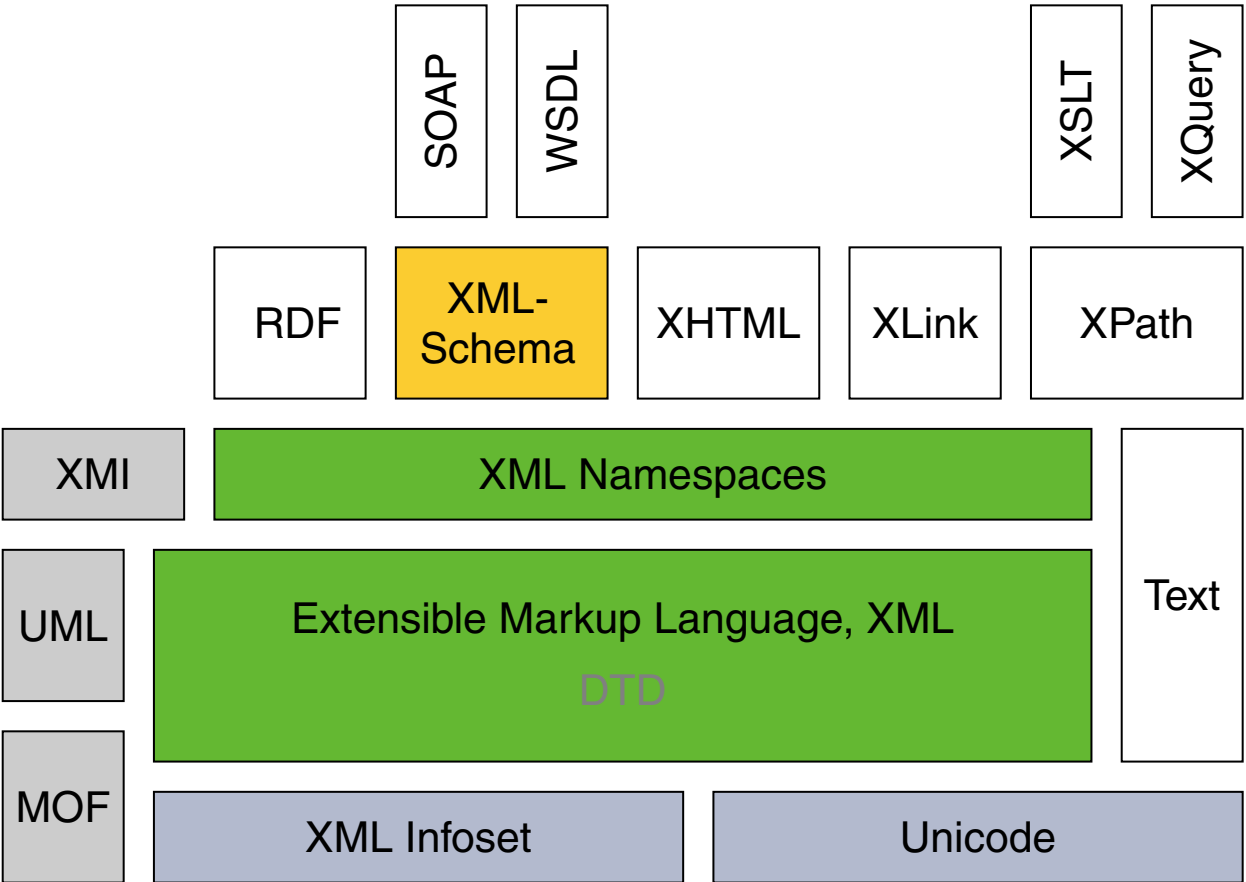
- ❑ Jeckle. *XML Vorlesung*.
www.jeckle.de
- ❑ Joel on Software. *Unicode Essay*.
www.joelonsoftware.com/articles/Unicode.htm
- ❑ Unicode. *Glossary*.
www.unicode.org/glossary
- ❑ W3C. *Character Encodings for Beginners*.
www.w3.org/International/questions/qa-what-is-encoding
- ❑ W3C. *Namespaces in XML 1.1*.
www.w3.org/TR/xml-names11
- ❑ W3C. *XML Information Set, Second Edition*.
www.w3.org/TR/xml-infoset (deutsche Übersetzung)
- ❑ W3 Schools. *XML Tutorial*.
www.w3schools.com/xml

III. Dokumentensprachen

- ❑ Auszeichnungssprachen
- ❑ HTML
- ❑ Cascading Stylesheets CSS
- ❑ XML-Grundlagen
- ❑ XML-Schema
- ❑ Die XSL-Familie
- ❑ APIs für XML-Dokumente

XML-Schema

Einordnung [Jeckle 2004]



XML-Schema

Historie: zentrale XML-Spezifikationen

- 2008 XML 1.0, Recommendation. [W3C [REC](#)]
- 2006 XML 1.1, Recommendation. [W3C [REC](#), [status](#)]

- 2004 XML-Schema Part 0: Primer, Recommendation. [W3C [REC](#)]
- 2012 XML-Schema (XSD) 1.1 Part 1: Structures. [W3C [REC](#)]
- 2012 XML-Schema (XSD) 1.1 Part 2: Datatypes. [W3C [REC](#), [status](#)]

- 2014 XSL Transformations (XSLT) 3.0. Latest Working Draft. [W3C [WD](#), [status](#)]
- 2014 XML Path Language (XPath) 3.0. Recommendation. [W3C [REC](#), [status](#)]
- 2014 XML Query Language (XQuery) 3.0. Recommendation. [W3C [REC](#), [status](#)]
- 2012 XSL Formatting Objects (XSL-FO) 2.0. Working Draft. [W3C [WD](#), [status](#)]

Bemerkungen:

- ❑ XML-Schema \equiv XML Schema Definition Language, XSD.
- ❑ XML-Schema bildet zusammen mit XML 1.1 und den Namensräumen die Basis aller weiteren W3C-XML-Sprachstandards. Die Grammatiken neu entwickelter XML-Sprachen werden nicht mehr in der DTD-Syntax formuliert.
- ❑ Der XSD-Sprachvorschlag gliedert sich in zwei Teile:
 1. XSD-Part 1 „Structures“ www.w3.org/TR/xmlschema11-1. Beschreibung von Inhaltsmodellen für Elemente, Attributstrukturen und wiederverwendbaren Strukturen. Bildet die Konzepte von DTDs nach.
 2. XSD-Part 2 „Datatypes“ www.w3.org/TR/xmlschema11-2. Beschreibung von Datentypen für XML-Schemas sowie andere XML-Spezifikationen. Es handelt sich um ein eigenständiges Typsystem, das in mehreren W3C-Arbeitsgruppen Verwendung findet.

Der XML-Schema Part 0 „Primer“ www.w3.org/TR/xmlschema-0 gibt eine gute Einführung in die Konzepte und Ziele zu XML-Schema.

- ❑ Wiederholung: Der Standardisierungsprozess der W3C ist formalisiert und spiegelt sich in den verschiedenen Levels der veröffentlichten Reports wider. [W3C [level](#)]

XML-Schema

DTD versus XML-Schema [Instanz mit DTD]

```
<!ELEMENT person (vorname, nachname, beruf+)>
<!ATTLIST person geburtsjahr NMTOKEN #IMPLIED>
<!ELEMENT vorname (#PCDATA)>
<!ELEMENT nachname (#PCDATA)>
<!ELEMENT beruf (#PCDATA)>
```


XML-Schema

DTD versus XML-Schema [Instanz mit DTD]

```
<!ELEMENT person (vorname, nachname, beruf+)>
<!ATTLIST person geburtsjahr NMTOKEN #IMPLIED>
<!ELEMENT vorname (#PCDATA)>
<!ELEMENT nachname (#PCDATA)>
<!ELEMENT beruf (#PCDATA)>
```

```
<?xml version="1.0"?>
<xs:schema ...>
  <xs:element name="person">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="vorname" type="xs:string"/>
        <xs:element name="nachname" type="xs:string"/>
        <xs:element name="beruf" type="xs:string"
          minOccurs="0" maxOccurs="3"/>
      </xs:sequence>
      <xs:attribute name="geburtsjahr" type="xs:gYear" use="optional"/>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

XML-Schema

DTD versus XML-Schema [Instanz mit DTD]

```
<!ELEMENT person (vorname, nachname, beruf+)>
<!ATTLIST person geburtsjahr NMTOKEN #IMPLIED>
<!ELEMENT vorname (#PCDATA)>
<!ELEMENT nachname (#PCDATA)>
<!ELEMENT beruf (#PCDATA)>
```

```
<?xml version="1.0"?>
<xs:schema ...>
  <xs:element name="person">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="vorname" type="xs:string"/>
        <xs:element name="nachname" type="xs:string"/>
        <xs:element name="beruf" type="xs:string"
          minOccurs="0" maxOccurs="3"/>
      </xs:sequence>
      <xs:attribute name="geburtsjahr" type="xs:gYear" use="optional"/>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

XML-Schema

DTD versus XML-Schema [Instanz mit DTD]

```
<!ELEMENT person (vorname, nachname, beruf+)>
<!ATTLIST person geburtsjahr NMTOKEN #IMPLIED>
<!ELEMENT vorname (#PCDATA)>
<!ELEMENT nachname (#PCDATA)>
<!ELEMENT beruf (#PCDATA)>
```

```
<?xml version="1.0"?>
<xs:schema ...>
  <xs:element name="person">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="vorname" type="xs:string"/>
        <xs:element name="nachname" type="xs:string"/>
        <xs:element name="beruf" type="xs:string"
          minOccurs="0" maxOccurs="3"/>
      </xs:sequence>
      <xs:attribute name="geburtsjahr" type="xs:gYear" use="optional"/>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

XML-Schema

DTD versus XML-Schema [Instanz mit DTD]

```
<!ELEMENT person (vorname, nachname, beruf+)>
<!ATTLIST person geburtsjahr NMTOKEN #IMPLIED>
<!ELEMENT vorname (#PCDATA)>
<!ELEMENT nachname (#PCDATA)>
<!ELEMENT beruf (#PCDATA)>
```

```
<?xml version="1.0"?>
<xs:schema ...>
  <xs:element name="person">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="vorname" type="xs:string"/>
        <xs:element name="nachname" type="xs:string"/>
        <xs:element name="beruf" type="xs:string"
          minOccurs="0" maxOccurs="3"/>
      </xs:sequence>
      <xs:attribute name="geburtsjahr" type="xs:gYear" use="optional"/>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Bemerkungen:

- ❑ Ein XML-Schema beinhaltet [\[W3C\]](#) :
 1. *Definitionen* von einfachen und komplexen *Datentypen*.
 2. *Deklarationen* von *Elementtypen*, die in Instanzdokumenten erlaubt sind; ein Elementtyp ist von einem bestimmten Datentyp.
- ❑ Zur Deklaration der erlaubten Elementtypen dient das `<xs:element>`-Element. Beachte, dass Instanzen des `<xs:element>`-Elements sowohl eine Typdefinition zwischen öffnendem und schließendem Tag aufnehmen können...

```
<xs:element name="person">
  <xs:complexType>
    ...
  </xs:complexType>
</xs:element>
```

...als auch leer Verwendung finden:

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

Die Syntax im ersten Fall folgt dem Entwurfsmuster zur Deklaration von Elementtypen einschließlich der Definition eines neuen (hier: anonymen, komplexen) Datentyps. Die Syntax im zweiten Fall folgt dem Entwurfsmuster zur Deklaration von Elementtypen unter Rückgriff auf einen existierenden Datentyp über das `type`-Attribut.

Bemerkungen (Fortsetzung) :

- ❑ Gewöhnungsbedürftig bei XML-Schema ist, dass die (kontextfreie) Grammatik zur Beschreibung einer Dokumentenstruktur nicht mehr in der Form von Regeln, sondern „objektorientiert“, mittels XSD-Elementinstanzen und deren Schachtelung geschieht.
- ❑ Gewöhnungsbedürftig bei XML-Schema ist auch, dass die Syntax der Metasprache (hier: Grammatik im XML-Schemadokument) gleich der Syntax der Objektsprache (hier: Elemente im XML-Instanzdokument) ist.

XML-Schema

DTD versus XML-Schema (Fortsetzung)

Grenzen von DTDs:

- ❑ nur wenige Datentypen, Typsystem nicht erweiterbar
- ❑ keine Vererbung
- ❑ Syntax nicht XML-konform
- ❑ keine Unterstützung von Namensräumen
- ❑ keine Möglichkeit zur DTD-Ergänzung
Die Definition einer DTD muss vollständig sein und sämtliche Regeln für ihre Anwendung definieren.
- ❑ keine Möglichkeit zur DTD-Modularisierung
Elementtypen sind nur innerhalb der definierenden DTD wiederverwendbar, Attribute sind an das umgebende Element gebunden.

XML-Schema

DTD versus XML-Schema (Fortsetzung)

Durch die Datentypen in XML-Schema sind Aufgaben einfacher geworden:

- ❑ Definition von Constraints für zugelassenen Inhalt
- ❑ Überprüfung der Korrektheit von Daten
- ❑ Verarbeitung von Daten aus Datenbanken
- ❑ Spezialisierung und Anpassung von Datentypen
- ❑ Definition komplexer Datentypen
- ❑ Konvertierung zwischen Daten verschiedenen Typs

Weitere Errungenschaften [\[w3schools\]](#) :

- ❑ XML-Schemata sind modular verwendbar
- ❑ XML-Schemata sind in XML-Syntax geschrieben
- ❑ XML-Schemata verwenden das XML-Namensraumkonzept

XML-Schema

DTD versus XML-Schema (Fortsetzung)

Unterscheidung von Dokumenten hinsichtlich ihres Aufbaus:

1. Erzählende (*narrative*) Dokumente.

Dokumente, die aus Abschnitten und Unterabschnitten bestehen; die gesamte Struktur ist weitgehend linear: Bücher, Artikel, etc.

2. Datensatzartige Dokumente.

Stark typisierte Dokumente; zielen auf Datenaustausch ab.

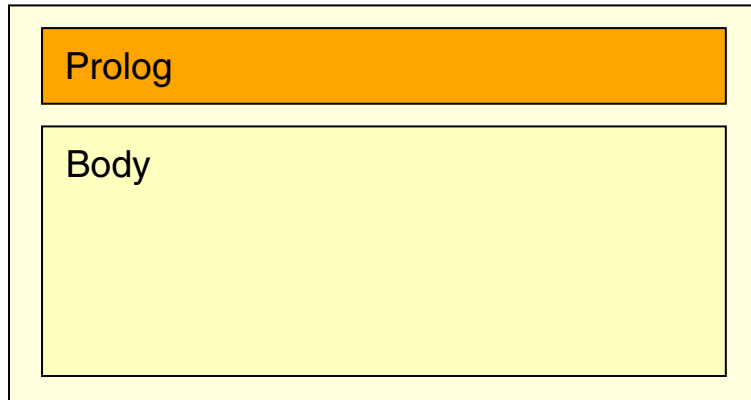
DTDs sind gut für erzählende Dokumente geeignet,
XML-Schema eignet sich gut für datensatzartige Dokumente.

XML-Schema

Aufbau eines XML-Schemas

XML-Schemata sind XML-Dokumente:

XML-
Schema



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

```
<xs:schema xmlns:xs=...>  
  <xs:element name=...>  
    ...  
  </xs:element>  
  ...  
</xs:schema>
```

- ❑ Wurzelement jedes XML-Schemas ist das Element `<xs:schema>`.
- ❑ Die als direkte (= nicht tiefer verschachtelt liegende) Kindelemente von `<xs:schema>` notierten Elemente und Attribute sind *global*.
- ❑ Die Attribute von `<xs:schema>` definieren Eigenschaften, die für alle Elemente und Attribute des Schemas gelten.
- ❑ Vergleiche hierzu die [XML-Dokumentstruktur](#).

Bemerkungen:

- ❑ Globale Elemente können als Wurzelement eines Instanzdokuments (= als Dokumentelement) verwendet werden.
- ❑ Die Dateiendung einer XML-Schemadatei ist `.xsd`.

XML-Schema

Aufbau eines XML-Schemas (Fortsetzung) [\[DTD vs Schema\]](#) [\[Instanz\]](#)

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="https://www.w3.org/2001/XMLSchema"
  targetNamespace="https://www.buw.de/webtec"
  elementFormDefault="qualified">
  <xs:element name="person">
    <xs:complexType>
      ...
    </xs:complexType>
  </xs:element>
</xs:schema>
```

XML-Schema

Aufbau eines XML-Schemas (Fortsetzung) [\[DTD vs Schema\]](#) [\[Instanz\]](#)

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="https://www.w3.org/2001/XMLSchema"
  targetNamespace="https://www.buw.de/webtec"
  elementFormDefault="qualified">
  <xs:element name="person">
    <xs:complexType>
      ...
    </xs:complexType>
  </xs:element>
</xs:schema>
```

XML-Schema

Aufbau eines XML-Schemas (Fortsetzung) [\[DTD vs Schema\]](#) [\[Instanz\]](#)

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="https://www.w3.org/2001/XMLSchema"
  targetNamespace="https://www.buw.de/webtec"
  elementFormDefault="qualified">
  <xs:element name="person">
    <xs:complexType>
      ...
    </xs:complexType>
  </xs:element>
</xs:schema>
```

XML-Schema

Aufbau eines XML-Schemas (Fortsetzung) [\[DTD vs Schema\]](#) [\[Instanz\]](#)

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="https://www.w3.org/2001/XMLSchema"
  targetNamespace="https://www.buw.de/webtec"
  elementFormDefault="qualified">
  <xs:element name="person">
    <xs:complexType>
      ...
    </xs:complexType>
  </xs:element>
</xs:schema>
```

XML-Schema

Aufbau eines XML-Schemas (Fortsetzung) [\[DTD vs Schema\]](#) [\[Instanz\]](#)

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="https://www.w3.org/2001/XMLSchema"
  targetNamespace="https://www.buw.de/webtec"
  elementFormDefault="qualified">
  <xs:element name="person">
    <xs:complexType>
      ...
    </xs:complexType>
  </xs:element>
</xs:schema>
```


XML-Schema

Aufbau eines XML-Schemas (Fortsetzung) [DTD vs Schema] [Instanz]

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="https://www.w3.org/2001/XMLSchema"
  targetNamespace="https://www.buw.de/webtec"
  elementFormDefault="qualified">
  <xs:element name="person">
    <xs:complexType>
      ...
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Zwei Arten von Namensräumen, die deklariert werden können:

1. Einen oder mehrere Namensräume, aus denen die *im Schema verwendeten* Elemente und Datentypen (= das XSD-Vokabular) gehören.
2. Einen **Zielnamensraum** (*Target Namespace*) für die global deklarierten Elemente und Attribute des Schemas (= das Autorenvokabular), der bei schemagültigen Instanzdokumenten beachtet werden muss.

Bemerkungen:

- ❑ Syntax und Semantik des XSD-Vokabulars sind durch die normativen Referenzen XML-Schema Part 0, Part 1 und Part 2 des W3C standardisiert.
Das XSD-Vokabular umfasst Namen für Elemente und Attribute, die zur **Erstellung von XML-Schemadokumenten** zur Verfügung stehen. Der zugehörige Namensraum heißt <https://www.w3.org/2001/XMLSchema>. Das übliche Präfix bei der Namensraumdeklaration ist `xsd:`, es kann aber beliebig gewählt werden.
- ❑ Alle im Schema *global* notierten Elemente und Attribute sind dem **Zielnamensraum** zugeordnet. Bei ihrer Instanziierung in einem schemagültigen Instanzdokument müssen sie zu dem entsprechenden Namensraum gehören, also entsprechend qualifiziert sein. Auch bei Referenzen innerhalb des Schemas muss diese Namensraumzugehörigkeit berücksichtigt werden.
- ❑ Das `targetNamespace`-Attribut, also die Deklaration eines Zielnamensraums, ist optional.
- ❑ Jedes XML-Schema ist eine eigenständige Datei; die Einbettung in ein Instanzdokument, also die Bildung von Internal Subsets, ist nicht möglich.

Bemerkungen (Fortsetzung):

- ❑ Durch Angabe der Attribute `elementFormDefault` und `attributeFormDefault` lässt sich die Zuordnung der *lokal definierten* Elemente und Attribute zum Zielnamensraum steuern. Standardmäßig sind diese Attribute auf `unqualified` gesetzt, und dann bezieht sich ein deklariertes Zielnamensraum nur auf die globalen Elemente und Attribute. Wird der Wert der beiden Attribute auf `qualified` gesetzt, so sind auch die lokalen Elemente und Attribute als `qualified` deklariert und müssen im Instanzdokument entsprechend qualifiziert verwendet werden.
- ❑ Um die Wartbarkeit und Lesbarkeit großer Schemata zu verbessern, kann ein Schema auf mehrere Schemadateien (mit dem gleichen Zielnamensraum) aufgeteilt werden. Die einzelnen Dateien werden dann mittels des Elements `include` aus dem XSD-Vokabular in einem Master-Dokument zusammengefasst.
- ❑ Mit dem Element `import` aus dem XSD-Vokabular lassen sich – unter Angabe eines neuen Zielnamensraums – Elemente aus „fremden“ Schemata importieren.

XML-Schema

Verknüpfung von XML-Schema und Instanzdokument [\[Schema\]](#)

```
<?xml version="1.0"?>
<person
  xmlns:xsi="https://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="https://www.buw.de/webtec person.xsd"
  xmlns="https://www.buw.de/webtec">
  <vorname>Alan</vorname>
  <nachname>Turing</nachname>
  <beruf>Mathematiker</beruf>
  <beruf>Informatiker</beruf>
</person>
```

XML-Schema

Verknüpfung von XML-Schema und Instanzdokument [\[Schema\]](#)

```
<?xml version="1.0"?>
<person
  xmlns:xsi="https://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="https://www.buw.de/webtec person.xsd"
  xmlns="https://www.buw.de/webtec">
  <vorname>Alan</vorname>
  <nachname>Turing</nachname>
  <beruf>Mathematiker</beruf>
  <beruf>Informatiker</beruf>
</person>
```

XML-Schema

Verknüpfung von XML-Schema und Instanzdokument [\[Schema\]](#)

```
<?xml version="1.0"?>
<person
  xmlns:xsi="https://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="https://www.buw.de/webtec person.xsd"
  xmlns="https://www.buw.de/webtec">
  <vorname>Alan</vorname>
  <nachname>Turing</nachname>
  <beruf>Mathematiker</beruf>
  <beruf>Informatiker</beruf>
</person>
```

XML-Schema

Verknüpfung von XML-Schema und Instanzdokument [\[Schema\]](#)

```
<?xml version="1.0"?>
<person
  xmlns:xsi="https://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="https://www.buw.de/webtec person.xsd"
  xmlns="https://www.buw.de/webtec">
  <vorname>Alan</vorname>
  <nachname>Turing</nachname>
  <beruf>Mathematiker</beruf>
  <beruf>Informatiker</beruf>
</person>
```

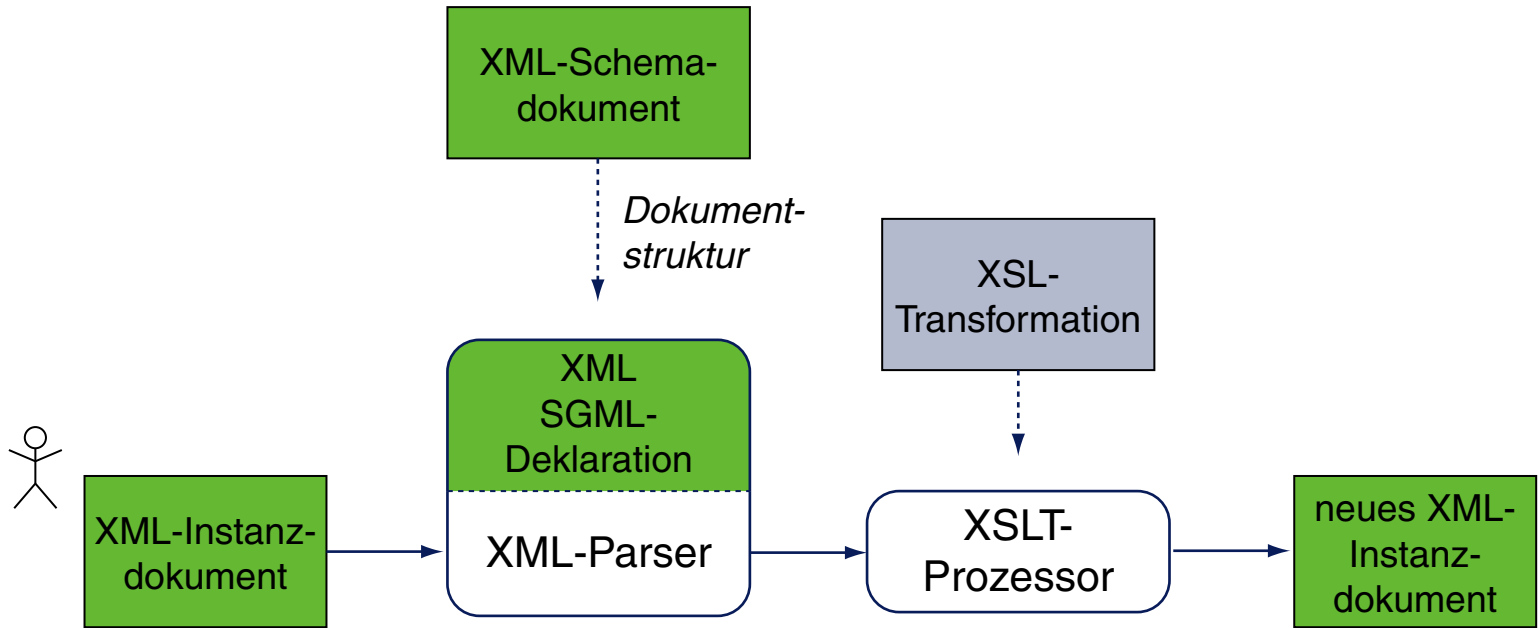
- ❑ Die Attribute `schemaLocation` und `noNamespaceSchemaLocation` dienen zur Referenz auf das Schema. Genau eines muss angegeben sein – abhängig davon, ob das Schema einen **Zielnamensraum** festlegt oder nicht.
- ❑ Der erste Parameter von `schemaLocation` ist eine URI für den **Zielnamensraum**. Der zweite Parameter ist eine relative oder absolute URL, die angibt, wo das **Schema** liegt.
- ❑ `noNamespaceSchemaLocation` hat nur den URL-Parameter für das **Schema**.

Bemerkungen:

- ❑ XSD-Part 1 „Structures“ definiert auch Vokabular für XML-Instanzdokumente; es handelt sich um insgesamt vier Attribute. Der zugehörige Namensraum heißt <https://www.w3.org/2001/XMLSchema-instance>. Das übliche Präfix bei der Namensraumdeklaration ist `xsi:`, es kann aber beliebig gewählt werden.
- ❑ Im Beispiel wird der Namensraum für Instanzdokumente deklariert und an `xsi:` gebunden, um das Attribut `schemaLocation` zu qualifizieren – das Attribut also vollständig zu benennen und so dem Parser bekannt zu machen.
- ❑ Der erste Parameter des Attributes `schemaLocation` im XML-Instanzdokument muss mit dem Wert des Attributes `targetNamespace` im XML-Schema übereinstimmen. Legt das XML-Schema *keinen* Zielnamensraum fest, so wird eine Referenz auf dieses Schema mittels `noNamespaceSchemaLocation` spezifiziert.
- ❑ Im Beispiel wird mit `xmlns="https://www.buw.de/webtec"` ein Default-Namensraum eingeführt, der dem Zielnamensraum `https://www.buw.de/webtec` entspricht. Somit gehört das `<person>`-Element im XML-Instanzdokument auch ohne Präfix zum Zielnamensraum und ist schemagültig instanziiert.

XML-Schema

Verknüpfung von XML-Schema und Instanzdokument (Fortsetzung)



Vergleiche hierzu

- ❑ die SGML Dokumentenverarbeitung,
- ❑ die HTML Dokumentenverarbeitung,
- ❑ und die XML Dokumentenverarbeitung mit DTDs.

XML-Schema

Verknüpfung von XML-Schema und Instanzdokument (Fortsetzung)

Definition 11 (gültig hinsichtlich Schema, Instanzdokument)

Ein XML-Dokument heißt gültig hinsichtlich eines Schemas (*schema valid*), wenn es über ein Schema verfügt, und konform zu diesem aufgebaut ist.

Das zu validierende Dokument wird als Instanzdokument bezeichnet.

Bemerkungen:

- ❑ Der angegebene Gültigkeitsbegriff lässt die Konformität hinsichtlich einer eventuell existierenden DTD außer Acht. So sind Dokumente denkbar, die zwar hinsichtlich einer gegebenen DTD als valide eingestuft werden, jedoch ein zugehöriges Schema verletzen – und umgekehrt.
- ❑ Aufgrund der Realisierung der Schemasprache als XML-Sprache ist jedes Schema auch ein XML-Dokument. Dadurch wird es möglich, alle erlaubten Schemata selbst durch ein Schema – ein sogenanntes *Metaschema* – zu beschreiben und zu validieren. [W3C] Somit können Schemata mit denselben Werkzeugen analysiert, verarbeitet und geprüft werden, die auch für Instanzdokumente Verwendung finden.
- ❑ Für ein Metaschema könnte wiederum ein Schema angegeben werden. Um eine unendliche Reihung zur Validierung notwendiger Schemata zu vermeiden, hat man bei der XML-Standardisierung darauf geachtet, dass die Sprache zur Beschreibung von XML-Schemata ausdrucksstark genug ist, um als Metasprache zur Beschreibung aller erlaubten Schemata zu fungieren.
- ❑ Online-Services und Werkzeuge:
 - www.freeformatter.com/xml-validator-xsd.html (Schemavalidierung)
 - www.utilities-online.info/xsdvalidation (Schemavalidierung)
 - www.altova.com/download-trial.html (allgemein)
- ❑ Java Xerces Library: `[user@pc]$ java dom.Writer -v -s Instanzdokument`

XML-Schema

Wie man ein XML-Schema entwickelt

Voraussetzungen, um ein neues XML-Schema zu entwickeln:

- ❑ Verständnis für den Zusammenhang zwischen
 - **Inhaltsmodell**: charakterisiert die Struktur von Elementinstanzen
 - **Elementtyp**: implementiert ein Inhaltsmodell, ist von einem Datentyp
 - **Datentyp** eines Elementtyps: deklariert erlaubte Daten
- ❑ Sicherer Umgang mit (Ziel-, Default-) Namensräumen

XML-Schema

Wie man ein XML-Schema entwickelt

Voraussetzungen, um ein neues XML-Schema zu entwickeln:

- Verständnis für den Zusammenhang zwischen
 - **Inhaltsmodell**: charakterisiert die Struktur von Elementinstanzen
 - **Elementtyp**: implementiert ein Inhaltsmodell, ist von einem Datentyp
 - **Datentyp** eines Elementtyps: deklariert erlaubte Daten
- Sicherer Umgang mit (Ziel-, Default-) Namensräumen

Aufgaben bei der Entwicklung eines XML-Schemas:

1. *Definition* neuer Datentypen sowie die *Deklaration* der benötigten Elementtypen.
2. Anwendung eines Entwurfsmusters: anonyme versus benannte Datentypen
3. Optimierung durch Anwendung leistungsfähiger Konzepte (Vererbung, Ableitung, etc.) bei der Definition neuer Datentypen.

XML-Schema

Wie man ein XML-Schema entwickelt (Fortsetzung) [[DTD vs Schema](#)] [[Vergleich](#)]

Zwei Entwurfsmuster zur Deklaration von Elementtypen:

1. `<xs:element name="Elementname"> Typedefinition </xs:element>`
Deklaration des Elementtyps *Elementname* einschließlich der Definition eines Datentyps.
2. `<xs:element name="Elementname" type="Typename" />`
Deklaration des Elementtyps *Elementname* unter Verwendung eines gegebenen Datentyps.

XML-Schema

Wie man ein XML-Schema entwickelt (Fortsetzung) [[DTD vs Schema](#)] [[Vergleich](#)]

Zwei Entwurfsmuster zur Deklaration von Elementtypen:

1. `<xs:element name="Elementname"> Typedefinition </xs:element>`
Deklaration des Elementtyps *Elementname* einschließlich der Definition eines Datentyps.
2. `<xs:element name="Elementname" type="Typename" />`
Deklaration des Elementtyps *Elementname* unter Verwendung eines gegebenen Datentyps.

Datentypdefinitionen:

- `<xs:complexType> ... </xs:complexType>`
Definition eines anonymen, komplexen Datentyps für Instanzen von *Elementname*.
(innerhalb eines `<xs:element name="Elementname">`-Elementes)

XML-Schema

Wie man ein XML-Schema entwickelt (Fortsetzung) [\[DTD vs Schema\]](#) [\[Vergleich\]](#)

Zwei Entwurfsmuster zur Deklaration von Elementtypen:

1. `<xs:element name="Elementname" > Typedefinition </xs:element>`
Deklaration des Elementtyps *Elementname* einschließlich der Definition eines Datentyps.
2. `<xs:element name="Elementname" type="Typename" />`
Deklaration des Elementtyps *Elementname* unter Verwendung eines gegebenen Datentyps.

Datentypdefinitionen:

- ❑ `<xs:complexType> ... </xs:complexType>`
Definition eines anonymen, komplexen Datentyps für Instanzen von *Elementname*.
(innerhalb eines `<xs:element name="Elementname">`-Elementes)
- ❑ `<xs:complexType name="Typename"> ... </xs:complexType>`
Definition eines benannten, komplexen Datentyps mit dem Namen *Typename*.
(global, auf oberster Ebene unter `<xs:schema>`)
- ❑ `<xs:simpleType name="Typename"> ... </xs:simpleType>`
Definition eines benannten, einfachen Datentyps mit dem Namen *Typename*.
(global, auf oberster Ebene unter `<xs:schema>`)

Bemerkungen:

- ❑ W3C-Definition von Inhaltsmodell:
“A content model is a simple grammar governing the allowed types of the child elements and the order in which they are allowed to appear.” [W3C [XML](#), [XSD](#)]
- ❑ Entwurfsmuster 1 zur Deklaration von Elementtypen kommt zur Anwendung, wenn kein passender Datentyp vorhanden ist und dieser *inline (ad-hoc)* und folglich *anonym* definiert werden soll.
- ❑ Entwurfsmuster 2 zur Deklaration von Elementtypen kommt zur Anwendung, wenn ein passender Datentyp vordefiniert (*built-in*) ist oder bereits an anderer Stelle von einem Autor definiert wurde.
- ❑ Zwischen der [Definition von Datentypen](#) und der [Deklaration von Elementtypen](#) ist sorgfältig zu unterscheiden. Das Erstgenannte deklariert einen Wertebereich für einen neu geschaffenen (= definierten) Datentyp; das Zweitgenannte deklariert die erlaubten Elementinstanzen in Instanzdokumenten. [[W3C](#)]
- ❑ Die Definition benannter Datentypen ist sinnvoll, wenn diese mehrfach (mittels des `type`-Attributs) Verwendung finden sollen.

XML-Schema

XML-Schema Teil 2: Datentypen [\[W3C\]](#)

Einteilung der Datentypen:

- ❑ Primitive Datentypen lassen sich nicht auf Basis anderer Datentypen ableiten. Gegensatz: abgeleitete Datentypen
- ❑ Vordefinierte Datentypen (*Built-in Datatypes*) sind Teil der Schema Definition Language XSD [\[W3C\]](#). Gegensatz: anwenderspezifische Datentypen

XML-Schema

XML-Schema Teil 2: Datentypen [\[W3C\]](#)

Einteilung der Datentypen:

- ❑ Primitive Datentypen lassen sich nicht auf Basis anderer Datentypen ableiten. Gegensatz: abgeleitete Datentypen
- ❑ Vordefinierte Datentypen (*Built-in Datatypes*) sind Teil der Schema Definition Language XSD [\[W3C\]](#). Gegensatz: anwenderspezifische Datentypen

Schemaelemente zur Definition anwenderspezifischer Datentypen:

1. `<xs:complexType>`

Definiert einen neuen komplexen Datentyp für Elemente. Kann weitere Elemente (= Kindelemente) deklarieren und Attribute haben.

2. `<xs:simpleType>`

Definiert einen neuen einfachen Datentyp für Elemente und Attribute, der ausschließlich aus Text besteht und weder Attribute noch Kindelemente deklarieren darf.

XML-Schema

XML-Schema Teil 2: Datentypen (Fortsetzung)

Wichtige Schemaelemente, um Constraints für Kindelemente in komplexen Datentypen zu spezifizieren:

Element	Semantik
<code><xs:all></code>	jedes Kindelement kann höchstens ein Mal auftreten
<code><xs:choice></code>	erlaubt das Auftreten eines der Kindelemente gemäß Attributen
<code><xs:sequence></code>	Kindelemente müssen in angegebener Reihenfolge auftreten

Weitere Konzepte zur Definition neuer Datentypen:

- ❑ Vererbung
- ❑ Ableiten durch Einschränkung
- ❑ Ableiten durch Erweiterung
- ❑ Formulierung von Wertebereichs-Constraints z.B. durch reguläre Ausdrücke

Bemerkungen:

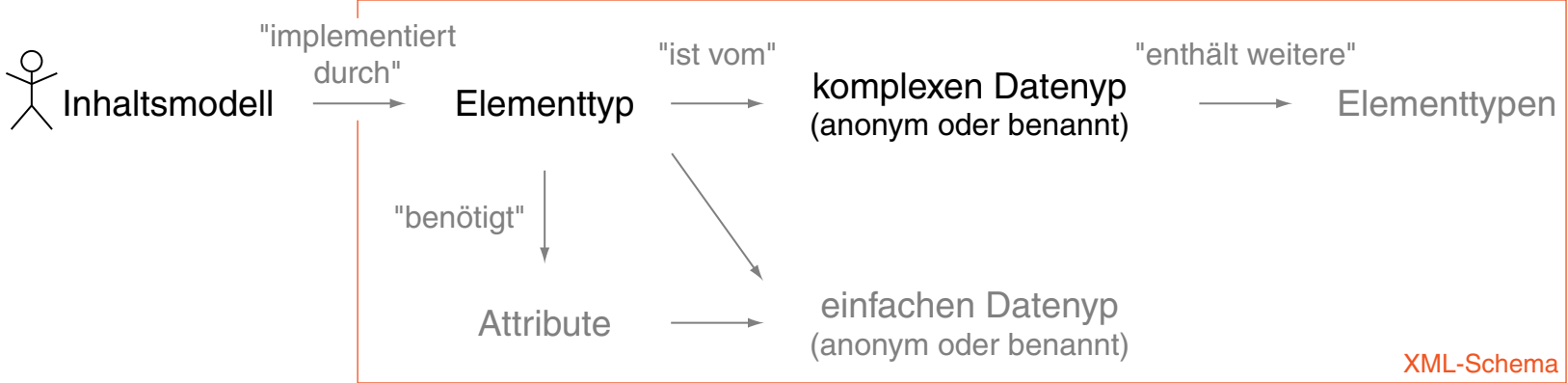
- ❑ Ein Beispiel für einen primitiven Datentyp ist `xs:float`. Dagegen ist `xs:integer` kein primitiver Datentyp, weil er durch Spezialisierung von `xs:decimal` abgeleitet ist. [\[W3C\]](#)
- ❑ Alle vier Datentypkombinationen sind möglich: primitive wie auch abgeleitete Datentypen können vordefiniert (*built-in*) oder anwenderspezifisch sein.
- ❑ Die Definition anwenderspezifischer Datentypen kann *anonym* oder *benannt* geschehen. Die Benennung erfolgt durch das `name`-Attribut im `<xs:complexType>` bzw. `<xs:simpleType>`-Element.

Anonyme Datentypen beziehen sich nur auf die Deklaration der Elementtypen, in der sie eingeführt wurden. Benannte Datentypen sind für alle tieferliegenden Ebenen sichtbar und lassen sich mittels des `type`-Attributes zur Datentypdeklaration verwenden.

- ❑ Bei der Erzeugung anwenderspezifischer Datentypen können innerhalb einer `<xs:complexType>`-Definition noch die Schemaelemente `<xs:simpleContent>` und `<xs:complexContent>` zum Einsatz kommen. Sie dienen zur Deklaration des Inhalts im Zusammenhang mit `restriction` (falls der Inhalt auf bestimmte Datentypen eingeschränkt sein soll) oder mit `extension` (falls der Inhalt hinsichtlich bestimmter Datentypen erweitert werden soll).

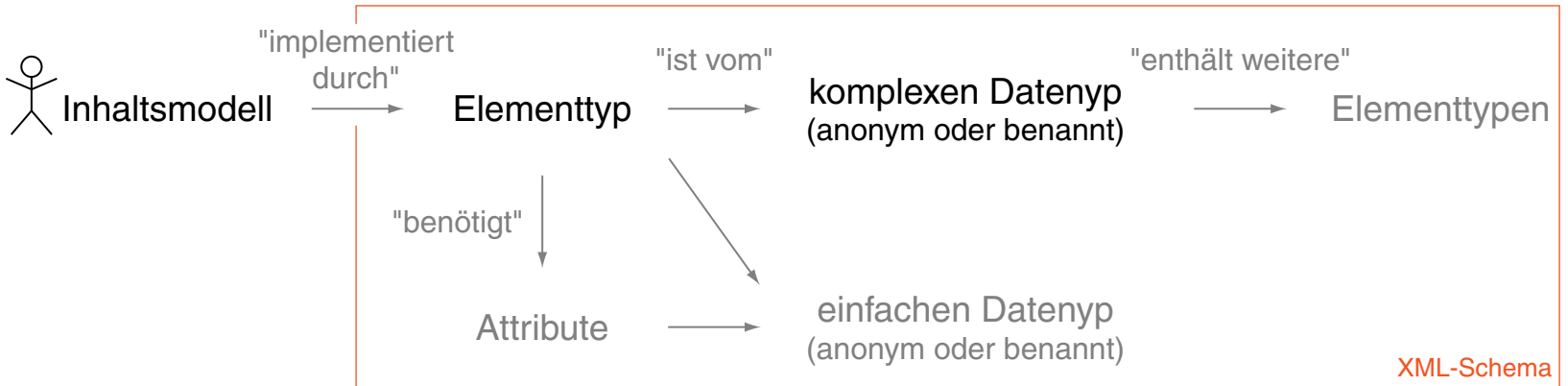
XML-Schema

XML-Schema Teil 1: Inhaltsmodelle



XML-Schema

XML-Schema Teil 1: Inhaltsmodelle



Wichtige Inhaltsmodelle [WT:III DTD] :

- (a) einfacher Inhalt: unstrukturierter, typisierter Text
- (b) explizite Kindelemente: feste Schachtelungsstruktur vorgeschrieben
- (c) gemischter Inhalt: Kombination von Elementen mit unstrukturiertem Text
- (d) beliebiger Inhalt: keine Constraints für Schachtelungsstrukturen
- (e) leerer Inhalt

XML-Schema

XML-Schema Teil 1: Inhaltsmodelle (Fortsetzung)

(a) einfacher Inhalt [\[W3C\]](#) :

```
<xs:element
  name="vorname"
  type="xs:string"
  minOccurs="1"
  maxOccurs="3" />
```

```
<xs:element
  name="Elementname"
  type="Typename" />
```

Vergleichbare DTD-Definition, ohne Typ-Deklaration:

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

Beispiel einer Elementinstanz:

```
<vorname>Tim</vorname>
```


XML-Schema

XML-Schema Teil 1: Inhaltsmodelle (Fortsetzung)

(a) einfacher Inhalt [\[W3C\]](#) :

```
<xs:element
  name="vorname"
  type="xs:string"
  minOccurs="1"
  maxOccurs="3"/>
```

```
<xs:element
  name="Elementname"
  type="Typename" />
```

Vergleichbare DTD-Definition, ohne Typ-Deklaration:

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

Beispiel einer Elementinstanz:

```
<vorname>Tim</vorname>
```

XML-Schema

XML-Schema Teil 1: Inhaltsmodelle (Fortsetzung)

(a) einfacher Inhalt [\[W3C\]](#) :

```
<xs:element
  name="vorname"
  type="xs:string"
  minOccurs="1"
  maxOccurs="3"/>
```

```
<xs:element
  name="Elementname"
  type="Typename" />
```

Vergleichbare DTD-Definition, ohne Typ-Deklaration:

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

Beispiel einer Elementinstanz:

```
<vorname>Tim</vorname>
```

XML-Schema

XML-Schema Teil 1: Inhaltsmodelle (Fortsetzung)

Wichtige Attribute in der Deklaration eines Elementtyps:

Attribut	Semantik
<code>default</code>	Zeichenkette mit Default-Wert, der konform zum gewählten Datentyp ist
<code>minOccurs</code>	Mindestanzahl zulässiger Instanzen dieses Elementtyps, Default ist 1
<code>maxOccurs</code>	Höchstanzahl zulässiger Instanzen dieses Elementtyps, Default ist 1
<code>name</code>	unqualifizierter Name des Elementtyps
<code>ref</code>	Verweis auf eine globale Deklaration des Elementtyps
<code>type</code>	Deklaration des Datentyps für den Elementtyp

Bemerkungen:

- ❑ Für einfache Inhaltsmodelle mit unstrukturiertem Text ist in DTDs nur der Datentyp `#PCDATA` vorgesehen. [XML-Schema Part 2](#) definiert 44 primitive Datentypen.
- ❑ Mit der Spezifikation `maxOccurs="unbounded"` wird eine beliebige Anzahl an Instanzen zugelassen.

XML-Schema

XML-Schema Teil 1: Inhaltsmodelle (Fortsetzung)

(b) explizite Kindelemente [\[W3C\]](#) [\[DTD vs Schema\]](#) :

```
<xs:element name="person">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="vorname" type="xs:string" maxOccurs="3"/>
      <xs:element name="nachname" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Vergleichbare DTD-Definition, ohne Anzahl-Constraint:

```
<!ELEMENT person (vorname+, nachname)>
```

Beispiel einer Elementinstanz:

```
<person>
  <vorname>Tim</vorname>
  <nachname>Berners-Lee</nachname>
</person>
```

XML-Schema

XML-Schema Teil 1: Inhaltsmodelle (Fortsetzung)

(b) explizite Kindelemente [\[W3C\]](#) [\[DTD vs Schema\]](#) :

```
<xs:element name="person">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="vorname" type="xs:string" maxOccurs="3"/>
      <xs:element name="nachname" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Vergleichbare DTD-Definition, ohne Anzahl-Constraint:

```
<!ELEMENT person (vorname+, nachname)>
```

Beispiel einer Elementinstanz:

```
<person>
  <vorname>Tim</vorname>
  <nachname>Berners-Lee</nachname>
</person>
```

Bemerkungen:

- ❑ Das Beispiel folgt [Entwurfsmuster 1](#): Deklaration eines Elementtyps einschließlich der Definition eines neuen (anonymen, komplexen) Datentyps.
- ❑ Das Inhaltsmodell mit explizit angegebenen Kindelementen stellt das wichtigste Inhaltsmodell für die Modellierungspraxis dar.
- ❑ Alternativ zu `<xs:sequence>` zur Angabe der Reihenfolge von Kindelementen kann die Angabe `<xs:choice>` für eine exklusive Auswahl oder `<xs:all>` für eine Liste von Kindelementen in beliebiger Reihenfolge (jedes höchstens einmal) erfolgen.
- ❑ Die angegebene Deklaration des Elementtyps ist eine Abkürzung der folgenden Deklaration, bei der ein komplexes Inhaltsmodell (`<xs:complexContent>`) mittels `<xs:restriction>` eines allgemeineren Typs `<xs:anyType>` notiert ist:

```
<xs:element name="person">
  <xs:complexType>
    <xs:complexContent>
      <xs:restriction base="xs:anyType">
        <xs:sequence>
          <xs:element name="vorname" type="xs:string" maxOccurs="3"/>
          <xs:element name="nachname" type="xs:string"/>
        </xs:sequence>
      </xs:restriction>
    </xs:complexContent>
  </xs:complexType>
</xs:element>
```

XML-Schema

XML-Schema Teil 1: Inhaltsmodelle (Fortsetzung)

(c) gemischter Inhalt [\[W3C\]](#) :

```
<xs:element name="Brief">
  <xs:complexType mixed="true">
    <xs:sequence>
      <xs:element name="Anrede" type="xs:string"/>
      <xs:element name="PS" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Vergleichbare DTD-Definition:

```
<!ELEMENT Brief (#PCDATA | Anrede | PS)*>
```

Beispiel einer Elementinstanz:

```
<Brief>
<Anrede>Sehr geehrter Herr Berners-Lee</Anrede> Die W3C-Empfehlung für
XSD bringt nicht nur Verbesserungen gegenüber DTDs. <PS>Viele Grüße</PS>
</Brief>
```


XML-Schema

XML-Schema Teil 1: Inhaltsmodelle (Fortsetzung)

(c) gemischter Inhalt [\[W3C\]](#) :

```
<xs:element name="Brief">
  <xs:complexType mixed="true">
    <xs:sequence>
      <xs:element name="Anrede" type="xs:string"/>
      <xs:element name="PS" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Vergleichbare DTD-Definition:

```
<!ELEMENT Brief (#PCDATA | Anrede | PS)*>
```

Beispiel einer Elementinstanz:

```
<Brief>
<Anrede>Sehr geehrter Herr Berners-Lee</Anrede> Die W3C-Empfehlung für
XSD bringt nicht nur Verbesserungen gegenüber DTDs. <PS>Viele Grüße</PS>
</Brief>
```

Bemerkungen:

- ❑ Das Beispiel folgt [Entwurfsmuster 1](#): Deklaration eines Elementtyps einschließlich der Definition eines neuen (anonymen, komplexen) Datentyps.
- ❑ Ein gemischtes Inhaltsmodell (*Mixed Content Model*) liegt vor, wenn die Instanz eines Elementtyps sowohl einfachen Inhalt (unstrukturierten Text) als auch Kindelemente aufnehmen kann. Vergleiche die W3C-Definition von Mixed Content:
“An element type has mixed content when elements of that type may contain character data, optionally interspersed with child elements. In this case, the types of the child elements may be constrained, but not their order or their number of occurrences.” [\[W3C\]](#)
- ❑ Während gemischte Inhalte aus Auszeichnungssymbolen und freiem Text durch DTD-validierende Parser nur rudimentär geprüft werden können, ermöglicht XSD eine vollständige Validierung gemischter Inhalte. Die Strukturvalidierung der DTD erlaubt zwar die Spezifikation von innerhalb unstrukturierter Textpassagen auftretenden Elementen, nicht jedoch die Überwachung deren Auftrittshäufigkeit oder Reihenfolge. [\[Jeckle 2004\]](#)

XML-Schema

XML-Schema Teil 1: Inhaltsmodelle (Fortsetzung)

(d) beliebiger Inhalt [\[W3C\]](#) :

```
<xs:element
  name="Elementname"
  type="xs:anyType">
</xs:element>
```

bzw.

```
<xs:element
  name="Elementname">
</xs:element>
```

oder

```
<xs:element name="Elementname" />
```

Vergleichbare DTD-Definition:

```
<!ELEMENT Elementname ANY>
```

Bemerkungen:

- ❑ Wird das `type`-Attribut nicht spezifiziert oder – alternativ – explizit der Wert `xs:anyType` zugewiesen, so können Instanzen dieses Elementtyps beliebige, XML-wohlgeformte Inhalte aufnehmen.

XML-Schema

XML-Schema Teil 1: Inhaltsmodelle (Fortsetzung)

(e) leerer Inhalt [\[W3C\]](#) :

```
<xs:element name="internationalPrice">
  <xs:complexType>
    <xs:attribute name="currency" type="xs:string"/>
    <xs:attribute name="value" type="xs:decimal"/>
  </xs:complexType>
</xs:element>
```

```
<xs:element
  name="Elementname">
  <xs:complexType/>
</xs:element>
```

Vergleichbare DTD-Definition:

```
<!ELEMENT Elementname EMPTY>
```

Beispiel einer Elementinstanz:

```
<internationalPrice currency="EUR" value="423.46"/>
```

XML-Schema

XML-Schema Teil 1: Inhaltsmodelle (Fortsetzung)

(e) leerer Inhalt [\[W3C\]](#) :

```
<xs:element name="internationalPrice">
  <xs:complexType>
    <xs:attribute name="currency" type="xs:string"/>
    <xs:attribute name="value" type="xs:decimal"/>
  </xs:complexType>
</xs:element>
```

```
<xs:element
  name="Elementname">
  <xs:complexType/>
</xs:element>
```

Vergleichbare DTD-Definition:

```
<!ELEMENT Elementname EMPTY>
```

Beispiel einer Elementinstanz:

```
<internationalPrice currency="EUR" value="423.46"/>
```

XML-Schema

XML-Schema Teil 1: Inhaltsmodelle (Fortsetzung)

(e) leerer Inhalt [\[W3C\]](#) :

```
<xs:element name="internationalPrice">
  <xs:complexType>
    <xs:attribute name="currency" type="xs:string"/>
    <xs:attribute name="value" type="xs:decimal"/>
  </xs:complexType>
</xs:element>
```

```
<xs:element
  name="Elementname">
  <xs:complexType/>
</xs:element>
```

Vergleichbare DTD-Definition:

```
<!ELEMENT Elementname EMPTY>
```

Beispiel einer Elementinstanz:

```
<internationalPrice currency="EUR" value="423.46"/>
```

Bemerkungen:

- ❑ Leere Inhaltsmodelle vermitteln ihre Information ausschließlich über Attribute oder über ihre Position innerhalb anderer Elemente.
- ❑ Ein XML-Schema-validierender Parser verhält sich hierbei identisch zu einem DTD-validierenden Parser für das Inhaltsmodell `EMPTY`. D.h., es werden nur die beiden folgenden Darstellungen zur Angabe eines leeren Elements akzeptiert: `<elementName/>`
`<elementName></elementName>`
- ❑ Die angegebene Deklaration des Elementtyps ist eine Abkürzung der folgenden Deklaration, bei der ein komplexes Inhaltsmodell (`<xs:complexContent>`) mittels `<xs:restriction>` eines allgemeineren Typs `<xs:anyType>` notiert ist:

```
<xs:element name="internationalPrice">
  <xs:complexType>
    <xs:complexContent>
      <xs:restriction base="xs:anyType">
        <xs:attribute name="currency" type="xs:string"/>
        <xs:attribute name="value" type="xs:decimal"/>
      </xs:restriction>
    </xs:complexContent>
  </xs:complexType>
</xs:element>
```


XML-Schema

XML-Schema Teil 1: Attribute

Beispiel:

```
<xs:attribute name="myDecimal" type="xs:decimal"/>
```

Innerhalb des `<xs:attribute>`-Elements lassen sich (durch Attribute) spezifische Eigenschaften festlegen, unter anderem:

Attribut	Semantik
<code>default</code>	Zeichenkette mit Default-Wert, der konform zum gewählten Datentyp ist
<code>fixed</code>	unveränderliche Wertbelegung
<code>name</code>	unqualifizierter Name des Attributes
<code>ref</code>	Verweis auf eine globale Attributdefinition
<code>type</code>	Deklaration des Datentyps für das Attribut

Vergleiche hierzu die [DTD-Attribut-Deklaration](#).

XML-Schema

XML-Schema Teil 1: Attribute

Beispiel 1:

```
<xs:element name="book">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="title" type="xs:string"/>
      <xs:element name="author" type="xs:string"/>
    </xs:sequence>
    <xs:attribute name="isbn" type="isbnType" use="required"/>
  </xs:complexType>
</xs:element>
```

XML-Schema

XML-Schema Teil 1: Attribute

Beispiel 1:

```
<xs:element name="book">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="title" type="xs:string"/>
      <xs:element name="author" type="xs:string"/>
    </xs:sequence>
    <xs:attribute name="isbn" type="isbnType" use="required"/>
  </xs:complexType>
</xs:element>
```

Beispiel 2:

```
<xs:element name="description">
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="xs:string">
        <xs:attribute name="source" type="xs:NMTOKEN"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>
```

Bemerkungen:

- ❑ Attribute dürfen nur von einfachem Typ sein.
- ❑ Attribute werden immer am Ende einer Deklaration angegeben, also vor `</xs:extension>`, `</xs:restriction>` oder `</xs:complexType>`.
- ❑ Beispiel 2 illustriert, dass auch bei der Deklaration eines Elementtyps ohne Kindelemente eine Attributdeklaration immer dazu führt, dass mittels `</xs:complexType>` ein komplexer Datentyp definiert werden muss.
- ❑ In der folgenden, nicht abgekürzten Deklaration für das Beispiel 1 zeigt sich die kanonische Herangehensweise bei der Deklaration von Elementtypen:

```
<xs:element name="book">
  <xs:complexType>
    <xs:complexContent>
      <xs:restriction base="xs:anyType">
        <xs:sequence>
          <xs:element name="title" type="xs:string"/>
          <xs:element name="author" type="xs:string"/>
        </xs:sequence>
        <xs:attribute name="isbn" type="isbnType" use="required"/>
      </xs:restriction>
    </xs:complexContent>
  </xs:complexType>
</xs:element>
```

XML-Schema

Schemaentwurf [Vlist 2001]

Instanzdokument:

Being a Dog Is a Full-Time Job

von Charles M. Schulz

Darsteller 1:

Name: Snoopy
befreundet mit: Peppermint Patty
seit: 1990-10-04
Eigenschaften: extroverted beagle

Darsteller 2:

Name: Peppermint Patty
seit: 1996-08-22
Eigenschaften: bold, brash and tomboyish

XML-Schema

Schemaentwurf [Vlist 2001]

Instanzdokument:

Being a Dog Is a Full-Time Job

von Charles M. Schulz

Darsteller 1:

Name: Snoopy
befreundet mit: Peppermint Patty
seit: 1990-10-04
Eigenschaften: extroverted beagle

Darsteller 2:

Name: Peppermint Patty
seit: 1996-08-22
Eigenschaften: bold, brash and tomboyish

Mögliche DTD:

```
<!ELEMENT book (title, author, character+)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT author (#PCDATA)>
<!ELEMENT character (name, since?, qualification*)>
...
```

XML-Schema

Schemaentwurf

XML-Source des Instanzdokuments:

```
<?xml version="1.0" encoding="UTF-8"?>
<book isbn="0836217462">

  <title>
  <author>

  <character>
    <name>
    <friend-of>
    <since>
    <qualification>
  </character>

  <character>
    <name>
    <since>
    <qualification>
  </character>

</book>
```

XML-Schema

Schemaentwurf

XML-Source des Instanzdokuments:

```
<?xml version="1.0" encoding="UTF-8"?>
<book isbn="0836217462">

  <title>Being a Dog Is a Full-Time Job</title>
  <author>Charles M. Schulz</author>

  <character>
    <name>Snoopy</name>
    <friend-of>Peppermint Patty</friend-of>
    <since>1950-10-04</since>
    <qualification>extroverted beagle</qualification>
  </character>

  <character>
    <name>Peppermint Patty</name>
    <since>1966-08-22</since>
    <qualification>bold, brash and tomboyish</qualification>
  </character>

</book>
```


XML-Schema

Entwurfsmuster 1a: „Russian Doll Design“

Deklarationen von Elementtypen isoliert betrachtet:

```
<xs:element name="book">
  <xs:complexType>
    <xs:sequence>
      ...
    </xs:sequence>
  </xs:complexType>
</xs:element>
```



XML-Schema

Entwurfsmuster 1a: „Russian Doll Design“

Deklarationen von Elementtypen isoliert betrachtet:

```
<xs:element name="book">  
  <xs:complexType>  
    <xs:sequence>  
      ...  
    </xs:sequence>  
  </xs:complexType>  
</xs:element>
```

```
<xs:attribute name="isbn" type="xs:string"/>
```

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

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

```
<xs:element name="character" minOccurs="0" maxOccurs="unbounded">  
  <xs:complexType>  
    <xs:sequence>  
      <xs:element name="name" type="xs:string"/>  
      ...  
      <xs:element name="qualification" type="xs:string"/>  
    </xs:sequence>  
  </xs:complexType>  
</xs:element>
```



XML-Schema

Entwurfsmuster 1a: „Russian Doll Design“ (Fortsetzung)

Ineinander geschachtelte Deklarationen von Elementtypen:

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:xs="https://www.w3.org/2001/XMLSchema">
  <xs:element name="book">
    <xs:complexType>
      <xs:sequence>

        </xs:sequence>

      </xs:complexType>
    </xs:element>
  </xs:schema>
```



XML-Schema

Entwurfsmuster 1a: „Russian Doll Design“ (Fortsetzung)

Ineinander geschachtelte Deklarationen von Elementtypen:

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:xs="https://www.w3.org/2001/XMLSchema">
  <xs:element name="book">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="title" type="xs:string"/>
        <xs:element name="author" type="xs:string"/>
      </xs:sequence>
      <xs:attribute name="isbn" type="xs:string"/>
    </xs:complexType>
  </xs:element>
</xs:schema>
```



XML-Schema

Entwurfsmuster 1a: „Russian Doll Design“ (Fortsetzung)

Ineinander geschachtelte Deklarationen von Elementtypen:

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:xs="https://www.w3.org/2001/XMLSchema">
  <xs:element name="book">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="title" type="xs:string"/>
        <xs:element name="author" type="xs:string"/>
        <xs:element name="character" minOccurs="0" maxOccurs="unbounded">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="name" type="xs:string"/>
              ...
              <xs:element name="qualification" type="xs:string"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
      <xs:attribute name="isbn" type="xs:string"/>
    </xs:complexType>
  </xs:element>
</xs:schema>
```



Bemerkungen:

- ❑ Bei dem Entwurfsmuster „Russian Doll Design“ ist das entstehende XML-Schema eng an das Instanzdokument angelehnt; die Elementtypen werden entsprechend ihrer Verwendung im Instanzdokument deklariert.
- ❑ Nachteil: Es können tiefgeschachtelte Schemata entstehen, die schwer zu verstehen und zu pflegen sind. Solche Schemata unterscheiden sich in ihrem Aufbau deutlich von den entsprechenden DTDs.

XML-Schema

Entwurfsmuster 1b: Verweise auf globale Deklarationen

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="https://www.w3.org/2001/XMLSchema">

  <!-- definition of simple type elements -->
  <xs:element name="title" type="xs:string"/>
  <xs:element name="author" type="xs:string"/>
  <xs:element name="name" type="xs:string"/>
  ...
  <xs:element name="qualification" type="xs:string"/>
  <!-- definition of attributes -->
  <xs:attribute name="isbn" type="xs:string"/>
```

XML-Schema

Entwurfsmuster 1b: Verweise auf globale Deklarationen

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="https://www.w3.org/2001/XMLSchema">

  <!-- definition of simple type elements -->
  <xs:element name="title" type="xs:string"/>
  <xs:element name="author" type="xs:string"/>
  <xs:element name="name" type="xs:string"/>
  ...
  <xs:element name="qualification" type="xs:string"/>
  <!-- definition of attributes -->
  <xs:attribute name="isbn" type="xs:string"/>

  <!-- definition of complex type elements -->
  <xs:element name="character">
    <xs:complexType>
      <xs:sequence>
        <!-- simple type elements are referenced with "ref" attribute -->
        <xs:element ref="name"/>
        ...
        <xs:element ref="qualification"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
```


XML-Schema

Entwurfsmuster 1b: Verweise auf globale Deklarationen (Fortsetzung)

```
<xs:element name="book">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="title"/>
      <xs:element ref="author"/>
      <!-- definition of cardinality when elements are referenced -->
      <xs:element ref="character" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute ref="isbn"/>
  </xs:complexType>
</xs:element>

</xs:schema>
```

Bemerkungen:

- ❑ Wie bei dem „Russian Doll Design“ werden auch hier die Elementtypen entsprechend ihrer Verwendung im Instanzdokument deklariert. Eine Auffaltung der Schachtelung wird dadurch vermieden, dass – ausgehend von den Blättern (also bottom-up) – für jede Ebene des Dokumentbaums eine Liste der in dieser Ebene neu hinzukommenden Elementtypen erstellt wird.

Die Deklaration von Elementtypen mit Kindelementen geschieht mittels Verweisen auf Elementtypen aus dieser Liste. Vergleiche hierzu das „Russian Doll Design“, bei dem statt der Verweise die gesamte Deklaration eingebettet wird.

- ❑ Man kann die Entwurfsmuster (1a) und (1b) als „Entwurf auf Basis anonymer Datentypen“ bezeichnen: die Deklarationen der Elementtypen für `character` und `book` basieren auf einem direkt (*inline, ad-hoc*) definierten, anonymen Datentyp. Sind mehrere Elementtypen gleich aufgebaut, so besitzen sie bis auf ihren Namen die gleiche Deklaration, was zu Redundanz führt. Die Alternative besteht in der Definition eines benannten Datentyps.
- ❑ Durch das Referenzierungskonzept existiert eine erste Möglichkeit zur Wiederverwendung bereits im Schema definierter Elementtypen. Bei dieser einfachen Form der Textersetzung werden die Elementtypen literal, also mit ihrem Namen eingebunden. Im Grunde genommen ist diese Art der Wiederverwendung bereits mit den Mitteln von DTDs möglich. [\[Jeckle 2004\]](#)
- ❑ “Another authoring style, applicable when all element names are unique within a namespace, is to create schemas in which all elements are global. This is similar in effect to the use of `<!ELEMENT>` in a DTD.” [\[W3C\]](#)

XML-Schema

Entwurfsmuster 2: Definition benannter Datentypen

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="https://www.w3.org/2001/XMLSchema">

  <!-- definition of named simple types -->
  <xs:simpleType name="nameType">
    <xs:restriction base="xs:string">
      <xs:maxLength value="32"/>
    </xs:restriction>
  </xs:simpleType>
```

XML-Schema

Entwurfsmuster 2: Definition benannter Datentypen

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="https://www.w3.org/2001/XMLSchema">

  <!-- definition of named simple types -->
  <xs:simpleType name="nameType">
    <xs:restriction base="xs:string">
      <xs:maxLength value="32"/>
    </xs:restriction>
  </xs:simpleType>
```

XML-Schema

Entwurfsmuster 2: Definition benannter Datentypen

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="https://www.w3.org/2001/XMLSchema">

  <!-- definition of named simple types -->
  <xs:simpleType name="nameType">
    <xs:restriction base="xs:string">
      <xs:maxLength value="32"/>
    </xs:restriction>
  </xs:simpleType>
```

XML-Schema

Entwurfsmuster 2: Definition benannter Datentypen

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="https://www.w3.org/2001/XMLSchema">

  <!-- definition of named simple types -->
  <xs:simpleType name="nameType">
    <xs:restriction base="xs:string">
      <xs:maxLength value="32"/>
    </xs:restriction>
  </xs:simpleType>

  <xs:simpleType name="sinceType">
    <xs:restriction base="xs:date"/>
  </xs:simpleType>

  <xs:simpleType name="qualificationType">
    <xs:restriction base="xs:string"/>
  </xs:simpleType>

  <xs:simpleType name="isbnType">
    <xs:restriction base="xs:string">
      <xs:pattern value="[0-9]10"/>
    </xs:restriction>
  </xs:simpleType>


```

XML-Schema

Entwurfsmuster 2: Definition benannter Datentypen (Fortsetzung)

```
<!-- definition of named complex types -->
<xs:complexType name="characterType">
  <xs:sequence>
    <xs:element name="name" type="nameType"/>
    ...
    <xs:element name="qualification" type="qualificationType"/>
  </xs:sequence>
</xs:complexType>
```

XML-Schema

Entwurfsmuster 2: Definition benannter Datentypen (Fortsetzung)

```
<!-- definition of named complex types -->
<xs:complexType name="characterType">
  <xs:sequence>
    <xs:element name="name" type="nameType"/>
    ...
    <xs:element name="qualification" type="qualificationType"/>
  </xs:sequence>
</xs:complexType>
```


XML-Schema

Entwurfsmuster 2: Definition benannter Datentypen (Fortsetzung)

```
<!-- definition of named complex types -->
<xs:complexType name="characterType">
  <xs:sequence>
    <xs:element name="name" type="nameType"/>
    ...
    <xs:element name="qualification" type="qualificationType"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="bookType">
  <xs:sequence>
    <xs:element name="title" type="nameType"/>
    <xs:element name="author" type="nameType"/>
    <xs:element name="character" type="characterType" minOccurs="0"/>
  </xs:sequence>
  <xs:attribute name="isbn" type="isbnType" use="required"/>
</xs:complexType>

</xs:schema>
```

XML-Schema

Entwurfsmuster 2: Definition benannter Datentypen (Fortsetzung)

```
<!-- definition of named complex types -->
<xs:complexType name="characterType">
  <xs:sequence>
    <xs:element name="name" type="nameType"/>
    ...
    <xs:element name="qualification" type="qualificationType"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="bookType">
  <xs:sequence>
    <xs:element name="title" type="nameType"/>
    <xs:element name="author" type="nameType"/>
    <xs:element name="character" type="characterType" minOccurs="0"/>
  </xs:sequence>
  <xs:attribute name="isbn" type="isbnType" use="required"/>
</xs:complexType>

<!-- The "book" element is of the data type "bookType" -->
<xs:element name="book" type="bookType"/>

</xs:schema>
```

XML-Schema

Gegenüberstellung der Entwurfsmuster [\[Vergleich\]](#)

1. Deklaration eines Elementtyps mit anonymem Datentyp:

```
<xs:element name="bondCar">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="color" type="xs:string"/>
      <xs:element name="enginepower" type="xs:integer"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

2. Deklaration eines Elementtyps mit benanntem Datentyp:

```
<xs:complexType name="automobile">
  <xs:sequence>
    <xs:element name="color" type="xs:string"/>
    <xs:element name="enginepower" type="xs:integer"/>
  </xs:sequence>
</xs:complexType>

<xs:element name="bondCar" type="automobile"/>
```

XML-Schema

Quellen zum Nachlernen und Nachschlagen im Web: Referenz

- ❑ MSDN. *Referenz zu XML-Schemata (XSD)*.
msdn.microsoft.com/en-us/library/aa153011
- ❑ W3C. *XML Schema Part 0: Primer*.
www.w3.org/TR/xmlschema-0
- ❑ W3C. *XML Schema Definition Language (XSD) 1.1 Part 1: Structures*.
www.w3.org/TR/xmlschema11-1/
- ❑ W3C. *XML Schema Definition Language (XSD) 1.1 Part 2: Datatypes*.
www.w3.org/TR/xmlschema11-2/

XML-Schema

Quellen zum Nachlernen und Nachschlagen im Web: Usage

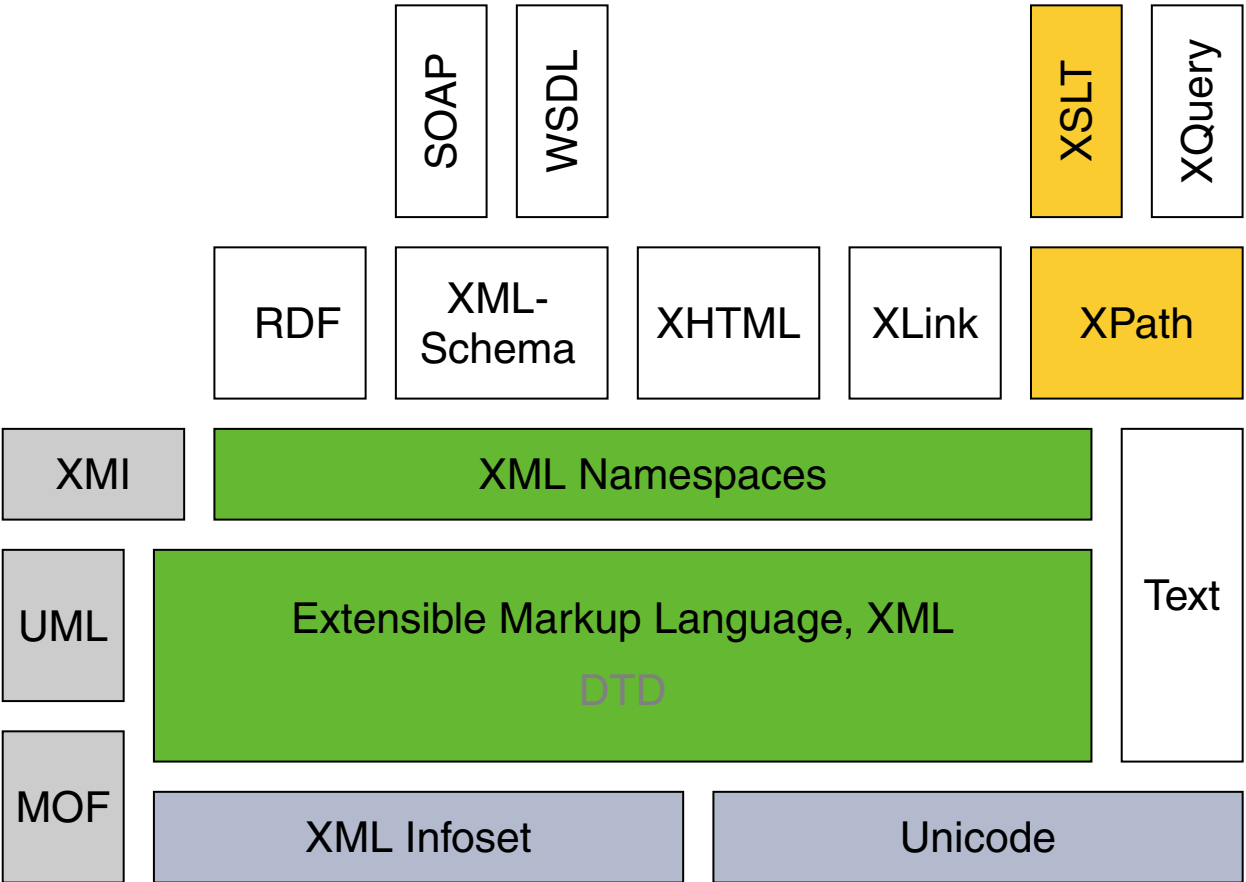
- ❑ Cover Pages. *XML Schemas*.
xml.coverpages.org/schemas.html
- ❑ Liquid Technologies. *XML Schema*.
www.liquid-technologies.com/Tutorials/XmlSchemas
- ❑ Vlist. *Using W3C XML Schema*.
www.xml.com/lpt/a/2000/11/29/schemas/part1.html
- ❑ W3 Schools. *XML Schema*.
www.w3schools.com/schema

III. Dokumentensprachen

- ❑ Auszeichnungssprachen
- ❑ HTML
- ❑ Cascading Stylesheets CSS
- ❑ XML-Grundlagen
- ❑ XML-Schema
- ❑ Die XSL-Familie
- ❑ APIs für XML-Dokumente

Die XSL-Familie

Einordnung [Jeckle 2004]



Die XSL-Familie

Historie: zentrale XML-Spezifikationen

2008 XML 1.0, Recommendation. [W3C [REC](#), [status](#)]

2006 XML 1.1, Recommendation. [W3C [REC](#), [status](#)]

2004 XML-Schema Part 0: Primer, Recommendation. [W3C [REC](#), [status](#)]

2012 XML-Schema (XSD) 1.1 Part 1: Structures. [W3C [REC](#), [status](#)]

2012 XML-Schema (XSD) 1.1 Part 2: Datatypes. [W3C [REC](#), [status](#)]

2013 XSL Transformations (XSLT) 3.0. Latest Working Draft. [W3C [WD](#), [status](#)]

2014 XML Path Language (XPath) 3.0. Recommendation. [W3C [REC](#), [status](#)]

2014 XML Query Language (XQuery) 3.0. Recommendation. [W3C [REC](#), [status](#)]

2012 XSL Formatting Objects (XSL-FO) 2.0. Working Draft. [W3C [WD](#), [status](#)]

Bemerkungen:

- ❑ XSL (*Extensible Style Language*) ist eine Formatierungssprache, die einem mit XML-basierten Markup versehenen Dokument eine Layout-Vorschrift zuordnet.
- ❑ “XSL is a family of recommendations for defining XML document transformation and presentation. It consists of three parts:” XSLT, XPath, XSL-FO. [\[W3C\]](#)
- ❑ XSL benutzt dasselbe Formattierungsmodell wie CSS (*Cascading Stylesheets*) und ermöglicht die Einbettung von Programm-Code zur Durchführung komplexer Ausgabetransformationen.
- ❑ CSS versus XML. Why two Style Sheet languages? [\[W3C\]](#)
- ❑ Wiederholung: Der Standardisierungsprozess der W3C ist formalisiert und spiegelt sich in den verschiedenen Leveln der veröffentlichten Reports wider. [\[W3C level\]](#)

Die XSL-Familie

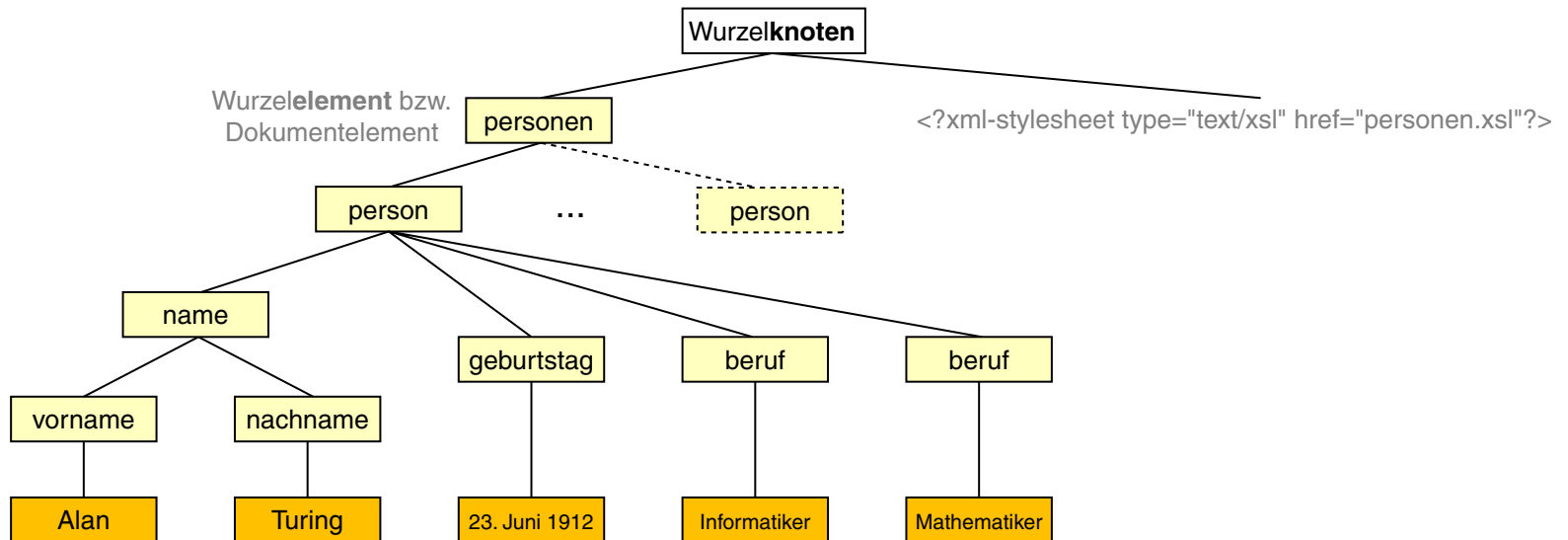
Verwendung von XPath

XSLT	Finden und Auswählen von Elementen im Eingabedokument, die in das Ausgabedokument kopiert werden.
XQuery	Finden und Auswählen von Elementen.
XPointer	Identifikation einer Stelle im XML-Dokument, auf die ein XLink verweist.
XML-Schema	Formulierung von Constraints hinsichtlich der Eindeutigkeit oder der Identität von Elementen.
XForm	Bindung von Formularsteuerungen an Instanzdaten; Formulierung von Werte-Constraints und Berechnungen.

Die XSL-Familie

XML-Knotentypen unter dem XPath-Modell

1. Wurzelknoten
2. Elementknoten
3. Textknoten
4. Attributknoten
5. Kommentarknoten
6. Verarbeitungsanweisungsknoten
7. Namensraumknoten



Bemerkungen:

- ❑ Der *Wurzelknoten* eines XML-Dokuments ist nicht identisch mit dem *Wurzelement*: Der Wurzelknoten entspricht dem *Document Information Item* des XML Information Sets. Das Wurzelement hingegen ist das erste benannte Element des Dokuments und wird durch ein *Element Information Item* dargestellt.
- ❑ XPath dient der Navigation in Dokumenten und der Auswahl von Dokumentbestandteilen; XPath ist keine Datenmanipulationssprache.
- ❑ XPath-Ausdrücke können zu einzelnen Knoten (XML-Element, XML-Attribut), zu Knotenmengen, zu Zeichenketten, zu Zahlen und zu Bool'schen Werten evaluieren. Deshalb stellt XPath Funktionen zum Zugriff auf Knotenmengen und zur Manipulation verschiedener Datentypen zur Verfügung.
- ❑ Wiederholung. Das W3C hat mittlerweile drei Datenmodelle für XML-Dokumente definiert: XML Information Set, XPath, Document Object Model (DOM). Das XPath-Datenmodell basiert auf einer Baumstruktur, die bei der Abfrage eines XML-Dokuments durchlaufen wird und ist dem XML Information Set ähnlich; DOM ist der Vorläufer beider Datenmodelle. DOM und das XPath-Datenmodell können als Interpretationen des XML Information Sets betrachtet werden. [\[MSDN\]](#)

Die XSL-Familie

XPath-Lokalisierungspfade

- ❑ Ein Lokalisierungspfad spezifiziert eine eventuell leere Menge von Knoten in einem XML-Dokument.
- ❑ Ein Lokalisierungspfad setzt sich aus aufeinander folgenden Lokalisierungsschritten (*Location steps*) zusammen.
- ❑ Jeder Lokalisierungsschritt wird relativ zu einem bestimmten Knoten des XML-Dokuments ausgewertet, der dann als **aktueller Knoten** (*Current node*) oder **Kontextknoten** bezeichnet wird.

Die XSL-Familie

XPath-Lokalisierungspfade

- Ein Lokalisierungspfad spezifiziert eine eventuell leere Menge von Knoten in einem XML-Dokument.
- Ein Lokalisierungspfad setzt sich aus aufeinander folgenden Lokalisierungsschritten (*Location steps*) zusammen.
- Jeder Lokalisierungsschritt wird relativ zu einem bestimmten Knoten des XML-Dokuments ausgewertet, der dann als **aktueller Knoten** (*Current node*) oder **Kontextknoten** bezeichnet wird.

- Lokalisierungsschritte werden durch Schrägstriche (*Slashes*) getrennt:

... /*Schritt_i* /*Schritt_{i+1}* / ...

- Beginnt ein Lokalisierungspfad mit einem Schrägstrich, bezeichnet dieser den Wurzelknoten. Der Wurzelknoten ist dann aktueller Knoten zum ersten Lokalisierungsschritt:

/*Schritt_1*

Die XSL-Familie

XPath-Lokalisierungspfade (Fortsetzung)

```
<?xml version="1.0" standalone="no" ?>
<?xml-stylesheet type="text/xsl" ...?>

<personen>
  <person>
    <name>
      <vorname>Alan</vorname>
      <nachname>Turing</nachname>
    </name>
    <geburtstag>23. Juni 1912</geburtstag>
    <beruf>Mathematiker</beruf>
    <beruf>Informatiker</beruf>
  </person>

  <person>
    <name>
      <vorname>Judea</vorname>
      <nachname>Pearl</nachname>
    </name>
    <geburtstag>unknown</geburtstag>
    <beruf>Informatiker</beruf>
  </person>
</personen>
```

Beispiele für Lokalisierungspfade:

(a) /personen/person

Die XSL-Familie

XPath-Lokalisierungspfade (Fortsetzung)

```
<?xml version="1.0" standalone="no" ?>
<?xml-stylesheet type="text/xsl" ...?>

<personen>
  <person>
    <name>
      <vorname>Alan</vorname>
      <nachname>Turing</nachname>
    </name>
    <geburtstag>23. Juni 1912</geburtstag>
    <beruf>Mathematiker</beruf>
    <beruf>Informatiker</beruf>
  </person>

  <person>
    <name>
      <vorname>Judea</vorname>
      <nachname>Pearl</nachname>
    </name>
    <geburtstag>unknown</geburtstag>
    <beruf>Informatiker</beruf>
  </person>
</personen>
```

Beispiele für Lokalisierungspfade:

(a) /personen/person

Die XSL-Familie

XPath-Lokalisierungspfade (Fortsetzung)

```
<?xml version="1.0" standalone="no" ?>
<?xml-stylesheet type="text/xsl" ...?>

<personen>
  <person>
    <name>
      <vorname>Alan</vorname>
      <nachname>Turing</nachname>
    </name>
    <geburtstag>23. Juni 1912</geburtstag>
    <beruf>Mathematiker</beruf>
    <beruf>Informatiker</beruf>
  </person>

  <person>
    <name>
      <vorname>Judea</vorname>
      <nachname>Pearl</nachname>
    </name>
    <geburtstag>unknown</geburtstag>
    <beruf>Informatiker</beruf>
  </person>
</personen>
```

Beispiele für Lokalisierungspfade:

(a) /personen/person

(b) /personen/person[1]/name/vorname

Die XSL-Familie

XPath-Lokalisierungspfade (Fortsetzung)

```
<?xml version="1.0" standalone="no" ?>
<?xml-stylesheet type="text/xsl" ...?>

<personen>
  <person>
    <name>
      <vorname>Alan</vorname>
      <nachname>Turing</nachname>
    </name>
    <geburtstag>23. Juni 1912</geburtstag>
    <beruf>Mathematiker</beruf>
    <beruf>Informatiker</beruf>
  </person>

  <person>
    <name>
      <vorname>Judea</vorname>
      <nachname>Pearl</nachname>
    </name>
    <geburtstag>unknown</geburtstag>
    <beruf>Informatiker</beruf>
  </person>
</personen>
```

Beispiele für Lokalisierungspfade:

(a) /personen/person

(b) /personen/person[1]/name/vorname

Die XSL-Familie

XPath-Lokalisierungspfade (Fortsetzung)

```
<?xml version="1.0" standalone="no" ?>
<?xml-stylesheet type="text/xsl" ...?>

<personen>
  <person>
    <name>
      <vorname>Alan</vorname>
      <nachname>Turing</nachname>
    </name>
    <geburtstag>23. Juni 1912</geburtstag>
    <beruf>Mathematiker</beruf>
    <beruf>Informatiker</beruf>
  </person>

  <person>
    <name>
      <vorname>Judea</vorname>
      <nachname>Pearl</nachname>
    </name>
    <geburtstag>unknown</geburtstag>
    <beruf>Informatiker</beruf>
  </person>
</personen>
```

Beispiele für Lokalisierungspfade:

(a) /personen/person

(b) /personen/person[1]/name/vorname

(c) /personen/person[1]/beruf

Die XSL-Familie

XPath-Lokalisierungspfade (Fortsetzung)

```
<?xml version="1.0" standalone="no" ?>
<?xml-stylesheet type="text/xsl" ...?>

<personen>
  <person>
    <name>
      <vorname>Alan</vorname>
      <nachname>Turing</nachname>
    </name>
    <geburtstag>23. Juni 1912</geburtstag>
    <beruf>Mathematiker</beruf>
    <beruf>Informatiker</beruf>
  </person>

  <person>
    <name>
      <vorname>Judea</vorname>
      <nachname>Pearl</nachname>
    </name>
    <geburtstag>unknown</geburtstag>
    <beruf>Informatiker</beruf>
  </person>
</personen>
```

Beispiele für Lokalisierungspfade:

(a) /personen/person

(b) /personen/person[1]/name/vorname

(c) /personen/person[1]/beruf

Die XSL-Familie

XPath-Lokalisierungspfade (Fortsetzung)

Allgemeine Form eines Lokalisierungsschritts:

Achse : : *Knotentest* [*Prädikat*]

Default = child:: optional

Die XSL-Familie

XPath-Lokalisierungspfade (Fortsetzung)

Allgemeine Form eines Lokalisierungsschritts:

$$\underbrace{Achse} : : \underbrace{Knotentest} [\underbrace{Prädikat}]$$

Default = child:: optional

1. Achsen. Spezifizieren Knotenmengen relativ zum aktuellen Knoten. Es werden 13 Achsen unterschieden.
2. Knotentests. Filtern die durch eine Achse (1.) spezifizierte Knotenmenge weiter. Hierzu gibt es für jeden Knotentyp ein Testschema.

Beispiele:

- ein qualifizierender Name ~ Test auf Knoten mit diesem Namen
- die Funktion `text()` ~ Test auf Textknoten

3. Prädikate. Filtern die durch Achse (1.) und Knotentest (2.) spezifizierte Knotenmenge weiter. Jeder gültige XPath-Ausdruck kann Prädikat sein.

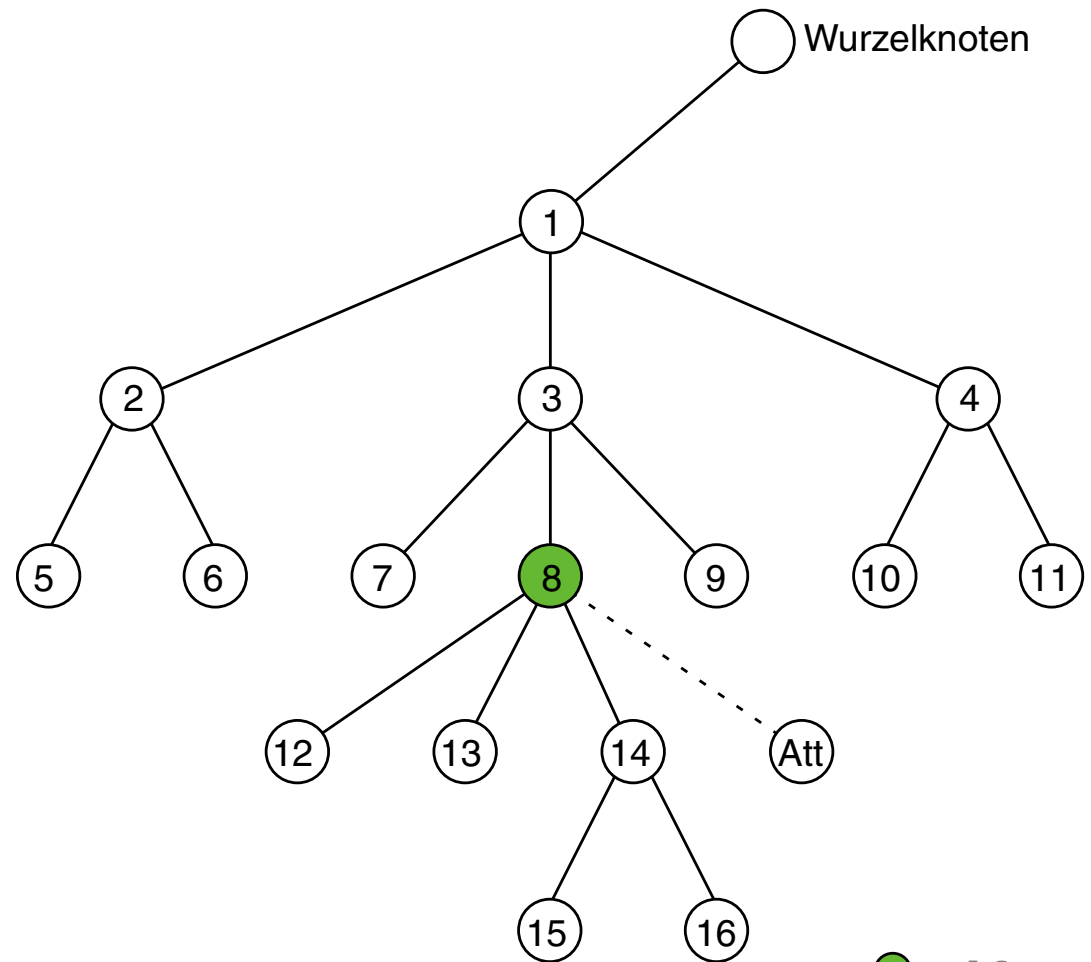
Beispiele: Test auf Kindknoten, Position bzw. Index

Die XSL-Familie

XPath-Lokalisierungspfade (Fortsetzung)

Illustration wichtiger Achsen [\[Jeckle 2004\]](#) :

```
<?xml version="1.0">  
<node1>  
  <node2>  
    <node5/>  
    <node6/>  
  </node2>  
  <node3>  
    <node7/>  
    <node8 Att="42">  
      <node12/>  
      <node13/>  
      <node14>  
        <node15/>  
        <node16/>  
      </node14>  
    </node8>  
    <node9/>  
  </node3>  
  <node4>  
    <node10/>  
    <node11/>  
  </node4>  
</node1>
```

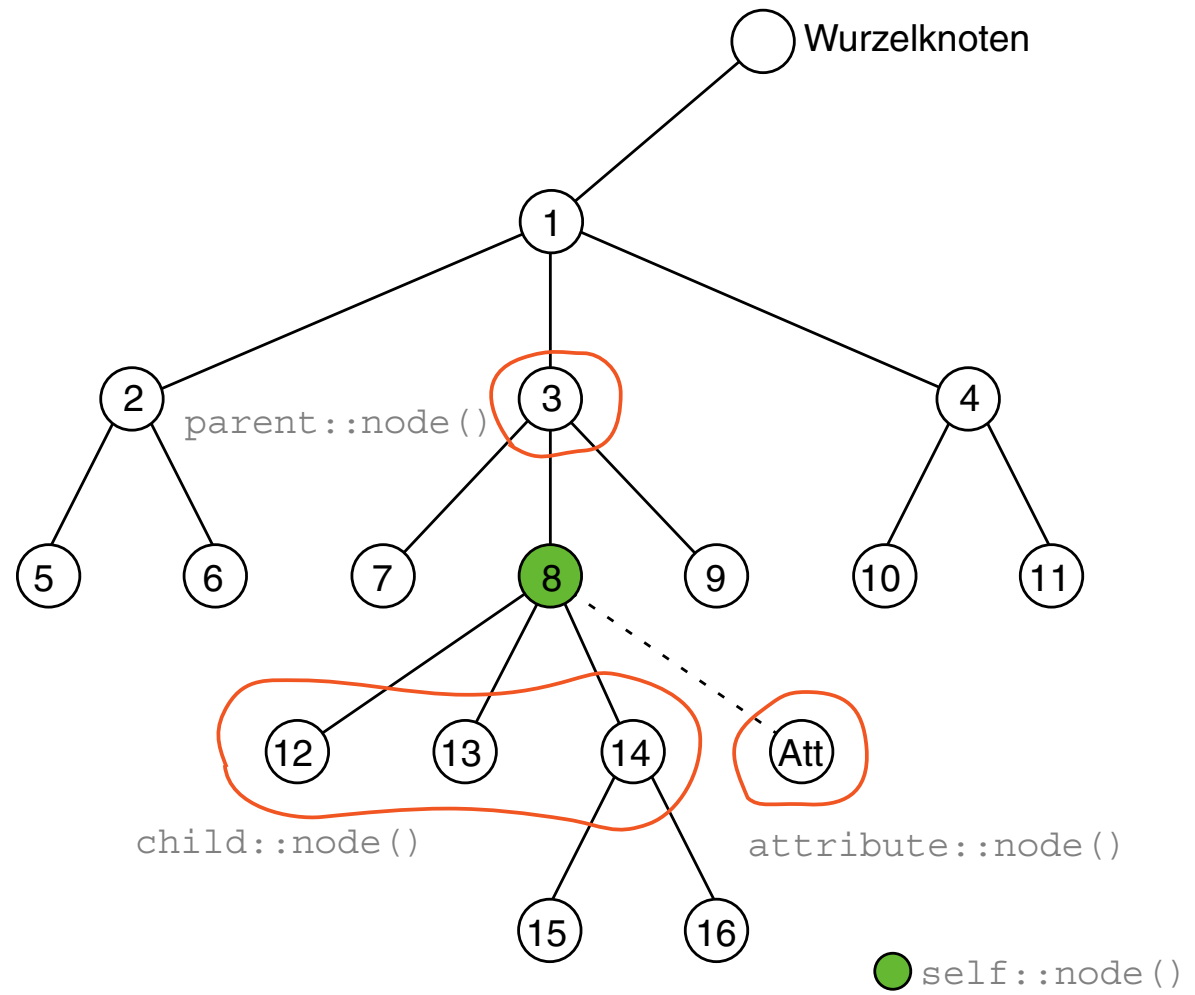


Die XSL-Familie

XPath-Lokalisierungspfade (Fortsetzung)

Illustration wichtiger Achsen [Jeckle 2004] :

```
<?xml version="1.0">  
<node1>  
  <node2>  
    <node5/>  
    <node6/>  
  </node2>  
  <node3>  
    <node7/>  
    <node8 Att="42">  
      <node12/>  
      <node13/>  
      <node14>  
        <node15/>  
        <node16/>  
      </node14>  
    </node8>  
    <node9/>  
  </node3>  
  <node4>  
    <node10/>  
    <node11/>  
  </node4>  
</node1>
```

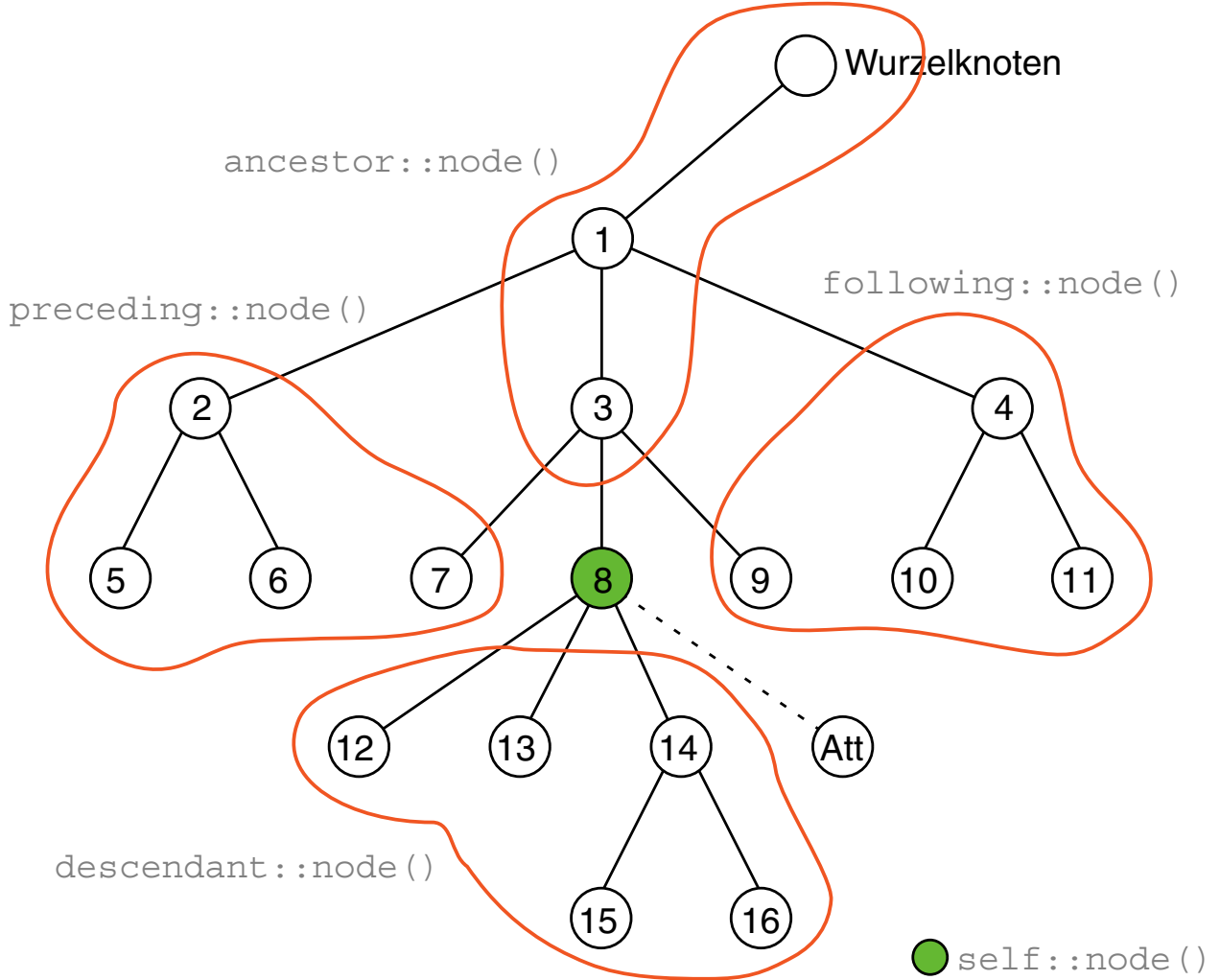


Die XSL-Familie

XPath-Lokalisierungspfade (Fortsetzung)

Illustration wichtiger Achsen [Jeckle 2004] :

```
<?xml version="1.0">  
<node1>  
  <node2>  
    <node5/>  
    <node6/>  
  </node2>  
  <node3>  
    <node7/>  
    <node8 Att="42">  
      <node12/>  
      <node13/>  
      <node14>  
        <node15/>  
        <node16/>  
      </node14>  
    </node8>  
    <node9/>  
  </node3>  
  <node4>  
    <node10/>  
    <node11/>  
  </node4>  
</node1>
```



Die XSL-Familie

XPath-Lokalisierungspfade (Fortsetzung)

- Schreibweisen häufig verwendeter Lokalisierungsschritte:

kurz	lang	Semantik
.	<code>self::node()</code>	Kontextknoten bzw. aktueller Knoten
..	<code>parent::node()</code>	Elternknoten des aktuellen Knotens
//	<code>/descendant-or-self::node()</code>	alle Nachkommen des aktuellen Knotens einschließlich diesem
@*	<code>attribute::node()</code>	alle Attributknoten

- Knotentypen und Wildcards für Namen in Knotentests:

<code>node()</code>	jeder <u>Knotentyp</u>
<code>*</code>	Knoten mit Namen (Element- oder Attributknoten)
<code>text()</code>	Textknoten
<code>comment()</code>	Kommentarknoten
<code>processing-instruction()</code>	Verarbeitungsanweisungsknoten

- Spezifikation von alternativen Lokalisierungspfaden:

`Pfad_1 | Pfad_2 | ... | Pfad_n`

Die XSL-Familie

XPath-Lokalisierungspfade (Fortsetzung)

- Schreibweisen häufig verwendeter Lokalisierungsschritte:

kurz	lang	Semantik
.	<code>self::node()</code>	Kontextknoten bzw. aktueller Knoten
..	<code>parent::node()</code>	Elternknoten des aktuellen Knotens
//	<code>/descendant-or-self::node()</code>	alle Nachkommen des aktuellen Knotens einschließlich diesem
@*	<code>attribute::node()</code>	alle Attributknoten

- Knotentypen und Wildcards für Namen in Knotentests:

<code>node()</code>	jeder <u>Knotentyp</u>
<code>*</code>	Knoten mit Namen (Element- oder Attributknoten)
<code>text()</code>	Textknoten
<code>comment()</code>	Kommentarknoten
<code>processing-instruction()</code>	Verarbeitungsanweisungsknoten

- Spezifikation von alternativen Lokalisierungspfaden:

Pfad_1 | Pfad_2 | ... | Pfad_n

Die XSL-Familie

XPath-Lokalisierungspfade (Fortsetzung)

- Schreibweisen häufig verwendeter Lokalisierungsschritte:

kurz	lang	Semantik
.	<code>self::node()</code>	Kontextknoten bzw. aktueller Knoten
..	<code>parent::node()</code>	Elternknoten des aktuellen Knotens
//	<code>/descendant-or-self::node()</code>	alle Nachkommen des aktuellen Knotens einschließlich diesem
@*	<code>attribute::node()</code>	alle Attributknoten

- Knotentypen und Wildcards für Namen in Knotentests:

<code>node()</code>	jeder <u>Knotentyp</u>
*	Knoten mit Namen (Element- oder Attributknoten)
<code>text()</code>	Textknoten
<code>comment()</code>	Kommentarknoten
<code>processing-instruction()</code>	Verarbeitungsanweisungsknoten

- Spezifikation von alternativen Lokalisierungspfaden:

Pfad_1 | *Pfad_2* | ... | *Pfad_n*

Die XSL-Familie

XPath-Lokalisierungspfade (Fortsetzung)

```
<?xml version="1.0" standalone="no" ?>
<?xml-stylesheet type="text/xsl" ...?>

<personen>
  <person>
    <name>
      <vorname>Alan</vorname>
      <nachname>Turing</nachname>
    </name>
    <geburtstag>23. Juni 1912</geburtstag>
    <beruf>Mathematiker</beruf>
    <beruf>Informatiker</beruf>
  </person>

  <person>
    <name>
      <vorname>Judea</vorname>
      <nachname>Pearl</nachname>
    </name>
    <geburtstag>unknown</geburtstag>
    <beruf>Informatiker</beruf>
  </person>
</personen>
```

Beispiele für Lokalisierungspfade:

(a) //person/name/descendant::*

Die XSL-Familie

XPath-Lokalisierungspfade (Fortsetzung)

```
<?xml version="1.0" standalone="no" ?>
<?xml-stylesheet type="text/xsl" ...?>

<personen>
  <person>
    <name>
      <vorname>Alan</vorname>
      <nachname>Turing</nachname>
    </name>
    <geburtstag>23. Juni 1912</geburtstag>
    <beruf>Mathematiker</beruf>
    <beruf>Informatiker</beruf>
  </person>

  <person>
    <name>
      <vorname>Judea</vorname>
      <nachname>Pearl</nachname>
    </name>
    <geburtstag>unknown</geburtstag>
    <beruf>Informatiker</beruf>
  </person>
</personen>
```

Beispiele für Lokalisierungspfade:

(a) //person/name/descendant::*

Die XSL-Familie

XPath-Lokalisierungspfade (Fortsetzung)

```
<?xml version="1.0" standalone="no" ?>
<?xml-stylesheet type="text/xsl" ...?>

<personen>
  <person>
    <name>
      <vorname>Alan</vorname>
      <nachname>Turing</nachname>
    </name>
    <geburtstag>23. Juni 1912</geburtstag>
    <beruf>Mathematiker</beruf>
    <beruf>Informatiker</beruf>
  </person>

  <person>
    <name>
      <vorname>Judea</vorname>
      <nachname>Pearl</nachname>
    </name>
    <geburtstag>unknown</geburtstag>
    <beruf>Informatiker</beruf>
  </person>
</personen>
```

Beispiele für Lokalisierungspfade:

(a) //person/name/descendant::*

(b) //geburtstag/parent::*[1]/name

Die XSL-Familie

XPath-Lokalisierungspfade (Fortsetzung)

```
<?xml version="1.0" standalone="no" ?>
<?xml-stylesheet type="text/xsl" ...?>

<personen>
  <person>
    <name>
      <vorname>Alan</vorname>
      <nachname>Turing</nachname>
    </name>
    <geburtstag>23. Juni 1912</geburtstag>
    <beruf>Mathematiker</beruf>
    <beruf>Informatiker</beruf>
  </person>

  <person>
    <name>
      <vorname>Judea</vorname>
      <nachname>Pearl</nachname>
    </name>
    <geburtstag>unknown</geburtstag>
    <beruf>Informatiker</beruf>
  </person>
</personen>
```

Beispiele für Lokalisierungspfade:

(a) //person/name/descendant::*

(b) //geburtstag/parent::*[1]

Die XSL-Familie

XPath-Lokalisierungspfade (Fortsetzung)

```
<?xml version="1.0" standalone="no" ?>
<?xml-stylesheet type="text/xsl" ...?>

<personen>
  <person>
    <name>
      <vorname>Alan</vorname>
      <nachname>Turing</nachname>
    </name>
    <geburtstag>23. Juni 1912</geburtstag>
    <beruf>Mathematiker</beruf>
    <beruf>Informatiker</beruf>
  </person>

  <person>
    <name>
      <vorname>Judea</vorname>
      <nachname>Pearl</nachname>
    </name>
    <geburtstag>unknown</geburtstag>
    <beruf>Informatiker</beruf>
  </person>
</personen>
```

Beispiele für Lokalisierungspfade:

- (a) //person/name/descendant::*
- (b) //geburtstag/parent::*[1]/name
- (c) /personen/child::name

Die XSL-Familie

XPath-Lokalisierungspfade (Fortsetzung)

```
<?xml version="1.0" standalone="no" ?>
<?xml-stylesheet type="text/xsl" ...?>

<personen>
  <person>
    <name>
      <vorname>Alan</vorname>
      <nachname>Turing</nachname>
    </name>
    <geburtstag>23. Juni 1912</geburtstag>
    <beruf>Mathematiker</beruf>
    <beruf>Informatiker</beruf>
  </person>

  <person>
    <name>
      <vorname>Judea</vorname>
      <nachname>Pearl</nachname>
    </name>
    <geburtstag>unknown</geburtstag>
    <beruf>Informatiker</beruf>
  </person>
</personen>
```

Beispiele für Lokalisierungspfade:

- (a) //person/name/descendant::*
- (b) //geburtstag/parent::*[1]/name
- (c) /personen/child::name

Die XSL-Familie

XPath-Lokalisierungspfade (Fortsetzung)

```
<?xml version="1.0" standalone="no" ?>
<?xml-stylesheet type="text/xsl" ...?>

<personen>
  <person>
    <name>
      <vorname>Alan</vorname>
      <nachname>Turing</nachname>
    </name>
    <geburtstag>23. Juni 1912</geburtstag>
    <beruf>Mathematiker</beruf>
    <beruf>Informatiker</beruf>
  </person>

  <person>
    <name>
      <vorname>Judea</vorname>
      <nachname>Pearl</nachname>
    </name>
    <geburtstag>unknown</geburtstag>
    <beruf>Informatiker</beruf>
  </person>
</personen>
```

Beispiele für Lokalisierungspfade:

- (a) //person/name/descendant::*
- (b) //geburtstag/parent::* /name[1]
- (c) /personen/child::name
- (d) /personen/descendant::name

Die XSL-Familie

XPath-Lokalisierungspfade (Fortsetzung)

```
<?xml version="1.0" standalone="no" ?>
<?xml-stylesheet type="text/xsl" ...?>

<personen>
  <person>
    <name>
      <vorname>Alan</vorname>
      <nachname>Turing</nachname>
    </name>
    <geburtstag>23. Juni 1912</geburtstag>
    <beruf>Mathematiker</beruf>
    <beruf>Informatiker</beruf>
  </person>

  <person>
    <name>
      <vorname>Judea</vorname>
      <nachname>Pearl</nachname>
    </name>
    <geburtstag>unknown</geburtstag>
    <beruf>Informatiker</beruf>
  </person>
</personen>
```

Beispiele für Lokalisierungspfade:

- (a) //person/name/descendant::*
- (b) //geburtstag/parent::* /name[1]
- (c) /personen/child::name
- (d) /personen/descendant::name

Die XSL-Familie

XPath-Lokalisierungspfade (Fortsetzung)

```
<?xml version="1.0" standalone="no" ?>
<?xml-stylesheet type="text/xsl" ...?>

<personen>
  <person>
    <name>
      <vorname>Alan</vorname>
      <nachname>Turing</nachname>
    </name>
    <geburtstag>23. Juni 1912</geburtstag>
    <beruf>Mathematiker</beruf>
    <beruf>Informatiker</beruf>
  </person>

  <person>
    <name>
      <vorname>Judea</vorname>
      <nachname>Pearl</nachname>
    </name>
    <geburtstag>unknown</geburtstag>
    <beruf>Informatiker</beruf>
  </person>
</personen>
```

Beispiele für Lokalisierungspfade:

- (a) //person/name/descendant::*
- (b) //geburtstag/parent::* /name[1]
- (c) /personen/child::name
- (d) /personen/descendant::name
- (e) //person[geburtstag!='unknown']

Die XSL-Familie

XPath-Lokalisierungspfade (Fortsetzung)

```
<?xml version="1.0" standalone="no" ?>
<?xml-stylesheet type="text/xsl" ...?>

<personen>
  <person>
    <name>
      <vorname>Alan</vorname>
      <nachname>Turing</nachname>
    </name>
    <geburtstag>23. Juni 1912</geburtstag>
    <beruf>Mathematiker</beruf>
    <beruf>Informatiker</beruf>
  </person>

  <person>
    <name>
      <vorname>Judea</vorname>
      <nachname>Pearl</nachname>
    </name>
    <geburtstag>unknown</geburtstag>
    <beruf>Informatiker</beruf>
  </person>
</personen>
```

Beispiele für Lokalisierungspfade:

- (a) //person/name/descendant::*
- (b) //geburtstag/parent::* /name[1]
- (c) /personen/child::name
- (d) /personen/descendant::name
- (e) //person[geburtstag!='unknown']

Die XSL-Familie

XPath-Lokalisierungspfade (Fortsetzung)

Algorithmus zur Auswertung eines Lokalisierungspfades:

$\dots / \text{Schritt}_i / \text{Schritt}_{i+1} / \dots$

1. Die Auswertung der Lokalisierungsschritte geschieht von links nach rechts.
2. Jeder Lokalisierungsschritt spezifiziert eine Knotenmenge M .

Die XSL-Familie

XPath-Lokalisierungspfade (Fortsetzung)

Algorithmus zur Auswertung eines Lokalisierungspfades:

$\dots / \text{Schritt}_i / \text{Schritt}_{i+1} / \dots$

1. Die Auswertung der Lokalisierungsschritte geschieht von links nach rechts.
2. Jeder Lokalisierungsschritt spezifiziert eine Knotenmenge M .
3. Jeder Knoten n der Knotenmenge M_i des Lokalisierungsschritts i wird als Kontextknoten hinsichtlich des Lokalisierungsschritts $i + 1$ interpretiert und spezifiziert im Lokalisierungsschritt $i + 1$ die Knotenmenge M_{i_n} .
4. Die Vereinigung der Mengen M_{i_n} , $n \in M_i$, bildet die Knotenmenge M_{i+1} des Lokalisierungsschritts $i + 1$.

Bemerkungen:

- ❑ Jeder der in einem Schritt spezifizierten Knoten kommt für die weitere Auswertung in die Rolle des Kontextknotens.
- ❑ **Beispiel:** `//beruf[3]` bzw. `/descendant-or-self::node()/beruf[3]` selektiert von jedem Elementknoten das jeweils dritte `<beruf>`-Kindelement, und nicht etwa, wie man vermuten könnte, das dritte `<beruf>`-Element in dem gesamten Dokument. Letzteres erreicht man mit `/descendant-or-self::beruf[3]`, das zunächst alle `<beruf>`-Elemente spezifiziert und darunter das dritte spezifiziert.

Die XSL-Familie

Lokalisierungspfade (Fortsetzung)

Knoten

```
1
2 <?xml version="1.0" standalone="no" ?>
3 <?xml-stylesheet type="text/xsl" ...?>
4 <personen>
5   <person>
6     <name>
7       <vorname>Alan</vorname>
8       <nachname>Turing</nachname>
6     </name>
9     <geburtstag>23. Juni 1912</geburtstag>
10    <beruf>Mathematiker</beruf>
11    <beruf>Informatiker</beruf>
5  </person>
12 <person>
13   <name>
14     <vorname>Judea</vorname>
15     <nachname>Pearl</nachname>
13   </name>
16   <geburtstag>unknown</geburtstag>
17   <beruf>Informatiker</beruf>
12  </person>
4 </personen>
```

Illustration des Algorithmus:

```
//person/name/descendant::*
//person/name/descendant::*
```

$$M_1 = \{1\}$$

$$M_2 = \{1, \dots, 17\}$$

$$M_3 = \{5, 12\}$$

$$M_4 = \{6, 13\}$$

$$M_5 = \{7, 8, 14, 15\}$$

Die XSL-Familie

Lokalisierungspfade (Fortsetzung)

Knoten

```
1
2 <?xml version="1.0" standalone="no" ?>
3 <?xml-stylesheet type="text/xsl" ...?>
4 <personen>
5   <person>
6     <name>
7       <vorname>Alan</vorname>
8       <nachname>Turing</nachname>
6     </name>
9     <geburtstag>23. Juni 1912</geburtstag>
10    <beruf>Mathematiker</beruf>
11    <beruf>Informatiker</beruf>
5  </person>
12 <person>
13   <name>
14     <vorname>Judea</vorname>
15     <nachname>Pearl</nachname>
13   </name>
16   <geburtstag>unknown</geburtstag>
17   <beruf>Informatiker</beruf>
12  </person>
4 </personen>
```

Illustration des Algorithmus:

```
//person/name/descendant::*
//person/name/descendant::*
```

$$M_1 = \{1\}$$

$$M_2 = \{1, \dots, 17\}$$

$$M_3 = \{5, 12\}$$

$$M_4 = \{6, 13\}$$

$$M_5 = \{7, 8, 14, 15\}$$

Die XSL-Familie

Lokalisierungspfade (Fortsetzung)

Knoten

```
1
2 <?xml version="1.0" standalone="no" ?>
3 <?xml-stylesheet type="text/xsl" ...?>
4 <personen>
5   <person>
6     <name>
7       <vorname>Alan</vorname>
8       <nachname>Turing</nachname>
6     </name>
9     <geburtstag>23. Juni 1912</geburtstag>
10    <beruf>Mathematiker</beruf>
11    <beruf>Informatiker</beruf>
5  </person>
12 <person>
13   <name>
14     <vorname>Judea</vorname>
15     <nachname>Pearl</nachname>
13   </name>
16   <geburtstag>unknown</geburtstag>
17   <beruf>Informatiker</beruf>
12 </person>
4 </personen>
```

Illustration des Algorithmus:

```
//person/name/descendant::*
//person/name/descendant::*
```

$$M_1 = \{1\}$$

$$M_2 = \{1, \dots, 17\}$$

$$M_3 = \{5, 12\}$$

$$M_4 = \{6, 13\}$$

$$M_5 = \{7, 8, 14, 15\}$$

Die XSL-Familie

Lokalisierungspfade (Fortsetzung)

Knoten

```
1
2 <?xml version="1.0" standalone="no" ?>
3 <?xml-stylesheet type="text/xsl" ...?>
4 <personen>
5   <person>
6     <name>
7       <vorname>Alan</vorname>
8       <nachname>Turing</nachname>
6     </name>
9     <geburtstag>23. Juni 1912</geburtstag>
10    <beruf>Mathematiker</beruf>
11    <beruf>Informatiker</beruf>
5   </person>
12  <person>
13    <name>
14      <vorname>Judea</vorname>
15      <nachname>Pearl</nachname>
13    </name>
16    <geburtstag>unknown</geburtstag>
17    <beruf>Informatiker</beruf>
12  </person>
4 </personen>
```

Illustration des Algorithmus:

```
//person/name/descendant::*
//person/name/descendant::*
```

$$M_1 = \{1\}$$

$$M_2 = \{1, \dots, 17\}$$

$$M_3 = \{5, 12\}$$

$$M_4 = \{6, 13\}$$

$$M_5 = \{7, 8, 14, 15\}$$

Die XSL-Familie

Lokalisierungspfade (Fortsetzung)

Knoten

```
1
2 <?xml version="1.0" standalone="no" ?>
3 <?xml-stylesheet type="text/xsl" ...?>
4 <personen>
5   <person>
6     <name>
7       <vorname>Alan</vorname>
8       <nachname>Turing</nachname>
6     </name>
9     <geburtstag>23. Juni 1912</geburtstag>
10    <beruf>Mathematiker</beruf>
11    <beruf>Informatiker</beruf>
5  </person>
12 <person>
13   <name>
14     <vorname>Judea</vorname>
15     <nachname>Pearl</nachname>
13   </name>
16   <geburtstag>unknown</geburtstag>
17   <beruf>Informatiker</beruf>
12  </person>
4 </personen>
```

Illustration des Algorithmus:

```
//person/name/descendant::*
//person/name/descendant::*
```

$$M_1 = \{1\}$$

$$M_2 = \{1, \dots, 17\}$$

$$M_3 = \{5, 12\}$$

$$M_4 = \{6, 13\}$$

$$M_5 = \{7, 8, 14, 15\}$$

Die XSL-Familie

Lokalisierungspfade (Fortsetzung)

Knoten

```
1
2 <?xml version="1.0" standalone="no" ?>
3 <?xml-stylesheet type="text/xsl" ...?>
4 <personen>
5   <person>
6     <name>
7       <vorname>Alan</vorname>
8       <nachname>Turing</nachname>
6     </name>
9     <geburtstag>23. Juni 1912</geburtstag>
10    <beruf>Mathematiker</beruf>
11    <beruf>Informatiker</beruf>
5   </person>
12  <person>
13    <name>
14      <vorname>Judea</vorname>
15      <nachname>Pearl</nachname>
13    </name>
16    <geburtstag>unknown</geburtstag>
17    <beruf>Informatiker</beruf>
12  </person>
4 </personen>
```

Illustration des Algorithmus:

```
//person/name/descendant::*
//person/name/descendant::*
```

$$M_1 = \{1\}$$

$$M_2 = \{1, \dots, 17\}$$

$$M_3 = \{5, 12\}$$

$$M_4 = \{6, 13\}$$

$$M_5 = \{7, 8, 14, 15\}$$

Die XSL-Familie

XPath-Lokalisierungspfade (Fortsetzung)

Komplexes Beispiel:

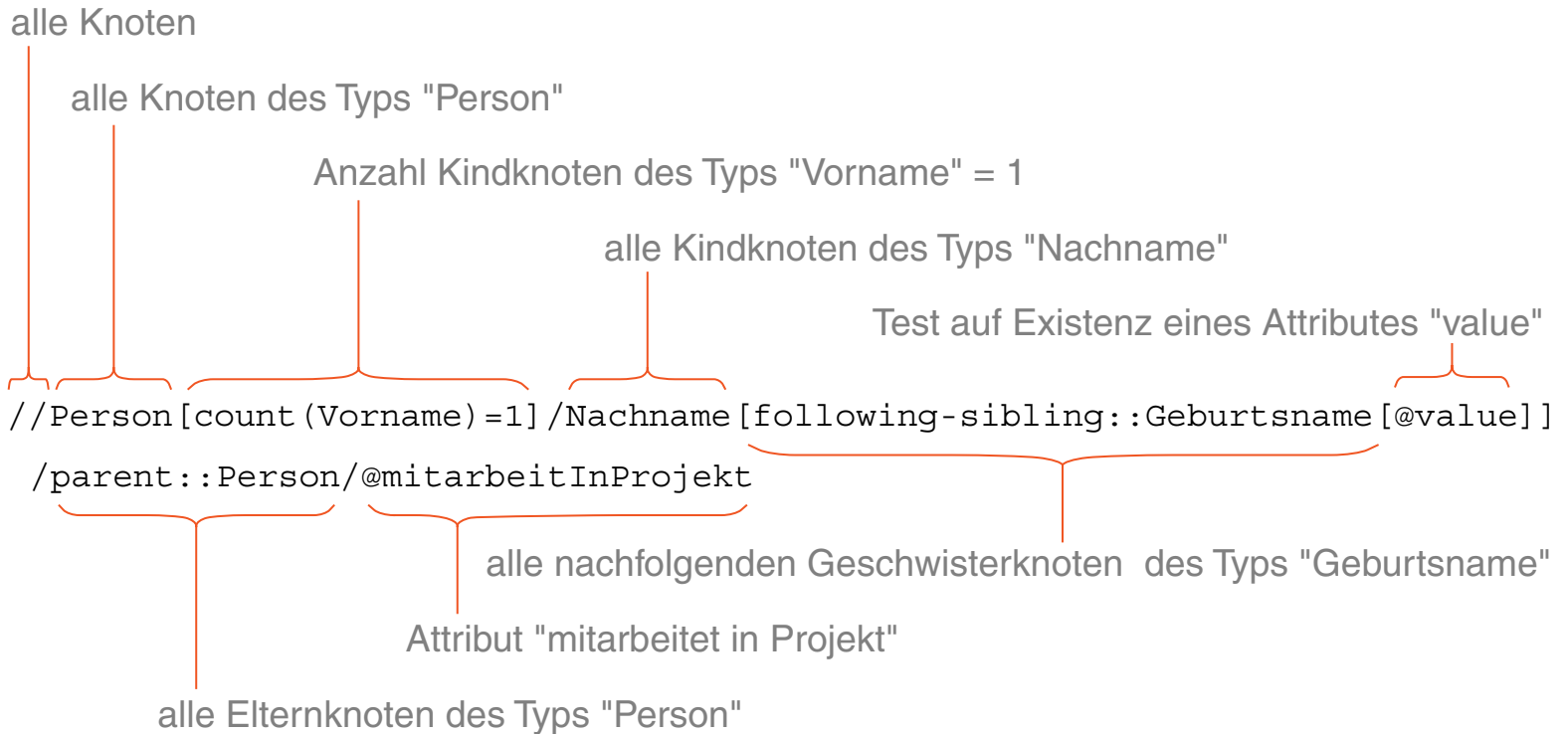
```
//Person[count(Vorname)=1]/Nachname[following-sibling::Geburtsname[@value]]  
/parent::Person/@mitarbeitInProjekt
```

[[Jeckle 2004](#)]

Die XSL-Familie

XPath-Lokalisierungspfade (Fortsetzung)

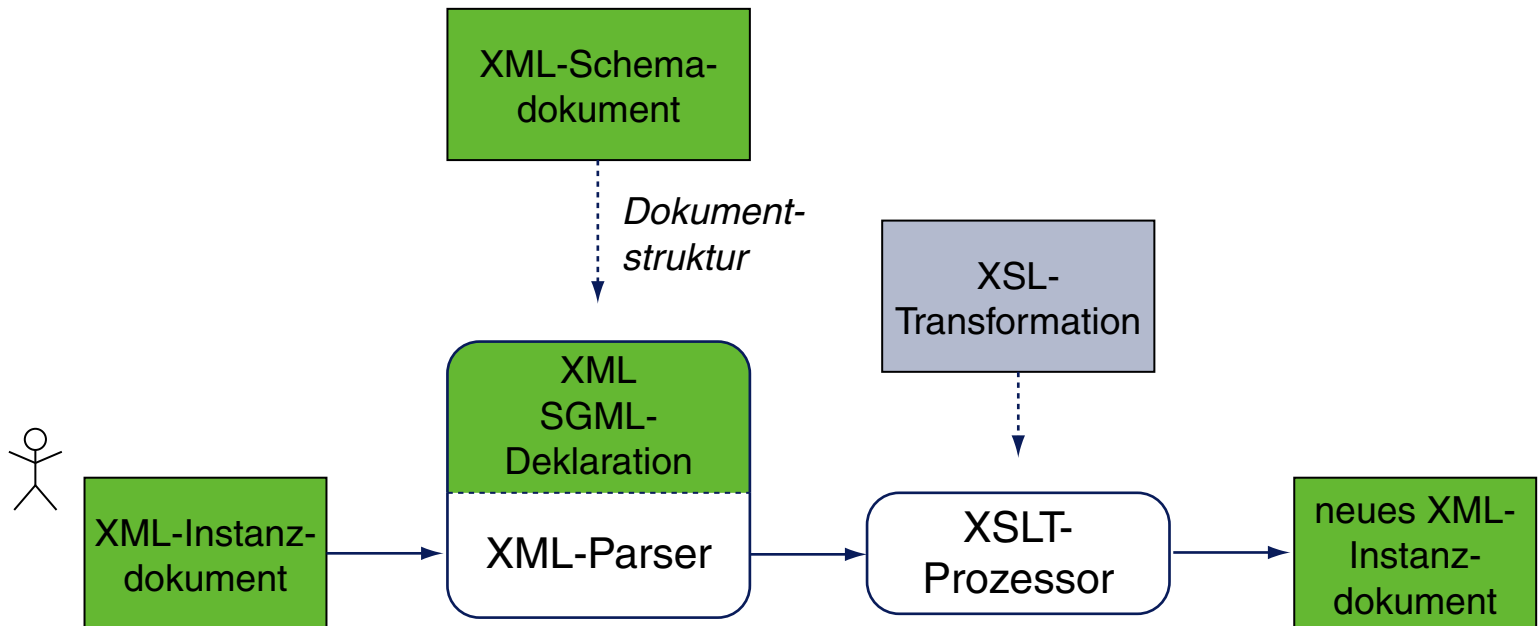
Komplexes Beispiel:



[Jeckle 2004]

Die XSL-Familie

XSL Transformation



XSLT ist eine Turing-vollständige Programmiersprache zur Transformation wohlgeformter XML-Dokumente in andere XML-Dokumente. Ein XSLT-Programm liegt üblicherweise als XSL-Stylesheet vor.

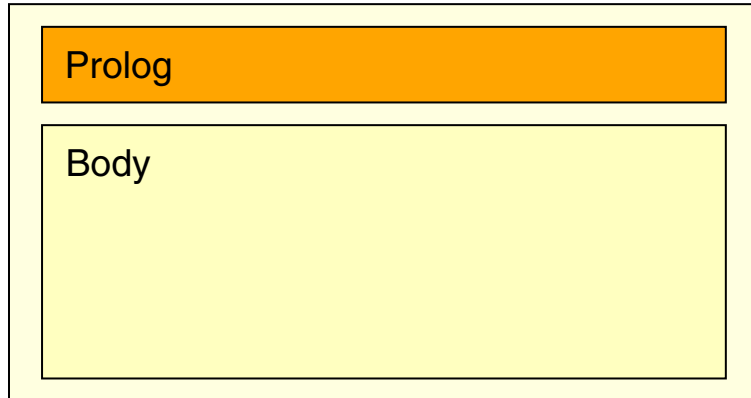
Die Transformation umfasst die Selektion von Teilen des Eingabedokuments, deren Umordnung sowie die Generierung neuer Inhalte aus den bestehenden.

Die XSL-Familie

Aufbau eines XSL-Stylesheets

XSL-Stylesheets sind XML-Dokumente:

XSL-
Stylesheet



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

```
<xsl:stylesheet xmlns:xsl=...>  
  <xsl:template match=...>  
    ...  
  </xsl:template>  
  ...  
</xsl:stylesheet>
```

- ❑ Wurzelement jedes XSL-Schemas ist das Element `<xsl:stylesheet>` oder synonym `<xsl:transform>`.
- ❑ Die Kindelemente von `<xsl:stylesheet>` bzw. `<xsl:transform>` definieren Transformationsvorschriften in Form von Template-Regeln.
- ❑ Vergleiche hierzu die XML-Dokumentstruktur und die XML-Schema-Dokumentstruktur.

Bemerkungen:

- ❑ Das Vokabular zur Definition von XSL-Stylesheets gehört zum Namensraum <https://www.w3.org/1999/XSL/Transform>. Das übliche Präfix bei der Namensraumdeklaration ist `xsl:`, es kann aber beliebig gewählt werden. Wird der offizielle Namensraum gebunden, ist auch das Attribut `version="1.0"` anzugeben.
- ❑ Die Dateiendung einer XSL-Stylesheet-Datei ist `.xsl`.
- ❑ Aufbau einer realen Turingmaschine.

Die XSL-Familie

XML-Beispieldokument

```
<?xml version="1.0" standalone="no" ?>
<?xml-stylesheet type="text/xsl" href="personen2html.xsl"?>

<personen>
  <person>
    <name>
      <vorname>Alan</vorname>
      <nachname>Turing</nachname>
    </name>
    <geburtstag>23. Juni 1912</geburtstag>
    <beruf>Mathematiker</beruf>
    <beruf>Informatiker</beruf>
  </person>

  <person>
    <name>
      <vorname>Judea</vorname>
      <nachname>Pearl</nachname>
    </name>
    <geburtstag>unknown</geburtstag>
    <beruf>Informatiker</beruf>
  </person>
</personen>
```

Die XSL-Familie

Elemente eines XSL-Stylesheets

Das einfachste (leere) Stylesheet:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="https://www.w3.org/1999/XSL/Transform">
</xsl:stylesheet>
```

Die XSL-Familie

Elemente eines XSL-Stylesheets

Das einfachste (leere) Stylesheet:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="https://www.w3.org/1999/XSL/Transform">
</xsl:stylesheet>
```

Die XSL-Familie

Elemente eines XSL-Stylesheets

Das einfachste (leere) Stylesheet:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="https://www.w3.org/1999/XSL/Transform">
</xsl:stylesheet>
```

Angewandt auf das Beispieldokument:

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
  Alan
  Turing
```

```
23. Juni 1912
Mathematiker
Informatiker
```

```
  Judea
  Pearl
```

```
unknown
Informatiker
```


Bemerkungen:

- ❑ In diesem Beispiel enthält das Stylesheet keine matchende Template-Regel. Die Ausgabe entsteht, weil in einer solchen Situation vom XSLT-Prozessor das Built-in-Template zur Ausgabe von Text- und Attributknoten angewandt wird.
- ❑ Diejenigen Konstrukte eines XML-Dokuments, die nicht zu einem der sieben [Knotentypen](#) des XPath-Modells gehören, werden unverändert übernommen. Hierzu zählt u.a. die `<?xml ...?>`-Deklaration.
- ❑ Die Verknüpfung von XML-Dokument und XSL-Stylesheet kann explizit, in Form von Parametern für den XSLT-Prozessor, aber auch implizit geschehen: Die Zeile `<?xml-stylesheet type="text/xsl" href="..."?>` im Prolog eines XML-Dokuments deklariert ein Stylesheet. Vergleiche hierzu die [Stylesheet-Deklaration in HTML-Dokumenten](#).
- ❑ Aufruf des XSLT-Prozessors Xalan über die Kommandozeile:

```
java org.apache.xalan.xslt.Process -in personen.xml -xsl tiny.xsl
```

Hierfür muss der Ort der Xalan-Bibliothek `xalan.jar` im Classpath spezifiziert sein.
Alternativ der Aufruf mit expliziter Angabe der Xalan-Bibliothek:

```
java -cp /usr/share/java/xalan.jar ...
```

Die XSL-Familie

Elemente eines XSL-Stylesheets (Fortsetzung)

Wichtigstes Stylesheet-Element ist die Template-Regel (*Template*):

Ersetzungsmuster {

```
<xsl:template match=" " >
```

Lokalisierungspfad

```
</xsl:template>
```

- ❑ Der Lokalisierungspfad des `match`-Attributs spezifiziert – ausgehend von dem Kontextknoten – eine Knotenmenge M .
- ❑ Eine Template-Regel **matched** einen Knoten n genau dann, falls zu irgend einem Zeitpunkt der Verarbeitung n ein Element der Menge M wird.

Die XSL-Familie

Elemente eines XSL-Stylesheets (Fortsetzung)

Wichtigstes Stylesheet-Element ist die Template-Regel (*Template*):

Ersetzungsmuster {

```
<xsl:template match=" " >
```

Lokalisierungspfad

```
</xsl:template>
```

- ❑ Der Lokalisierungspfad des `match`-Attributs spezifiziert – ausgehend von dem Kontextknoten – eine Knotenmenge M .
- ❑ Eine Template-Regel **matched** einen Knoten n genau dann, falls zu irgend einem Zeitpunkt der Verarbeitung n ein Element der Menge M wird.
- ❑ Wird ein Template auf einen Knoten n angewandt, behandelt das Ersetzungsmuster den gesamten Teilbaum des XML-Dokuments, der Knoten n als Wurzel hat. **Dieser Teilbaum gilt als abgearbeitet.**

Bemerkungen:

- ❑ Der Wert des `match`-Attributes im `<xsl:template>`-Element ist ein Lokalisierungspfad in eingeschränkter XPath-Syntax.
- ❑ Um die Knotenmenge M zu spezifizieren für deren Elemente eine Template-Regel `matched`, sind alternative Pfadangaben möglich. Beispielsweise spezifizieren die Ausdrücke `match="//Elementname"` und `match="Elementname"` dieselbe Knotenmenge. D.h., ein relativer Lokalisierungspfad des `<xsl:template>`-Elements kann wie der entsprechende absolute, durch `"//"` eingeleitete Lokalisierungspfad aufgefasst werden – und umgekehrt.

Die XSL-Familie

Elemente eines XSL-Stylesheets (Fortsetzung)

Stylesheet mit literaler Ausgabe:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="https://www.w3.org/...">

  <xsl:template match="person">
    <xsl:text>Person found!</xsl:text>
  </xsl:template>

</xsl:stylesheet>
```

Angewandt auf das [Beispieldokument](#):

Die XSL-Familie

Elemente eines XSL-Stylesheets (Fortsetzung)

Stylesheet mit literaler Ausgabe:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="https://www.w3.org/...">

  <xsl:template match="person">
    <xsl:text>Person found!</xsl:text>
  </xsl:template>

</xsl:stylesheet>
```

Angewandt auf das Beispieldokument:

Person found!

Person found!

Die XSL-Familie

Elemente eines XSL-Stylesheets (Fortsetzung)

Stylesheet zum Kopieren der Elemente:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="https://www.w3.org/...">

  <xsl:template match="person">
    <xsl:copy-of select="self::*"/>
  </xsl:template>

</xsl:stylesheet>
```

Angewandt auf das [Beispieldokument](#):

Die XSL-Familie

Elemente eines XSL-Stylesheets (Fortsetzung)

Stylesheet zum Kopieren der Elemente:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="https://www.w3.org/...">

  <xsl:template match="person">
    <xsl:copy-of select="self::*"/>
  </xsl:template>

</xsl:stylesheet>
```

Angewandt auf das [Beispieldokument](#):

```
<person>
  <name>
    <vorname>Alan</vorname>
    <nachname>Turing</nachname>
  </name>
  <geburtstag>23. Juni 1912</geburtstag>
  <beruf>Mathematiker</beruf>
  <beruf>Informatiker</beruf>
</person>
...
```


Die XSL-Familie

Elemente eines XSL-Stylesheets (Fortsetzung)

Stylesheet zum Kopieren der Elementinhalte:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="https://www.w3.org/...">

  <xsl:template match="person">
    <xsl:value-of select="self::*"/>
  </xsl:template>

</xsl:stylesheet>
```

Angewandt auf das Beispieldokument:

Die XSL-Familie

Elemente eines XSL-Stylesheets (Fortsetzung)

Stylesheet zum Kopieren der Elementinhalte:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="https://www.w3.org/...">

  <xsl:template match="person">
    <xsl:value-of select="self::*"/>
  </xsl:template>

</xsl:stylesheet>
```

Angewandt auf das Beispieldokument:

Alan
Turing

23. Juni 1912
Mathematiker
Informatiker

...

Die XSL-Familie

Elemente eines XSL-Stylesheets (Fortsetzung)

Stylesheet zur Elementselektion mittels leerer Template-Regeln:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="https://www.w3.org/...">

  <xsl:template match="name">
    <xsl:value-of select="nachname"/>
    <xsl:text>, </xsl:text>
    <xsl:value-of select="vorname"/>
  </xsl:template>

  <xsl:template match="geburtstag"/>
  <xsl:template match="beruf"/>

</xsl:stylesheet>
```

Angewandt auf das [Beispieldokument](#):

Die XSL-Familie

Elemente eines XSL-Stylesheets (Fortsetzung)

Stylesheet zur Elementselektion mittels leerer Template-Regeln:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="https://www.w3.org/...">

  <xsl:template match="name">
    <xsl:value-of select="nachname"/>
    <xsl:text>, </xsl:text>
    <xsl:value-of select="vorname"/>
  </xsl:template>

  <xsl:template match="geburtstag"/>
  <xsl:template match="beruf"/>

</xsl:stylesheet>
```

Angewandt auf das [Beispieldokument](#):

Turing, Alan

Pearl, Judea

Vergleiche die Elementselektion durch [explizite Verarbeitungssteuerung](#).

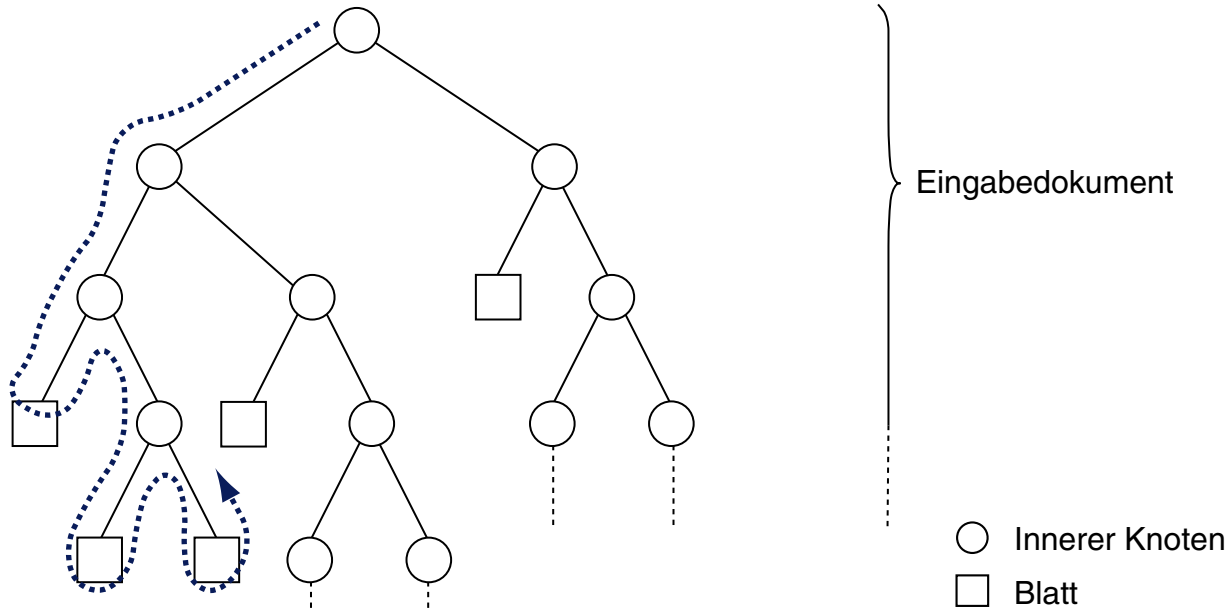
Bemerkungen (Wiederholung):

- ❑ Matched eine Template-Regel einen Knoten im XML-Dokument, so gilt der Knoten einschließlich des zugehörigen Teilbaums als abgearbeitet. Mit leeren Template-Regeln kann man Knoten und Teilbäume filtern, die nicht in der Ausgabe erscheinen sollen.
- ❑ Matched keine Template-Regel des Stylesheets einen Knoten im XML-Dokument, wird vom XSLT-Prozessor das Built-in-Template zur Ausgabe von Text- und Attributknoten angewandt.

Die XSL-Familie

XSLT-Prozessor: Verarbeitungsstrategie

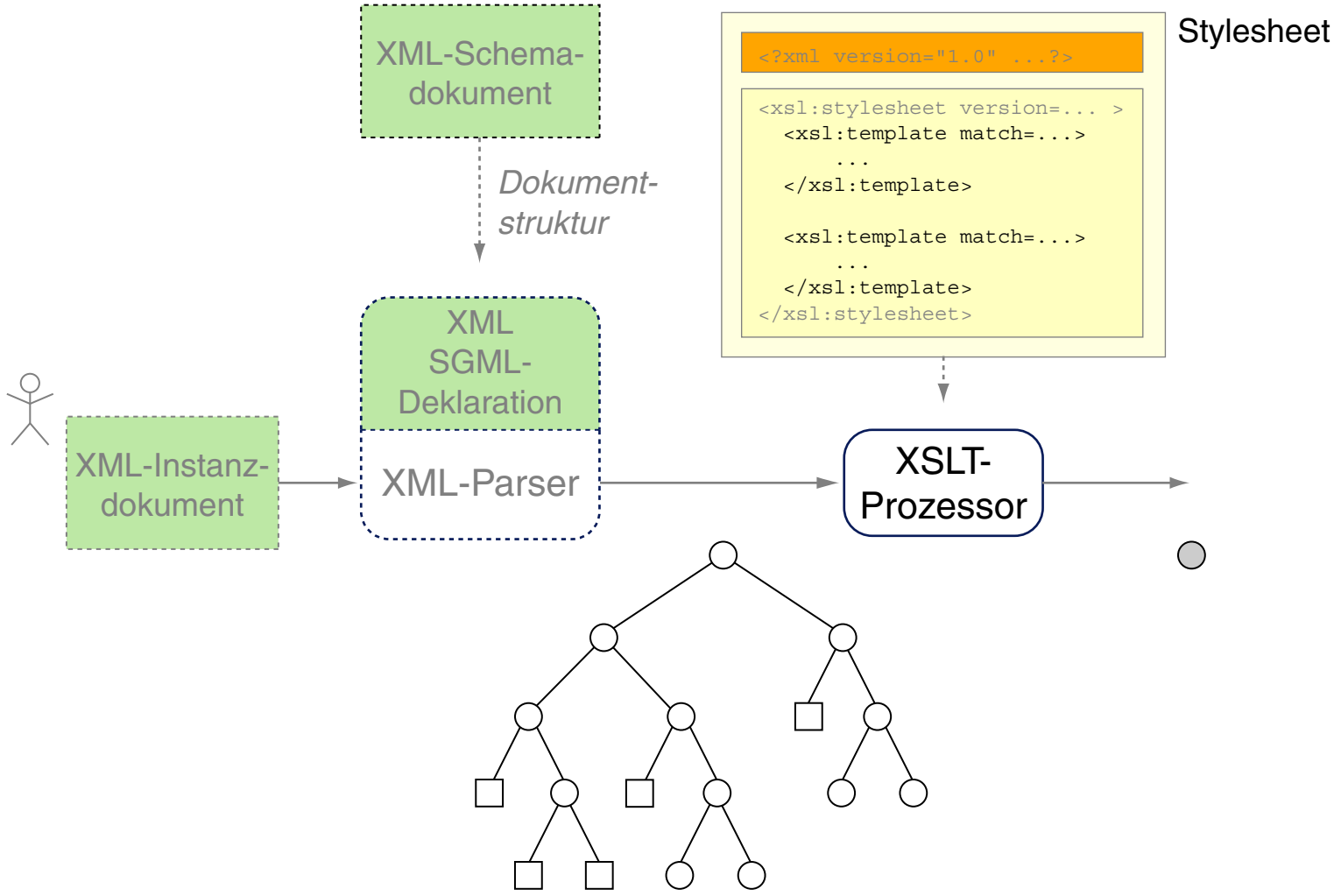
Standardmäßig durchläuft der XSLT-Prozessor den aus dem Eingabedokument erzeugten Baum ausgehend vom Wurzelknoten in Preorder-Reihenfolge.



Während des Traversierungsvorgangs wird für jeden besuchten Knoten das spezielleste, matchende Template gesucht und angewandt. So transformiert der XSLT-Prozessor einen XML-Quellbaum in einen XML-Zielbaum.

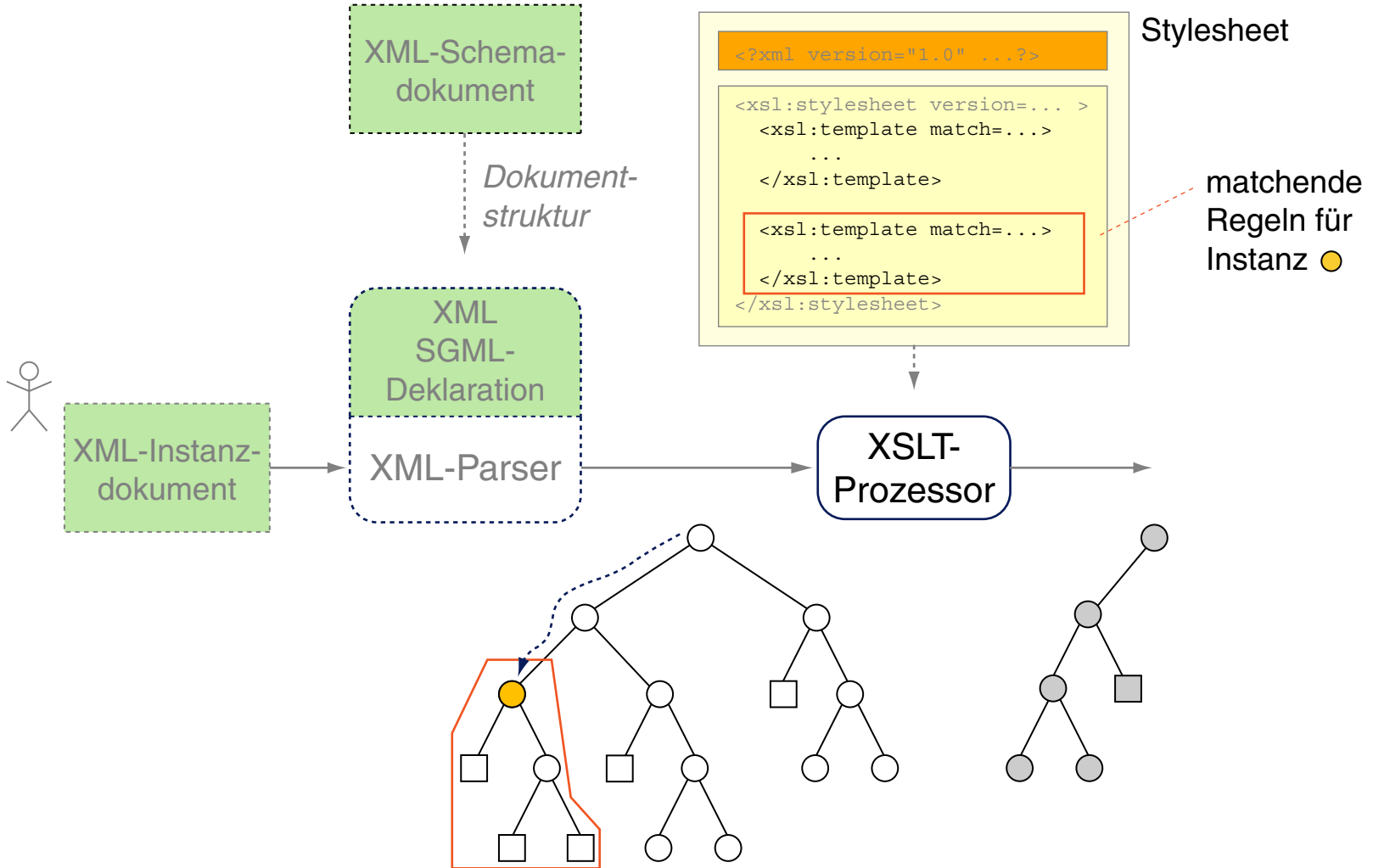
Die XSL-Familie

XSLT-Prozessor: Verarbeitungsstrategie (Fortsetzung)



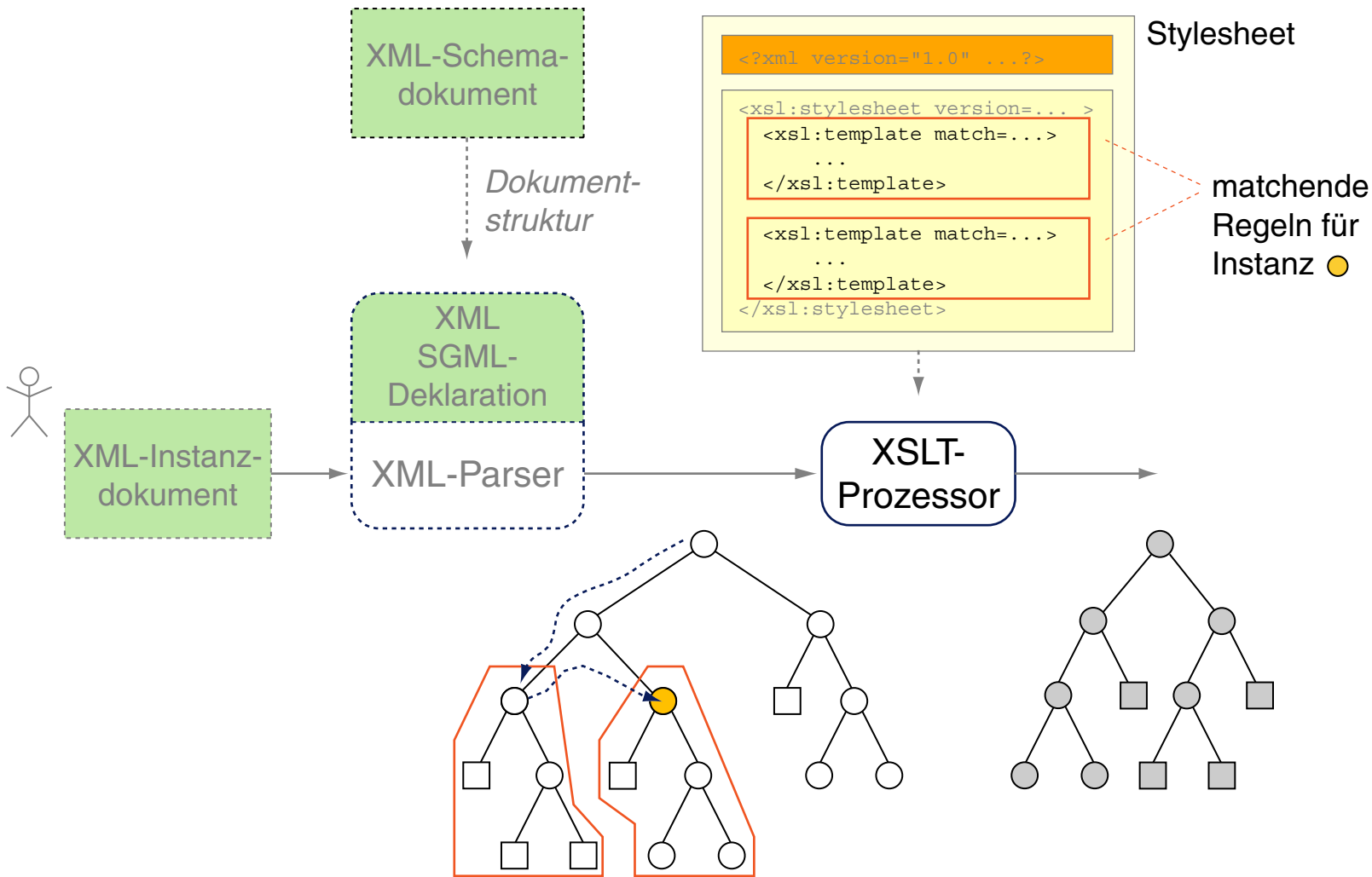
Die XSL-Familie

XSLT-Prozessor: Verarbeitungsstrategie (Fortsetzung)



Die XSL-Familie

XSLT-Prozessor: Verarbeitungsstrategie (Fortsetzung) [WT:III CSS-Verarbeitung]



Bemerkungen:

- ❑ Aus Verarbeitungssicht spielt somit die Reihenfolge der Template-Regeln in einem XSL-Stylesheet keine Rolle: die Verarbeitung wird ausschließlich durch die *Reihenfolge der Elemente im Eingabedokument* bestimmt.
- ❑ Ein Anwendungskonflikt liegt vor, wenn Lokalisierungspfade von verschiedenen Template-Regeln t_1, t_2 einen Knoten n in ihrer spezifizierten Knotenmengen M_{t_1}, M_{t_2} enthalten. In diesem Fall kommt das Template $t_x, x \in \{1, 2\}$, mit dem speziellsten Pfad im `match`-Attribut zur Anwendung: $|M_{t_x}| \leq \min\{|M_{t_1}|, |M_{t_2}|\}$

Die XSL-Familie

XSLT-Prozessor: Verarbeitungsstrategie (Fortsetzung)

Stylesheet zur Elementselektion mit expliziter Verarbeitungssteuerung:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="https://www.w3.org/...">

  <xsl:template match="person">
    <xsl:apply-templates select="name"/>
  </xsl:template>

  <xsl:template match="name">
    <xsl:value-of select="nachname"/>
    <xsl:text>, </xsl:text>
    <xsl:value-of select="vorname"/>
  </xsl:template>

</xsl:stylesheet>
```

Angewandt auf das [Beispieldokument](#):

Die XSL-Familie

XSLT-Prozessor: Verarbeitungsstrategie (Fortsetzung)

Stylesheet zur Elementselektion mit expliziter Verarbeitungssteuerung:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="https://www.w3.org/...">

  <xsl:template match="person">
    <xsl:apply-templates select="name"/>
  </xsl:template>

  <xsl:template match="name">
    <xsl:value-of select="nachname"/>
    <xsl:text>, </xsl:text>
    <xsl:value-of select="vorname"/>
  </xsl:template>

</xsl:stylesheet>
```

Angewandt auf das [Beispieldokument](#):

```
Turing, Alan
Pearl, Judea
```

Vergleiche die Elementselektion mittels [leerer Template-Regeln](#).

Bemerkungen:

- ❑ Das `<xsl:apply-templates>`-Element startet für die mit dem `select`-Attribut spezifizierte Knotenmenge erneut einen Preorder-Durchlauf zur Anwendung der Template-Regeln des Stylesheets.
- ❑ Der Wert des `select`-Attributes im `<xsl:apply-templates>`-Element ist ein Lokalisierungspfad in eingeschränkter XPath-Syntax. Weil sich so beliebige Knoten im Dokument spezifizieren lassen, ermöglicht das `<xsl:apply-templates>`-Element die mehrmalige Verarbeitung von Knoten, also auch die Erzeugung von Endlosschleifen.
- ❑ Falls keine andere Achse angegeben ist, setzt der Lokalisierungspfad des `<xsl:apply-templates>`-Elements den Pfad des matchenden Knoten fort. Das heißt, die Ausdrücke `select="./Elementname"` und `select="Elementname"` spezifizieren dieselbe Knotenmenge.
- ❑ Enthält das `<xsl:apply-templates>`-Element kein `select`-Attribut, so gelten per Default die Kindknoten (`child::`-Achse) des matchenden Knoten als spezifiziert.

Die XSL-Familie

XSLT-Prozessor: Verarbeitungsstrategie (Fortsetzung)

Algorithm: `xsl:apply-templates`

Input: `select`. XPath expression or empty string.

n_c . Context node.

T . XSL stylesheet with templates.

Output: –

```
xsl:apply-templates(select,  $n_c$ ,  $T$ )
```

1. `nodes = evalXPath(select, n_c)`
2. LOOP
3. IF `nodes = \emptyset` THEN RETURN
4. `n = pop(nodes)`
5. `t = mostSpecificTemplate(T , n)`
6. IF `t \neq Null`
THEN `executeTemplate(t , n)`
ELSE `executeBuiltinTemplate(n)`
7. ENDLIST

Bemerkungen:

- ❑ Die Funktionen *executeTemplate*(t, n) und *executeBuiltInTemplate*(n) wenden das Ersetzungsmuster eines `<xsl:template>`-Elements auf den Knoten n an.
- ❑ Der Preorder-Durchlauf entsteht durch den rekursiven Aufruf von `xsl:apply-templates()` in Schritt 6 – entweder durch benutzerdefinierte `<xsl:apply-templates>`-Elemente in t oder durch Anwendung eines [Built-in-Templates](#).
- ❑ Der XSLT-Prozessor verwaltet intern das XML Information Set des zu verarbeitenden XML-Dokuments und stellt der Funktion `xsl:apply-templates()` den Kontextknoten n_c und das Stylesheet T zur Verfügung.

Die XSL-Familie

XSLT-Prozessor: Beispiele zur Verarbeitungsstrategie

Stylesheet mit zweifacher Verarbeitung der `<name>`-Kindelemente:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="https://www.w3.org/...">

  <xsl:template match="person">
    <xsl:apply-templates select="name"/>
    <xsl:apply-templates select="name"/>
  </xsl:template>

  <xsl:template match="name">
    <xsl:value-of select="nachname"/>
    <xsl:text>, </xsl:text>
    <xsl:value-of select="vorname"/>
  </xsl:template>

</xsl:stylesheet>
```

Angewandt auf das [Beispieldokument](#):

Die XSL-Familie

XSLT-Prozessor: Beispiele zur Verarbeitungsstrategie

Stylesheet mit zweifacher Verarbeitung der `<name>`-Kindelemente:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="https://www.w3.org/...">

  <xsl:template match="person">
    <xsl:apply-templates select="name"/>
    <xsl:apply-templates select="name"/>
  </xsl:template>

  <xsl:template match="name">
    <xsl:value-of select="nachname"/>
    <xsl:text>, </xsl:text>
    <xsl:value-of select="vorname"/>
  </xsl:template>
</xsl:stylesheet>
```

Angewandt auf das [Beispieldokument](#):

Turing, AlanTuring, Alan

Pearl, JudeaPearl, Judea

Die XSL-Familie

XSLT-Prozessor: Beispiele zur Verarbeitungsstrategie (Fortsetzung)

Stylesheet zur **wiederholten** Verarbeitung **aller** <name>-Elemente:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="https://www.w3.org/...">

  <xsl:template match="person">
    <xsl:apply-templates select="//name"/>
  </xsl:template>

  <xsl:template match="name">
    <xsl:value-of select="nachname"/>
    <xsl:text>, </xsl:text>
    <xsl:value-of select="vorname"/>
  </xsl:template>

</xsl:stylesheet>
```

Angewandt auf das Beispieldokument:

Die XSL-Familie

XSLT-Prozessor: Beispiele zur Verarbeitungsstrategie (Fortsetzung)

Stylesheet zur **wiederholten** Verarbeitung **aller** <name>-Elemente:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="https://www.w3.org/...">

  <xsl:template match="person">
    <xsl:apply-templates select="//name"/>
  </xsl:template>

  <xsl:template match="name">
    <xsl:value-of select="nachname"/>
    <xsl:text>, </xsl:text>
    <xsl:value-of select="vorname"/>
  </xsl:template>

</xsl:stylesheet>
```

Angewandt auf das Beispieldokument:

Turing, AlanPearl, Judea

Turing, AlanPearl, Judea

Die XSL-Familie

XSLT-Prozessor: Beispiele zur Verarbeitungsstrategie (Fortsetzung)

Stylesheet, dessen matchende Template-Regel die leere Knotenmenge liefert:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="https://www.w3.org/...">

  <xsl:template match="person">
    <xsl:apply-templates select="nachname"/>
  </xsl:template>

  <xsl:template match="name">
    <xsl:value-of select="nachname"/>
    <xsl:text>, </xsl:text>
    <xsl:value-of select="vorname"/>
  </xsl:template>

</xsl:stylesheet>
```

Angewandt auf das [Beispieldokument](#):

Die XSL-Familie

XSLT-Prozessor: Beispiele zur Verarbeitungsstrategie (Fortsetzung)

Stylesheet, dessen Verarbeitung in eine Endlosschleife führt:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="https://www.w3.org/...">

  <xsl:template match="person">
    <xsl:apply-templates select="/personen/person"/>
  </xsl:template>

  <xsl:template match="name">
    <xsl:value-of select="nachname"/>
    <xsl:text>, </xsl:text>
    <xsl:value-of select="vorname"/>
  </xsl:template>

</xsl:stylesheet>
```

Angewandt auf das [Beispieldokument](#):

Die XSL-Familie

XSLT-Prozessor: Beispiele zur Verarbeitungsstrategie (Fortsetzung)

Stylesheet, dessen Verarbeitung in eine Endlosschleife führt:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="https://www.w3.org/...">

  <xsl:template match="person">
    <xsl:apply-templates select="/personen/person"/>
  </xsl:template>

  <xsl:template match="name">
    <xsl:value-of select="nachname"/>
    <xsl:text>, </xsl:text>
    <xsl:value-of select="vorname"/>
  </xsl:template>

</xsl:stylesheet>
```

Angewandt auf das Beispieldokument:

```
(Location of error unknown)XSLT Error (java.lang.StackOverflowError) :
null
```

Die XSL-Familie

XSLT-Prozessor: Built-in-Templates

1. Built-in-Template, das die rekursive Verarbeitung garantiert, falls kein matchendes Template im Stylesheet existiert:

```
<xsl:template match="*|/">
  <xsl:apply-templates/>
</xsl:template>
```

2. Built-in-Template zur Ausgabe von Text- und Attributknoten:

```
<xsl:template match="text()|@*">
  <xsl:value-of select="."/>
</xsl:template>
```

3. Built-in-Template, das die Kommentare matched und ignoriert:

```
<xsl:template match="processing-instruction()|comment()"/>
```

Vergleiche die Elementselektion mittels leerer Template-Regeln.

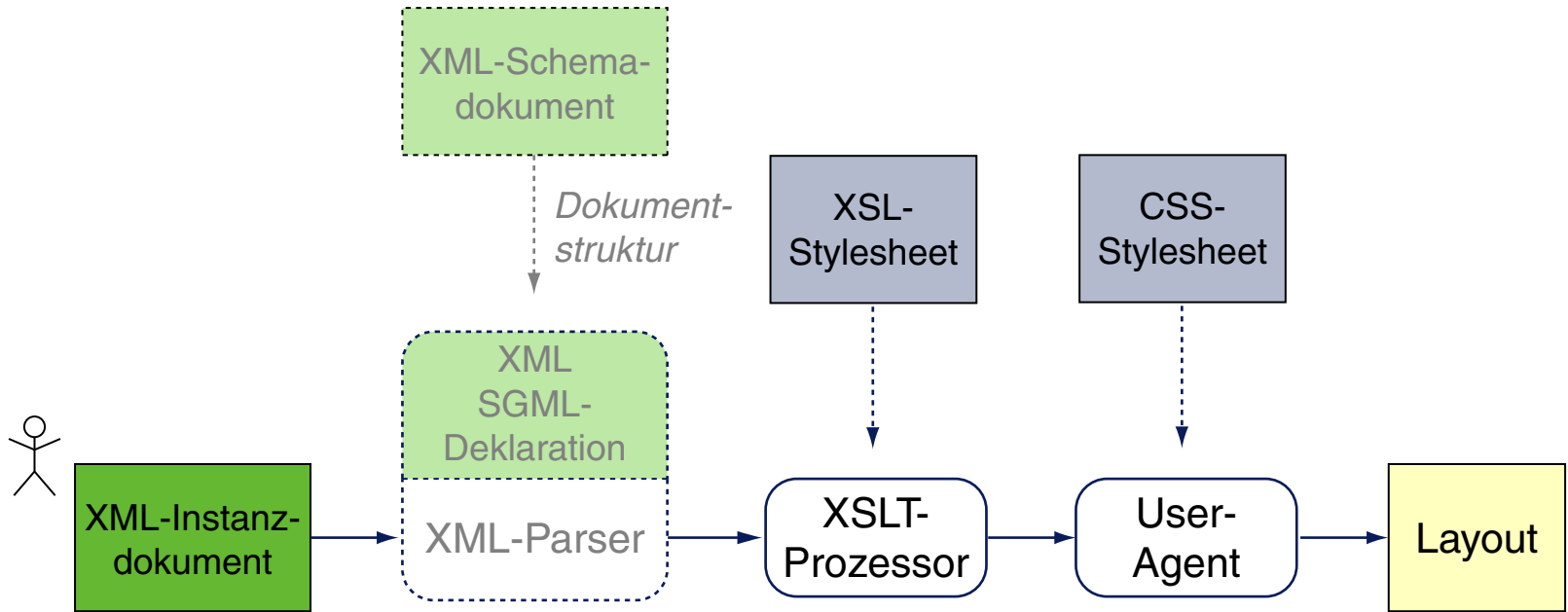
Die XSL-Familie

Weitere XSLT-Konzepte

- ❑ Template-Modi zur Charakterisierung von Verarbeitungsphasen
- ❑ benannte Templates zur Realisierung direkter Aufrufe
- ❑ Nummerierung und Sortierung von Ausgabeelementen
- ❑ bedingte Verarbeitung und Schleifen
- ❑ Import anderer Stylesheets

Die XSL-Familie

XML-Dokumentenverarbeitung: Erzeugung von HTML-Dokumenten



Vergleiche hierzu den Standardprozess der XSL Transformation.

Die XSL-Familie

Erzeugung von HTML-Dokumenten: Zusammenspiel mit CSS

```
<?xml version="1.0" standalone="no" ?>
<?xml-stylesheet type="text/xsl" href="personen2html.xsl"?>

<personen>
  <person>
    <name>
      <vorname>Alan</vorname>
      <nachname>Turing</nachname>
    </name>
    <geburtstag>23. Juni 1912</geburtstag>
    <beruf>Mathematiker</beruf>
    <beruf>Informatiker</beruf>
  </person>

  <person>
    <name>
      <vorname>Judea</vorname>
      <nachname>Pearl</nachname>
    </name>
    <geburtstag>unknown</geburtstag>
    <beruf>Informatiker</beruf>
  </person>
</personen>
```

Die XSL-Familie

Erzeugung von HTML-Dokumenten: Zusammenspiel mit CSS (Fortsetzung)

```
<xsl:template match="personen">
  <html>
    <head>
      <title>
        <xsl:text>Personen</xsl:text>
      </title>
    </head>
    <body>
      <xsl:apply-templates/>
    </body>
  </html>
</xsl:template>
```

Die XSL-Familie

Erzeugung von HTML-Dokumenten: Zusammenspiel mit CSS (Fortsetzung)

```
<xsl:template match="personen">
  <html>
    <head>
      <title>
        <xsl:text>Personen</xsl:text>
      </title>
    </head>
    <body>
      <xsl:apply-templates/>
    </body>
  </html>
</xsl:template>
```

Die XSL-Familie

Erzeugung von HTML-Dokumenten: Zusammenspiel mit CSS (Fortsetzung)

```
<xsl:template match="personen">
  <html>
    <head>
      <title>
        <xsl:text>Personen</xsl:text>
      </title>
    </head>
    <body>
      <xsl:apply-templates/>
    </body>
  </html>
</xsl:template>
```

Die XSL-Familie

Erzeugung von HTML-Dokumenten: Zusammenspiel mit CSS (Fortsetzung)

```
<xsl:template match="personen">
  <html>
    <head>
      <title>
        <xsl:text>Personen</xsl:text>
      </title>
    </head>
    <body>
      <xsl:apply-templates/>
    </body>
  </html>
</xsl:template>

<xsl:template match="name">
  <p/>
  <div>
    <xsl:text>Name: </xsl:text>
    <span style="font-weight:bold">
      <xsl:value-of select="self::*"/>
    </span>
  </div>
</xsl:template>

...
```

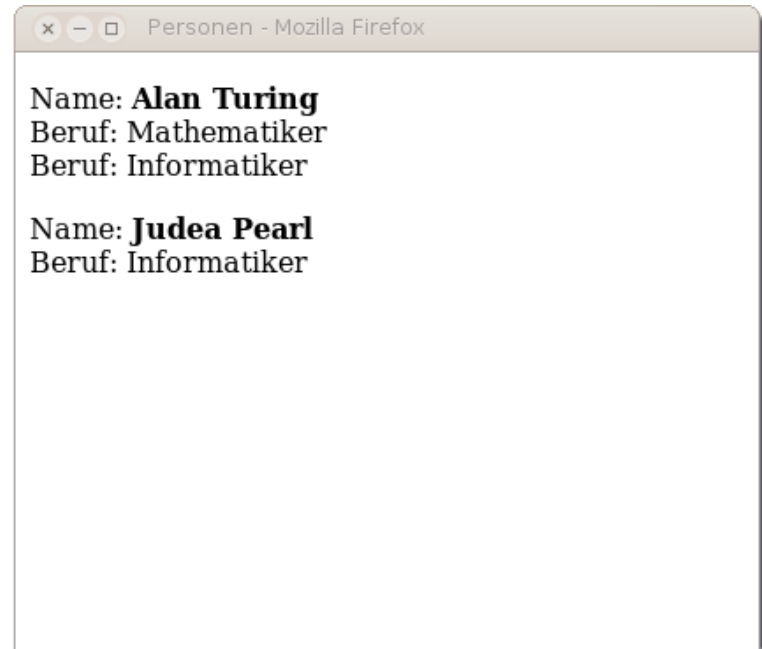
Die XSL-Familie

Erzeugung von HTML-Dokumenten: Zusammenspiel mit CSS (Fortsetzung)

```
<xsl:template match="personen">
  <html>
    <head>
      <title>
        <xsl:text>Personen</xsl:text>
      </title>
    </head>
    <body>
      <xsl:apply-templates/>
    </body>
  </html>
</xsl:template>

<xsl:template match="name">
  <p/>
  <div>
    <xsl:text>Name: </xsl:text>
    <span style="font-weight:bold">
      <xsl:value-of select="self::*"/>
    </span>
  </div>
</xsl:template>

...
```



Die XSL-Familie

Erzeugung von HTML-Dokumenten: Zusammenspiel mit CSS (Fortsetzung)

```
<xsl:template match="personen">
  <html>
    <head>
      <title>
        <xsl:text>Personen</xsl:text>
      </title>
      <link rel="stylesheet" type="text/css" href="personen.css"/>
    </head>
    <body>
      <xsl:apply-templates/>
    </body>
  </html>
</xsl:template>
```

```
<xsl:template match="name">
  <p/>
  <div>
    <xsl:text>Name: </xsl:text>
    <xsl:value-of select="self::*"/>
  </div>
</xsl:template>
```

...

Bemerkungen:

- Eine Anwendung nach diesem Prinzip sind die FAQs des W3C:
Aus der XML-Source [faq.xml](#) gemäß der DTD [faq.dtd](#) wird mittels des Stylesheets [faqxsl.xsl](#) das HTML-Dokument [faq.html](#) erzeugt.

Weil in [faq.xml](#) das Stylesheet [faq.css](#) verlinkt ist, zeigt der Browser nicht den XML-Dokumentenbaum an:

```
<?xml version="1.0"?>
<!DOCTYPE faq SYSTEM "faq.dtd">
<?xml-stylesheet href="faq.css" type="text/css"?>
<faq>
  <head>
    <title>Document Object Model FAQ</title>
    ...
```

Die XSL-Familie

Erzeugung von HTML-Dokumenten: Datenaufbereitung

CD-Datenbank als XML-Beispieldokument [\[w3schools\]](#) :

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="cdcatalog.xsl"?>

<catalog>
  <cd>
    <title>Empire Burlesque</title>
    <artist>Bob Dylan</artist>
    <company>Columbia</company>
    <price>10.90</price>
    <year>1985</year>
  </cd>

  ...

  <cd>
    <title>Unchain my heart</title>
    <artist>Joe Cocker</artist>
    <company>EMI</company>
    <price>8.20</price>
    <year>1987</year>
  </cd>
</catalog>
```

Die XSL-Familie

Erzeugung von HTML-Dokumenten: Datenaufbereitung (Fortsetzung)

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="https://www.w3.org/...">
<xsl:template match="/">
  <html>
  <body>
    <h2>My CD Collection</h2>
    <table border="1">
      <tr bgcolor="#9acd32">
        <th>Title</th>
        <th>Artist</th>
      </tr>
      <xsl:for-each select="catalog/cd">
        <tr>
          <td><xsl:value-of select="title"/></td>
          <td><xsl:value-of select="artist"/></td>
        </tr>
      </xsl:for-each>
    </table>
  </body>
</html>
</xsl:template>
</xsl:stylesheet>
```

Die XSL-Familie

Erzeugung von HTML-Dokumenten: Datenaufbereitung (Fortsetzung)

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="https://www.w3.org/...">
<xsl:template match="/">
  <html>
  <body>
    <h2>My CD Collection</h2>
    <table border="1">
      <tr bgcolor="#9acd32">
        <th>Title</th>
        <th>Artist</th>
      </tr>
      <xsl:for-each select="catalog/...">
        <tr>
          <td><xsl:value-of select="ti...">
          <td><xsl:value-of select="ar...">
        </tr>
      </xsl:for-each>
    </table>
  </body>
</html>
</xsl:template>
</xsl:stylesheet>
```

[w3schools [xml](#), [xsl](#)]



Title	Artist
Empire Burlesque	Bob Dylan
Hide your heart	Bonnie Tyler
Greatest Hits	Dolly Parton
Still got the blues	Gary Moore
Eros	Eros Ramazzotti
One night only	Bee Gees
Sylvias Mother	Dr.Hook
Maggie May	Rod Stewart
Romanza	Andrea Bocelli
When a man loves a woman	Percy Sledge
Black angel	Savage Rose
1999 Grammy Nominees	Many
For the good times	Kenny Rogers

Die XSL-Familie

Erzeugung von HTML-Dokumenten: Datenaufbereitung (Fortsetzung)

Filtern mit XPath:

```
<xsl:template match="/">
  <html>
  <body>
    <h2>My CD Collection</h2>
    <table border="1">
      <tr bgcolor="#9acd32">
        <th>Title</th>
        <th>Artist</th>
      </tr>
      <xsl:for-each select="catalog/cd[artist='Bob Dylan']">
        <tr>
          <td><xsl:value-of select="title"/></td>
          <td><xsl:value-of select="artist"/></td>
        </tr>
      </xsl:for-each>
    </table>
  </body>
</html>
</xsl:template>
```

Die XSL-Familie

Erzeugung von HTML-Dokumenten: Datenaufbereitung (Fortsetzung)

Filtern mit XPath:

```
<xsl:template match="/">
  <html>
  <body>
    <h2>My CD Collection</h2>
    <table border="1">
      <tr bgcolor="#9acd32">
        <th>Title</th>
        <th>Artist</th>
      </tr>
      <xsl:for-each select="catalog/cd[artist='Bob Dylan']">
        <tr>
          <td><xsl:value-of select="ti
          <td><xsl:value-of select="ar
        </tr>
      </xsl:for-each>
    </table>
  </body>
</html>
</xsl:template>
```



[w3schools [xml](#), [xsl](#)]

Die XSL-Familie

Erzeugung von HTML-Dokumenten: Datenaufbereitung (Fortsetzung)

Sortieren:

```
<xsl:template match="/">
  <html>
  <body>
    <h2>My CD Collection</h2>
    <table border="1">
      <tr bgcolor="#9acd32">
        <th>Title</th>
        <th>Artist</th>
      </tr>
      <xsl:for-each select="catalog/cd">
        <xsl:sort select="artist"/>
        <tr>
          <td><xsl:value-of select="title"/></td>
          <td><xsl:value-of select="artist"/></td>
        </tr>
      </xsl:for-each>
    </table>
  </body>
</html>
</xsl:template>
```


Die XSL-Familie

Erzeugung von HTML-Dokumenten: Datenaufbereitung (Fortsetzung)

Sortieren:

```
<xsl:template match="/">
  <html>
  <body>
    <h2>My CD Collection</h2>
    <table border="1">
      <tr bgcolor="#9acd32">
        <th>Title</th>
        <th>Artist</th>
      </tr>
      <xsl:for-each select="catalog/">
        <xsl:sort select="artist"/>
        <tr>
          <td><xsl:value-of select="ti
          <td><xsl:value-of select="ar
        </tr>
      </xsl:for-each>
    </table>
  </body>
</html>
</xsl:template>
```



Title	Artist
Romanza	Andrea Bocelli
One night only	Bee Gees
Empire Burlesque	Bob Dylan
Hide your heart	Bonnie Tyler
The very best of	Cat Stevens
Greatest Hits	Dolly Parton
Sylvias Mother	Dr.Hook
Eros	Eros Ramazzotti
Still got the blues	Gary Moore
Unchain my heart	Joe Cocker
Soulsville	Jorn Hoel
For the good times	Kenny Rogers
Midt om natten	Kim Larsen

Die XSL-Familie

Erzeugung von HTML-Dokumenten: Datenaufbereitung (Fortsetzung)

Verwenden von Bedingungen:

```
<xsl:template match="/">
  <html>
  <body>
    <h2>My CD Collection</h2>
    <table border="1">
      <tr bgcolor="#9acd32">
        <th>Title</th>
        <th>Artist</th>
      </tr>
      <xsl:for-each select="catalog/cd">
        <xsl:if test="price > 10">
          <tr>
            <td><xsl:value-of select="title"/></td>
            <td><xsl:value-of select="artist"/></td>
          </tr>
        </xsl:if>
      </xsl:for-each>
    </table>
  </body>
</html>
</xsl:template>
```

Die XSL-Familie

Erzeugung von HTML-Dokumenten: Datenaufbereitung (Fortsetzung)

Verwenden von Bedingungen:

```
<xsl:template match="/">
  <html>
  <body>
    <h2>My CD Collection</h2>
    <table border="1">
      <tr bgcolor="#9acd32">
        <th>Title</th>
        <th>Artist</th>
      </tr>
      <xsl:for-each select="catalog/album">
        <xsl:if test="price > 10">
          <tr>
            <td><xsl:value-of select="title"/></td>
            <td><xsl:value-of select="artist"/></td>
          </tr>
        </xsl:if>
      </xsl:for-each>
    </table>
  </body>
</html>
</xsl:template>
```



Title	Artist
Empire Burlesque	Bob Dylan
Still got the blues	Gary Moore
One night only	Bee Gees
Romanza	Andrea Bocelli
Black angel	Savage Rose
1999 Grammy Nominees	Many

Die XSL-Familie

Erzeugung von HTML-Dokumenten: Datenaufbereitung (Fortsetzung)

Verwenden von Bedingungen:

```
<xsl:template match="/">
  <html>
  <body>
    <h2>My CD Collection</h2>
    <table border="1">
      ...
      <xsl:for-each select="catalog/cd">
        <tr>
          <td><xsl:value-of select="title"/></td>
          <xsl:choose>
            <xsl:when test="price > 10">
              <td bgcolor="#ff00ff"><xsl:value-of select="artist"/></td>
            </xsl:when>
            <xsl:otherwise>
              <td><xsl:value-of select="artist"/></td>
            </xsl:otherwise>
          </xsl:choose>
        </tr>
      </xsl:for-each>
    </table>
    ...
  </body>
</template>
```

Die XSL-Familie

Erzeugung von HTML-Dokumenten: Datenaufbereitung (Fortsetzung)

Verwenden von Bedingungen:

```
<xsl:template match="/">
  <html>
  <body>
    <h2>My CD Collection</h2>
    <table border="1">
      ...
      <xsl:for-each select="catalog/">
      <tr>
        <td><xsl:value-of select="ti
          <xsl:choose>
            <xsl:when test="price >
              <td bgcolor="#ff00ff"><xs
            </xsl:when>
            <xsl:otherwise>
              <td><xsl:value-of select=
            </xsl:otherwise>
          </xsl:choose>
        </td>
      </tr>
    </xsl:for-each>
  </table>
  ...
```



Title	Artist
Empire Burlesque	Bob Dylan
Hide your heart	Bonnie Tyler
Greatest Hits	Dolly Parton
Still got the blues	Gary Moore
Eros	Eros Ramazzotti
One night only	Bee Gees
Sylvias Mother	Dr. Hook
Maggie May	Rod Stewart
Romanza	Andrea Bocelli
When a man loves a woman	Percy Sledge
Black angel	Savage Rose
1999 Grammy Nominees	Many
For the good times	Kenny Rogers

Die XSL-Familie

XML-Dokumentenverarbeitung: Elementinhalte anpassen [WT:III DOM-API]

```
<?xml version="1.0" standalone="no" ?>
<?xml-stylesheet type="text/xsl" href="personen2html.xsl"?>

<personen>
  <person>
    <name>
      <vorname>Alan</vorname>
      <nachname>Turing</nachname>
    </name>
    <geburtstag>23. Juni 1912</geburtstag>
    <beruf>Mathematiker</beruf>
    <beruf>Informatiker</beruf>
  </person>

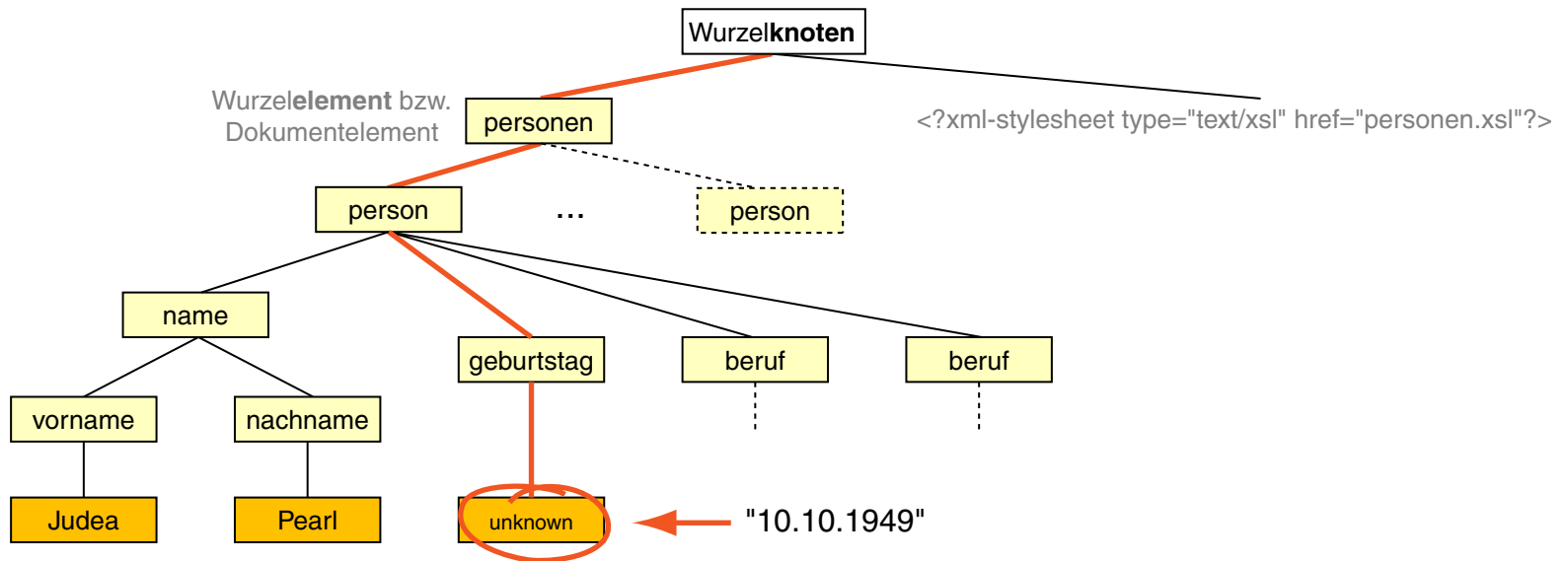
  <person>
    <name>
      <vorname>Judea</vorname>
      <nachname>Pearl</nachname>
    </name>
    <geburtstag>unknown</geburtstag>
    <beruf>Informatiker</beruf>
  </person>
</personen>
```

Die XSL-Familie

XML-Dokumentenverarbeitung: Elementinhalte anpassen (Fortsetzung)

Aufgabe:

1. Die Person „Judea Pearl“ finden.
2. Seinen Geburtstag auf einen bestimmten Wert setzen.



Die XSL-Familie

XML-Dokumentenverarbeitung: Elementinhalte anpassen (Fortsetzung)

Stylesheet:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="https://www.w3.org/...">

  <xsl:template match="@*|node()"> \[W3C\]
    <xsl:copy><xsl:apply-templates select="@*|node()" /></xsl:copy>
  </xsl:template>

</xsl:stylesheet>
```


Die XSL-Familie

XML-Dokumentenverarbeitung: Elementinhalte anpassen (Fortsetzung)

Stylesheet:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="https://www.w3.org/...">

  <xsl:template match="@*|node()"> \[W3C\]
    <xsl:copy><xsl:apply-templates select="@*|node()" /></xsl:copy>
  </xsl:template>

  <xsl:template match="person[name/nachname='Pearl' and
    name/vorname='Judea']/geburtstag/text()">
    <xsl:text>10.10.1949</xsl:text>
  </xsl:template>
</xsl:stylesheet>
```

Die XSL-Familie

XML-Dokumentenverarbeitung: Elementinhalte anpassen (Fortsetzung)

Stylesheet:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="https://www.w3.org/...">

  <xsl:template match="@*|node()"> \[W3C\]
    <xsl:copy><xsl:apply-templates select="@*|node()" /></xsl:copy>
  </xsl:template>

  <xsl:template match="person[name/nachname='Pearl' and
    name/vorname='Judea']/geburtstag/text()">
    <xsl:text>10.10.1949</xsl:text>
  </xsl:template>

</xsl:stylesheet>
```

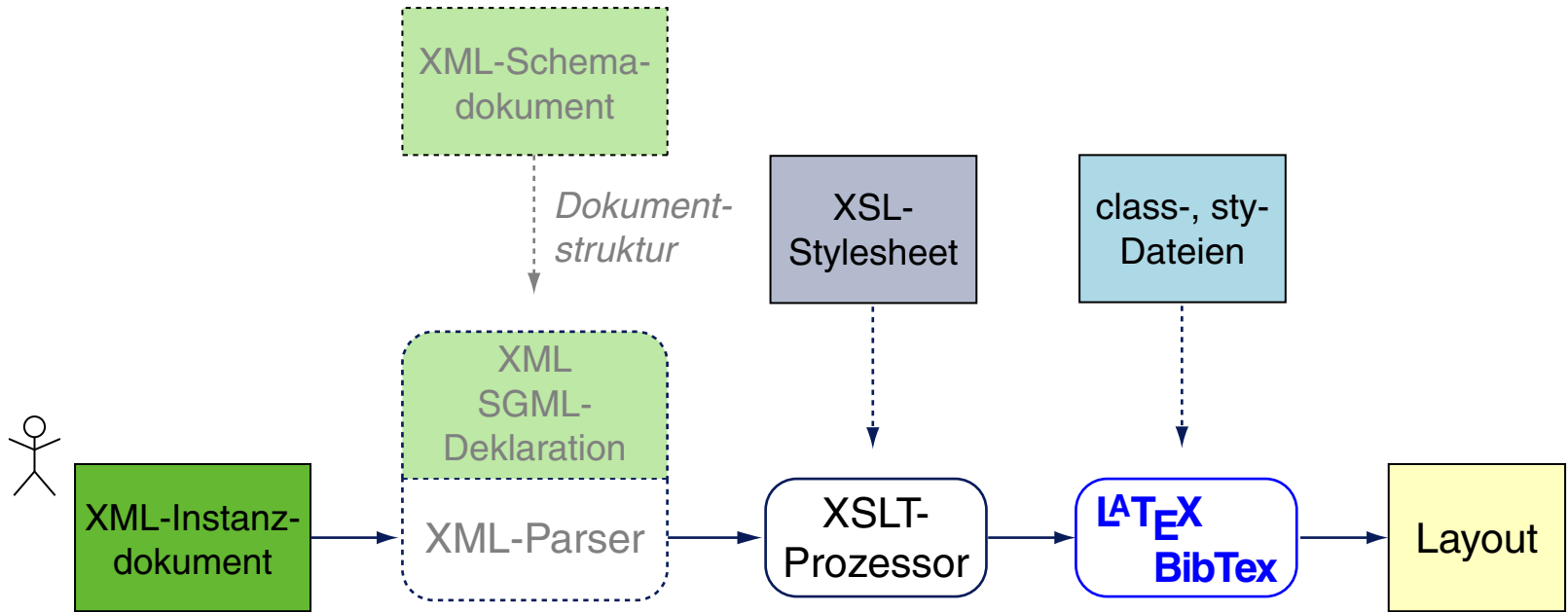
Angewandt auf das Beispieldokument:

```
...
<name>
  <vorname>Judea</vorname>
  <nachname>Pearl</nachname>
</name>
<geburtstag>10.10.1949</geburtstag>
```

...

Die XSL-Familie

XML-Dokumentenverarbeitung: Prozesskette für Printmedien



Vergleiche hierzu

- ❑ den Standardprozess der XSL Transformation
- ❑ und die HTML-Prozesskette.

Die XSL-Familie

Prozesskette für Printmedien: Erzeugung von Latex-Dokumenten

```
<?xml version="1.0" standalone="no" ?>  
<?xml-stylesheet type="text/xsl" href="xml2latex.xsl"?>  
  
<book>  
  
<section>  
  <title>Eine Überschrift</title>  
  
  Hier ist der Fließtext ...  
</section>  
  
</book>
```

Die XSL-Familie

Prozesskette für Printmedien: Erzeugung von Latex-Dokumenten (Fortsetzung)

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="https://www.w3.org/...">
<xsl:template match="/">
  \documentclass{article}
  \usepackage[T1]{fontenc}
  \usepackage[english,german]{babel}

  \begin{document}
  <xsl:apply-templates/>
  \end{document}
</xsl:template>

<xsl:template match="section">
  <xsl:apply-templates/>
</xsl:template>

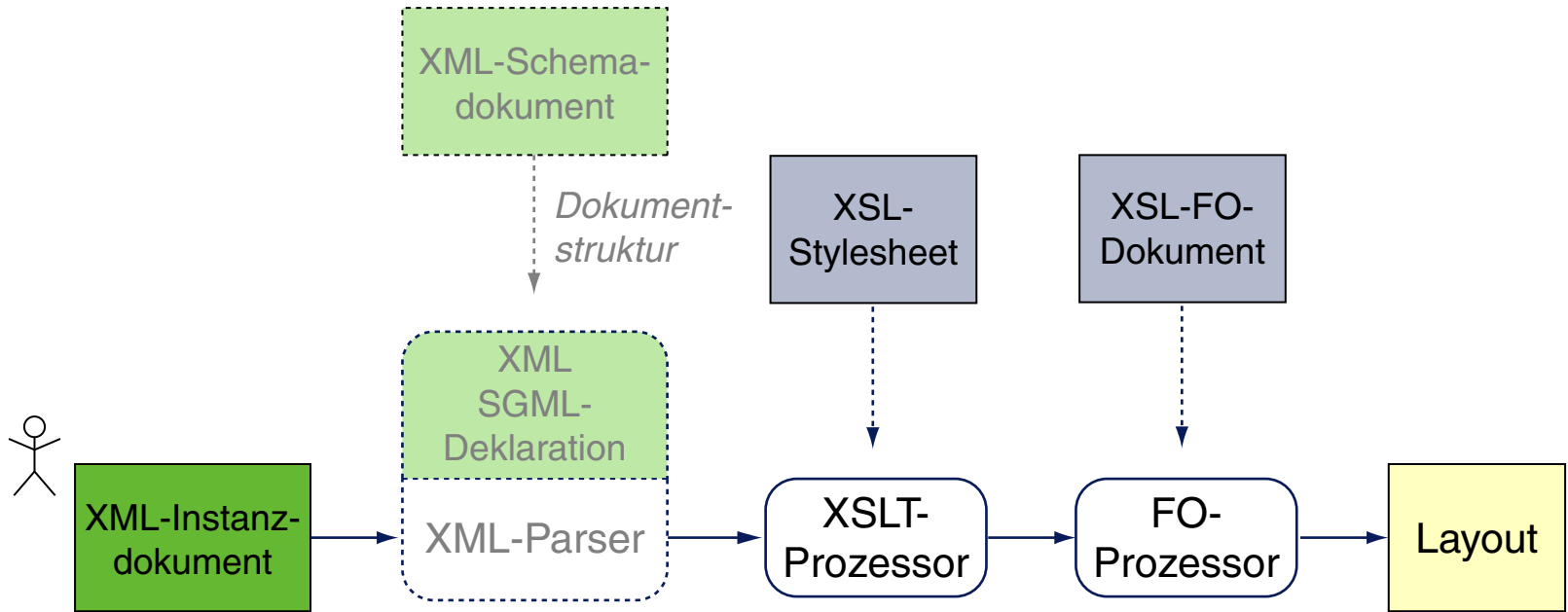
<xsl:template match="title">
  \section{<xsl:value-of select="self::*"/>}
</xsl:template>

...

</xsl:stylesheet>
```

Die XSL-Familie

XML-Dokumentenverarbeitung: Erzeugung beliebiger Formate mit XSL-FO



Vergleiche hierzu

- ❑ den Standardprozess der XSL Transformation,
- ❑ die HMTL-Prozesskette
- ❑ und die Latex-Prozesskette.

Die XSL-Familie

Quellen zum Nachlernen und Nachschlagen im Web: Referenz

- ❑ W3C. *XSL Transformations (XSLT) 3.0*.
www.w3.org/TR/xslt-30
- ❑ W3C *XML Path Language (XPath) 3.0*.
www.w3.org/TR/xpath-30
- ❑ W3C *XML Query Language (XQuery) 3.0*.
www.w3.org/TR/xquery-30
- ❑ W3C *XSL Formatting Objects (XSL-FO) 2.0*.
www.w3.org/standards/techs/xsl

Die XSL-Familie

Quellen zum Nachlernen und Nachschlagen im Web: Usage

- ❑ Nic/Jirat. *XPath Tutorial*.
www.zvon.org
- ❑ Cover Pages. *Extensible Stylesheet Language*.
xml.coverpages.org/xsl.html
- ❑ W3 Schools. *XSLT*.
www.w3schools.com/xsl
- ❑ Apache. *Xalan Project*.
xalan.apache.org
- ❑ Saxonica.com. *XSLT and XQuery Processing*.
www.saxonica.com

III. Dokumentensprachen

- ❑ Auszeichnungssprachen
- ❑ HTML
- ❑ Cascading Stylesheets CSS
- ❑ XML-Grundlagen
- ❑ XML-Schema
- ❑ Die XSL-Familie
- ❑ APIs für XML-Dokumente

APIs für XML-Dokumente

Einordnung

XML-Dokumente sind uns bisher in **serialisierter Form**, z.B. als Inhalte von Dateien, begegnet. Ihre Manipulation durch Programme erfordert:

1. Geeignete Repräsentation im Hauptspeicher eines Rechners
2. Möglichkeit zum Zugriff und zur Manipulation: API

“An application programming interface (API) is a set of routines, protocols, and tools for building software applications. [. . .] An API defines functionalities that are independent of their respective implementations, which allows definitions and implementations to vary without compromising each other.” [\[Wikipedia\]](#)

APIs für XML-Dokumente

Einordnung

XML-Dokumente sind uns bisher in **serialisierter Form**, z.B. als Inhalte von Dateien, begegnet. Ihre Manipulation durch Programme erfordert:

1. Geeignete Repräsentation im Hauptspeicher eines Rechners
2. Möglichkeit zum Zugriff und zur Manipulation: API

“An application programming interface (API) is a set of routines, protocols, and tools for building software applications. [...] An API defines functionalities that are independent of their respective implementations, which allows definitions and implementations to vary without compromising each other.” [\[Wikipedia\]](#)

API-Technologien zum Zugriff und zur Manipulation von XML-Dokumenten:

- ❑ DOM, Document Object Model
- ❑ SAX, Simple API for XML
- ❑ StAX, Streaming API for XML [\[XML.com\]](#)
- ❑ XPP, Common API for XML Pull Parsing
- ❑ XML Data Binding

APIs für XML-Dokumente

DOM: Historie

- 1998 DOM Level 1, Recommendation. What is the DOM? [W3C [REC](#)]
- 2004 DOM Level 3 Core, Recommendation. [W3C [REC](#)]
- 2013 DOM-Implementierung in Java 1.8 (JDK8). [[Javadoc](#)]
- 2015 DOM4, Working Draft. [W3C [WD](#), [status](#)]
- 2016 DOM. Living Standard. [[whatwg](#)]

Bemerkungen:

- ❑ Das Document Object Model, DOM, entstand aus dem Wunsch, Java- und JavaScript-Programme zwischen Browsern austauschbar zu machen. Voran gegangen waren herstellerspezifische Ideen und Implementierungen für das sogenannte „Dynamic HTML“.
- ❑ “The Document Object Model (DOM) is a platform- and language-neutral interface that will allow programs and scripts to dynamically access and update the content, structure, and style of documents.” [\[W3C\]](#)
- ❑ Level 1 und Level 2 von DOM enthalten noch keine Spezifikation dafür, wie ein Dokument in eine DOM-Struktur geladen oder aus ihr heraus gespeichert werden kann: sie setzen das Vorhandensein der Dokumente in einer Browser-Umgebung voraus. Seit DOM Level 3 gehören auch Methoden zum Laden und Speichern zur Spezifikation.
- ❑ www.quirksmode.org zeigt eine sehr gute Übersicht mit W3C DOM Compatibility Tables für die verbreiteten Browser.

APIs für XML-Dokumente

DOM: Konzepte

- ❑ DOM modelliert ein Dokument gemäß seiner Struktur als eine Hierarchie von Knoten. [W3C [DOM Level 1](#), [DOM4](#)]
- ❑ Für die Knoten definiert DOM keine Datentypen, sondern **Objekte**. Die Spezifikation beschreibt **Interfaces** mit den für die Knotenobjekte erlaubten Operationen. [[W3C](#)]
- ❑ Die Semantik der Knotenobjekte orientiert sich am [XML Information Set](#).

APIs für XML-Dokumente

DOM: Konzepte

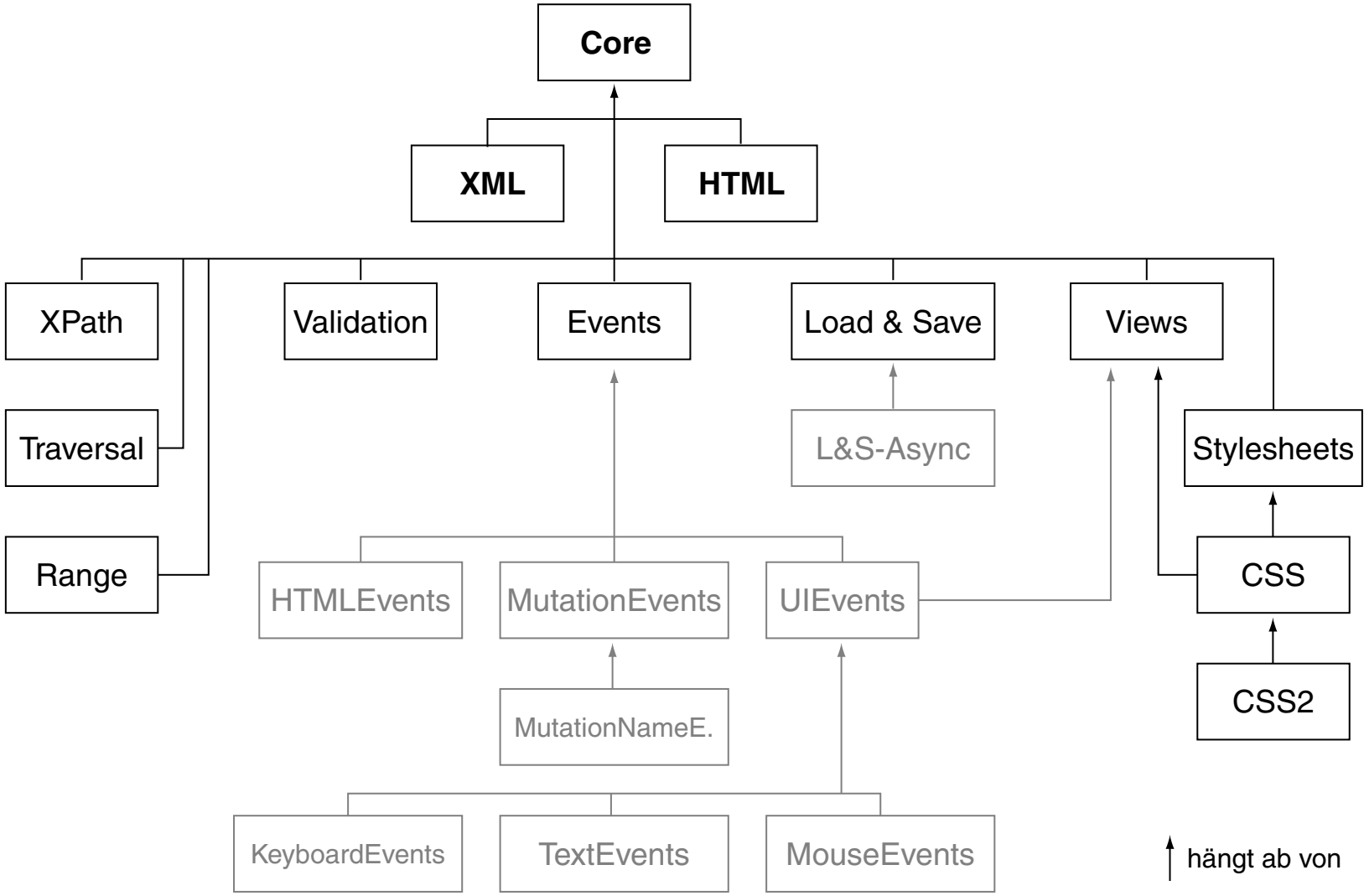
- ❑ DOM modelliert ein Dokument gemäß seiner Struktur als eine Hierarchie von Knoten. [W3C [DOM Level 1](#), [DOM4](#)]
- ❑ Für die Knoten definiert DOM keine Datentypen, sondern **Objekte**. Die Spezifikation beschreibt **Interfaces** mit den für die Knotenobjekte erlaubten Operationen. [W3C]
- ❑ Die Semantik der Knotenobjekte orientiert sich am [XML Information Set](#).
- ❑ Die DOM-Spezifikation ist neutral in Bezug auf Betriebssysteme und Programmiersprachen: die Interfaces sind in der *Interface Definition Language* [Web IDL](#) verfasst.
- ❑ Die sprachspezifische Umsetzung von DOM erfolgt durch sogenannte **Language Bindings**. [W3C [ECMAScript](#)]

Bemerkungen:

- ❑ Oft wird mit dem Begriff „DOM“ auch die Datenstruktur zur Repräsentation eines Dokuments bezeichnet – und nicht die Programmierschnittstelle (API) zum Zugriff auf die Datenstruktur. Diese Sicht motiviert sich aus dem Verständnis der objektorientierten Programmierung:
“The name *Document Object Model* was chosen because it is an *object model* in the traditional object oriented design sense: documents are modeled using objects, and the model encompasses not only the structure of a document, but also the behavior of a document and the objects of which it is composed.” [\[W3C\]](#)
- ❑ Wiederholung. Das W3C hat mittlerweile drei Datenmodelle für XML-Dokumente definiert: XML Information Set, XPath, Document Object Model (DOM). Das XPath-Datenmodell basiert auf einer Baumstruktur, die bei der Abfrage eines XML-Dokuments durchlaufen wird und ist dem [XML Information Set](#) ähnlich; DOM ist der Vorläufer beider Datenmodelle. DOM und das XPath-Datenmodell können als Interpretationen des XML Information Sets betrachtet werden. [\[MSDN\]](#)
- ❑ Mit Hilfe einer Schnittstellenbeschreibungssprache (*Interface Definition Language, IDL*) lassen sich Objekte und die auf sie anwendbaren Methoden einschließlich Parametern und Datentypen beschreiben, ohne dabei die Eigenschaften einer bestimmten Programmiersprache zu verwenden. Ein Compiler kann diese Definitionen in eine bestimmte Programmiersprache und Rechnerarchitektur umsetzen, das so genannte *Language Binding*. [\[Wikipedia IDL, Language Binding\]](#)

APIs für XML-Dokumente

DOM Level 3: Struktur der API [\[W3C\]](#)



Bemerkungen:

- ❑ Über die Interfaces greifen Scriptsprachen wie JavaScript oder JScript, Browser-Plug-Ins und ActiveX-Controls auf HTML-Dokumente im Browser zu.
- ❑ Die Interfaces der meisten DOM-Objekte sind von dem generischen `node`-Interface abgeleitet. Das `node`-Interface behandelt die gemeinsamen Anteile der verschiedenen Knoten eines XML-Baums.
- ❑ Mit DOM4 hat das W3C die Modularisierung der DOM-API aufgegeben.

APIs für XML-Dokumente

DOM: Java Language Binding

org.w3c.dom-Package [\[Javadoc\]](#) :

org.w3c.dom (Java Platform SE 7)

File Edit View History Bookmarks Tools Help

http://docs.oracle.com/javase/7/docs/api/index.html?org/w3c/dom/package-summary.html

org.w3c.dom (Java Platform SE 7)

org.omg.PortableServer
org.omg.PortableServer.CurrentPa
org.omg.PortableServer.POAMana
org.omg.PortableServer.POAPack
org.omg.PortableServer.portable
org.omg.PortableServer.ServantLo
org.omg.SendingContext
org.omg.stub.java.rmi
org.w3c.dom
org.w3c.dom.bootstrap
org.w3c.dom.events
org.w3c.dom.ls
org.xml.sax
org.xml.sax.ext
org.xml.sax.helpers

Overview **Package** Class Use Tree Deprecated Index Help

Prev Package Next Package Frames No Frames

Package org.w3c.dom

Provides the interfaces for the Document Object Model (DOM) which is a component API of the Java API for XML Processing.

See: Description

Interface Summary

Interface	Description
Attr	The <code>Attr</code> interface represents an attribute in an <code>Element</code> object.
CDATASection	CDATA sections are used to escape blocks of text containing characters that would otherwise be regarded as markup.
CharacterData	The <code>CharacterData</code> interface extends <code>Node</code> with a set of attributes and methods for accessing character data in the DOM.
Comment	This interface inherits from <code>CharacterData</code> and represents the content of a comment, i.e., all the characters between the starting ' <code><!--</code> ' and ending ' <code>--></code> '.
Document	The <code>Document</code> interface represents the entire HTML or XML document.
DocumentFragment	<code>DocumentFragment</code> is a "lightweight" or "minimal" <code>Document</code> object.
DocumentType	Each <code>Document</code> has a <code>doctype</code> attribute whose value is either <code>null</code> or a <code>DocumentType</code> object.
DOMConfiguration	The <code>DOMConfiguration</code> interface represents the configuration of a document and maintains a table

APIs für XML-Dokumente

DOM: Java Language Binding (Fortsetzung)

Methoden des `node`-Interface [\[Javadoc\]](#) :

Method Summary

Methods

Modifier and Type	Method and Description
Node	appendChild(Node newChild) Adds the node <code>newChild</code> to the end of the list of children of this node.
Node	cloneNode(boolean deep) Returns a duplicate of this node, i.e., serves as a generic copy constructor for nodes.
short	compareDocumentPosition(Node other) Compares the reference node, i.e.
NamedNodeMap	getAttributes() A <code>NamedNodeMap</code> containing the attributes of this node (if it is an <code>Element</code>) or <code>null</code> otherwise.
String	getBaseURI() The absolute base URI of this node or <code>null</code> if the implementation wasn't able to obtain an absolute URI.
NodeList	getChildNodes() A <code>NodeList</code> that contains all children of this node.
Object	getFeature(String feature, String version) This method returns a specialized object which implements the specialized APIs of the specified feature and version, as specified in .
Node	getFirstChild() The first child of this node.
Node	getLastChild() The last child of this node.
String	getLocalName() Returns the local part of the qualified name of this node.
String	getNamespaceURI() The namespace URI of this node, or <code>null</code> if it is unspecified (see).
Node	getNextSibling() The node immediately following this node.

APIs für XML-Dokumente

DOM: Anwendung

```
<?xml version="1.0" standalone="no" ?>
<?xml-stylesheet type="text/xsl" ... ?>

<personen>
  <person>
    <name>
      <vorname>Alan</vorname>
      <nachname>Turing</nachname>
    </name>
    <geburtstag>23. Juni 1912</geburtstag>
    <beruf>Mathematiker</beruf>
    <beruf>Informatiker</beruf>
  </person>

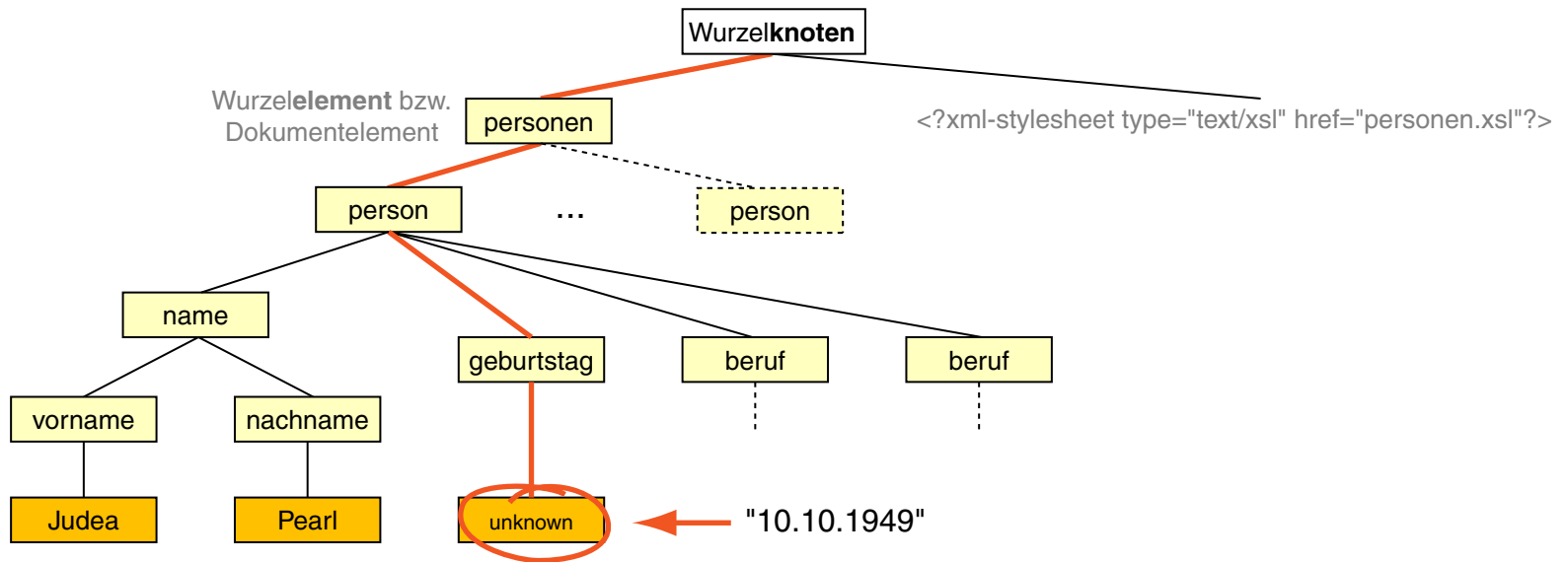
  <person>
    <name>
      <vorname>Judea</vorname>
      <nachname>Pearl</nachname>
    </name>
    <geburtstag>unknown</geburtstag>
    <beruf>Informatiker</beruf>
  </person>
</personen>
```

APIs für XML-Dokumente

DOM: Anwendung (Fortsetzung)

Aufgabe:

1. Die Person „Judea Pearl“ finden.
2. Seinen Geburtstag auf einen bestimmten Wert setzen.



APIs für XML-Dokumente

DOM: Anwendung (Fortsetzung)

Java-Klasse:

```
package documentlanguages.xmlparser.dom;

import java.io.*;
import org.w3c.dom.*;
import org.xml.sax.*;
import javax.xml.*;

public class DomParserExample {

    public Document load(String filename)...

    public Node findPerson(Node node, String firstName, String lastName)...
    private boolean matchesPerson(Node n, String firstName, ...
    public void setBirthday(Node personNode, String birthday) ...

    public void save(Document docNode, String filename, String encoding)...

    public static void main(String[] args) {...
}
```

APIs für XML-Dokumente

DOM: Anwendung (Fortsetzung)

main-Methode:

```
public static void main(String[] args) throws Exception {  
  
    DomParserExample dpe = new DomParserExample();  
    Document docNode =  
        dpe.load("./bin/documentlanguages/xmlparser/personen.xml");  
  
    Node personNode = dpe.findPerson(docNode, "Judea", "Pearl");  
    dpe.setBirthday(personNode, "10.10.1949");  
  
    dpe.save(docNode,  
            "./bin/documentlanguages/xmlparser/personen-neu.xml", "UTF-8");  
}
```


APIs für XML-Dokumente

DOM: Anwendung (Fortsetzung)

DOM-Parser instantiieren und Dokument in DOM-Objektmodell einlesen:

```
public Document load(String filename)
    throws ParserConfigurationException, IOException, SAXException {

    DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
    DocumentBuilder docBuilder = dbf.newDocumentBuilder();
    return docBuilder.parse(new File(filename));
}
```

APIs für XML-Dokumente

DOM: Anwendung (Fortsetzung)

Navigation im Dokumentbaum. DFS mit generischem `node`-Interface [\[Javadoc\]](#) :

```
public Node findPerson(Node node, String firstName, String lastName)
{
    if(matchesPerson(node, firstName, lastName))
    {
        return node;
    }
    // Perform Depth First Search (DFS).
    NodeList nodeList = node.getChildNodes();
    for(int i=0; i< nodeList.getLength(); ++i)
    {
        Node person = findPerson(nodeList.item(i), firstName, lastName);
        if(person != null) {return person;}
    }
    return null;
}
```

Vergleiche XPath-Variante.

APIs für XML-Dokumente

DOM: Anwendung (Fortsetzung)

Navigation im Dokumentbaum. DFS mit generischem `node`-Interface:

```
private boolean matchesPerson(Node n, String firstName, String lastName){
    if(!n.getNodeName().equals("person")) {return false;}
    NodeList personChildren = n.getChildNodes();
    for(int i=0; i< personChildren.getLength(); ++i){
        Node personChild = personChildren.item(i);
        if(personChild.getNodeName().equals("name")){
            boolean FIRSTNAME_OK = false, LASTNAME_OK = false;
            NodeList nameChildren = personChild.getChildNodes();
            for(int j=0; j< nameChildren.getLength(); ++j){
                Node nameChild = nameChildren.item(j);
                if(nameChild.getNodeName().equals("vorname") &&
                    nameChild.getTextContent().equals(firstName))
                    {FIRSTNAME_OK = true;}
                else if(nameChild.getNodeName().equals("nachname") &&
                    nameChild.getTextContent().equals(lastName))
                    {LASTNAME_OK = true;}
                if(FIRSTNAME_OK && LASTNAME_OK) {return true;}
            }
        }
    }
    return false;
}
```

APIs für XML-Dokumente

DOM: Anwendung (Fortsetzung)

Navigation im Dokumentbaum. DFS mit generischem `node`-Interface:

```
private boolean matchesPerson(Node n, String firstName, String lastName){
    if(!n.getNodeName().equals("person")) {return false;}
    NodeList personChildren = n.getChildNodes();
    for(int i=0; i< personChildren.getLength(); ++i){
        Node personChild = personChildren.item(i);
        if(personChild.getNodeName().equals("name")){
            boolean FIRSTNAME_OK = false, LASTNAME_OK = false;
            NodeList nameChildren = personChild.getChildNodes();
            for(int j=0; j< nameChildren.getLength(); ++j){
                Node nameChild = nameChildren.item(j);
                if(nameChild.getNodeName().equals("vorname") &&
                    nameChild.getTextContent().equals(firstName))
                    {FIRSTNAME_OK = true;}
                else if(nameChild.getNodeName().equals("nachname") &&
                    nameChild.getTextContent().equals(lastName))
                    {LASTNAME_OK = true;}
                if(FIRSTNAME_OK && LASTNAME_OK) {return true;}
            }
        }
    }
    return false;
}
```

APIs für XML-Dokumente

DOM: Anwendung (Fortsetzung)

Navigation im Dokumentbaum. DFS mit generischem `node`-Interface:

```
private boolean matchesPerson(Node n, String firstName, String lastName){
    if(!n.getNodeName().equals("person")) {return false;}
    NodeList personChildren = n.getChildNodes();
    for(int i=0; i< personChildren.getLength(); ++i){
        Node personChild = personChildren.item(i);
        if(personChild.getNodeName().equals("name")){
            boolean FIRSTNAME_OK = false, LASTNAME_OK = false;
            NodeList nameChildren = personChild.getChildNodes();
            for(int j=0; j< nameChildren.getLength(); ++j){
                Node nameChild = nameChildren.item(j);
                if(nameChild.getNodeName().equals("vorname") &&
                    nameChild.getTextContent().equals(firstName))
                    {FIRSTNAME_OK = true;}
                else if(nameChild.getNodeName().equals("nachname") &&
                    nameChild.getTextContent().equals(lastName))
                    {LASTNAME_OK = true;}
                if(FIRSTNAME_OK && LASTNAME_OK) {return true;}
            }
        }
    }
    return false;
}
```

APIs für XML-Dokumente

DOM: Anwendung (Fortsetzung)

<geburtstag>-Knoten im <person>-Knoten ändern.

Variante mit generischem `node`-Interface:

```
public void setBirthday(Node personNode, String birthday) {  
  
    NodeList personChildren = personNode.getChildNodes();  
    for(int i=0; i<personChildren.getLength(); ++i) {  
        Node personChild = personChildren.item(i);  
        if (personChild.getNodeName().equals("geburtstag")) {  
            System.out.println(" [DOM] Updating geburtstag: " +  
                personChild.getTextContent() + " -> " + birthday);  
            personChild.setTextContent(birthday);  
        }  
    }  
}
```

APIs für XML-Dokumente

DOM: Anwendung (Fortsetzung)

<geburtstag>-Knoten im <person>-Knoten ändern.

Variante mit Element-Interface [\[Javadoc\]](#) :

```
public void setBirthday(Node personNode, String birthday) {  
  
    Element person = (Element) personNode;  
    NodeList birthdayElements = person.getElementsByTagName("geburtstag");  
    if (birthdayElements.getLength() > 0)  
    {  
        System.out.println(" [DOM] Updating geburtstag: " +  
            birthdayElements.item(0).getTextContent() + " -> " + birthday);  
        birthdayElements.item(0).setTextContent(birthday);  
    }  
}
```

APIs für XML-Dokumente

DOM: Anwendung (Fortsetzung)

Dokumentmodell serialisieren:

```
public void save(Document docNode, String filename, String encoding)
    throws IOException, TransformerException,
    UnsupportedEncodingException {

    TransformerFactory tf = TransformerFactory.newInstance();
    Transformer serializer = tf.newTransformer();

    serializer.setOutputProperty(OutputKeys.ENCODING, encoding);
    serializer.transform(new DOMSource(docNode),
        new StreamResult(new FileOutputStream(filename)));
}
```


APIs für XML-Dokumente

DOM: Anwendung (Fortsetzung)

Aufruf in der Shell:

```
[user@pc WORKINGDIR]$ java -cp CLASSPATH  
documentlanguages.xmlparser.dom.DomParserExample
```

```
[DOM] Updating geburtstag: unknown -> 10.10.1949
```

APIs für XML-Dokumente

DOM: Anwendung (Fortsetzung)

Navigation im Dokumentbaum. Direkter Zugriff mit `xpath`-Interface [\[Javadoc\]](#) :

```
public Node findPerson(Node node, String firstName, String lastName)
    throws XPathExpressionException{

    XPathFactory factory = XPathFactory.newInstance();
    XPath xpath = factory.newXPath();
    String path = "//person[name/vorname= '" + firstName + "' and " +
        "name/nachname='" + lastName + "']";
    return (Node) xpath.evaluate(path, node, XPathConstants.NODE);
}
```

Vergleiche DFS-Variante.

APIs für XML-Dokumente

SAX: Historie

- 1997 Unter der Koordination von [David Megginson](#) entwickeln Teilnehmer der XML-DEV Mailing List einen einfachen, effizienten Parser.
- 2004 SAX 2.0.2. [saxproject.org]
- 2013 SAX2-Implementierung in Java 1.8 (JDK8). [[Javadoc](#)]

Bemerkungen:

- ❑ Die Simple API for XML, SAX, entstand aus dem Wunsch, die Entwicklung von Programmen zur Verarbeitung von XML-Dokumenten (XML-Prozessoren) zu vereinfachen.
- ❑ SAX stellt einen „leichtgewichtigen“ Ansatz für die ereignisbasierte Verarbeitung von XML-Dokumenten dar. SAX ist kein Parser sondern ein Gerüst in Form einer Schnittstellensammlung, um Parser zu implementieren.
- ❑ Ursprünglich entstand die SAX-API aus einer Sammlung generischer Java-Schnittstellen für XML-Parser. Inzwischen hat sie sich als eigenständige Möglichkeit zur Verarbeitung von XML-Dokumenten in verschiedenen Hochsprachen entwickelt. Neben den Umsetzungen für Java existieren auch Implementierungen für C++, Python, Perl und Eiffel.

APIs für XML-Dokumente

SAX: Konzepte [\[Wikipedia\]](#)

Verwendung eines SAX-Parsers in folgenden Schritten:

1. Instantiierung einer spezifischen Parser-Ausprägung.
Stichwort: Factory-Pattern
2. Implementierung und Zuweisung eines Content-Handlers.
3. Aufruf des Parsers.

Konsequenzen:

- ❑ Das **Dokument definiert die Ereignisse**, auf die der Parser reagiert.
- ❑ Parse-Methoden werden nicht explizit vom Programmierer aufgerufen.
- ❑ Das Programm weist keinen erkennbaren Parse-Kontrollfluss auf.

Stichwort: **Push-Prinzip**

Bemerkungen:

- ❑ Die SAX-API impliziert keine spezielle Datenstruktur. Deshalb ist der Ansatz mit nur geringen Modifikationen auf viele Programmiersprachen übertragbar.
- ❑ Das Push-Prinzip stellt nur minimale Speicheranforderungen: nur die Tiefe des Dokumentbaums und die Übergabeparameter der Callback-Funktionen sind verantwortlich für den variablen Teil des “Memory Footprint”.
- ❑ Aus dem Prinzip der SAX-Verarbeitung folgt, dass keine (in-Memory) Modifikationen am Eingabedokument möglich sind. Modifikationen werden durch eine veränderte Ausgabe des Eingabedokuments realisiert. Der Umfang dieser Transformationen hängt davon ab, wieviel von dem Eingabedokument während des Parse-Vorgangs zwischenspeichert wird.

APIs für XML-Dokumente

SAX: Struktur der API

1. Parser-Factory.

Dient zur Erzeugung verschiedener Ausprägungen eines Parsers. Optionen sind u. a.: validierend, nicht-validierend, Namensraum-auswertend.

2. Parser.

Definiert abstrakte Schnittstellen und bedient die Callback-Funktionen in diesen Schnittstellen beim Eintreffen der entsprechenden Ereignisse.

3. Schnittstellen.

- | | |
|---------------------------------|---|
| (a) <code>ContentHandler</code> | Methoden zur Reaktion auf Dokumentereignisse |
| (b) <code>ErrorHandler</code> | Methoden zur Reaktion auf in der XML-Spezifikation definierte Fehlerereignisse: <code>warning</code> , <code>error</code> , <code>fatalError</code> |
| (c) <code>DTDHandler</code> | Methoden für Notation-Deklarationen und ungeparste Entities |
| (d) <code>EntityResolver</code> | Methoden zur Namensauflösung von Entities |

APIs für XML-Dokumente

SAX: Struktur der API (Fortsetzung)

Wichtige Methoden (Callback-Funktionen) der `ContentHandler`-Schnittstelle:

Methode	Beschreibung
<code>startDocument()</code>	einmaliger Aufruf bei Beginn eines Dokuments
<code>startElement()</code>	Aufruf bei Beginn (öffnender Tag) eines Elements
<code>characters()</code>	Aufruf bei der Verarbeitung von Zeichenkettendaten innerhalb eines Elements
<code>ignorableWhitespace()</code>	Aufruf beim Auftreten ignorierbarer Leerzeichen
<code>endElement()</code>	Aufruf bei Erreichen eines Elementendes
<code>endDocument()</code>	letztes Ereignis eines Parse-Vorgangs

APIs für XML-Dokumente

SAX: Anwendung

```
<?xml version="1.0" standalone="no" ?>
<?xml-stylesheet type="text/xsl" ... ?>

<personen>
  <person>
    <name>
      <vorname>Alan</vorname>
      <nachname>Turing</nachname>
    </name>
    <geburtstag>23. Juni 1912</geburtstag>
    <beruf>Mathematiker</beruf>
    <beruf>Informatiker</beruf>
  </person>

  <person>
    <name>
      <vorname>Judea</vorname>
      <nachname>Pearl</nachname>
    </name>
    <geburtstag>unknown</geburtstag>
    <beruf>Informatiker</beruf>
  </person>
</personen>
```

APIs für XML-Dokumente

SAX: Anwendung

```
<?xml version="1.0" standalone="no" ?> → startDocument()  
<?xml-stylesheet type="text/xsl" ... ?>
```

```
<personen>  
  <person>  
    <name>  
      <vorname>Alan</vorname>  
      <nachname>Turing</nachname>  
    </name>  
    <geburtstag>23. Juni 1912</geburtstag>  
    <beruf>Mathematiker</beruf>  
    <beruf>Informatiker</beruf>  
  </person>  
  
  <person>  
    <name>  
      <vorname>Judea</vorname>  
      <nachname>Pearl</nachname>  
    </name>  
    <geburtstag>unknown</geburtstag>  
    <beruf>Informatiker</beruf>  
  </person>  
</personen>
```

APIs für XML-Dokumente

SAX: Anwendung

```
<?xml version="1.0" standalone="no" ?> → startDocument()  
<?xml-stylesheet type="text/xsl" ... ?> processingInstruction()
```

```
<personen>  
  <person>  
    <name>  
      <vorname>Alan</vorname>  
      <nachname>Turing</nachname>  
    </name>  
    <geburtstag>23. Juni 1912</geburtstag>  
    <beruf>Mathematiker</beruf>  
    <beruf>Informatiker</beruf>  
  </person>  
  
  <person>  
    <name>  
      <vorname>Judea</vorname>  
      <nachname>Pearl</nachname>  
    </name>  
    <geburtstag>unknown</geburtstag>  
    <beruf>Informatiker</beruf>  
  </person>  
</personen>
```

APIs für XML-Dokumente

SAX: Anwendung

```
<?xml version="1.0" standalone="no" ?> → startDocument()
<?xml-stylesheet type="text/xsl" ... ?> processingInstruction()

<personen> startElement()
  <person>
    <name>
      <vorname>Alan</vorname>
      <nachname>Turing</nachname>
    </name>
    <geburtstag>23. Juni 1912</geburtstag>
    <beruf>Mathematiker</beruf>
    <beruf>Informatiker</beruf>
  </person>

  <person>
    <name>
      <vorname>Judea</vorname>
      <nachname>Pearl</nachname>
    </name>
    <geburtstag>unknown</geburtstag>
    <beruf>Informatiker</beruf>
  </person>
</personen>
```

APIs für XML-Dokumente

SAX: Anwendung

```
<?xml version="1.0" standalone="no" ?> → startDocument()
<?xml-stylesheet type="text/xsl" ... ?> processingInstruction()

<personen>
  <person>
    <name>
      <vorname>Alan</vorname>
      <nachname>Turing</nachname>
    </name>
    <geburtstag>23. Juni 1912</geburtstag>
    <beruf>Mathematiker</beruf>
    <beruf>Informatiker</beruf>
  </person>

  <person>
    <name>
      <vorname>Judea</vorname>
      <nachname>Pearl</nachname>
    </name>
    <geburtstag>unknown</geburtstag>
    <beruf>Informatiker</beruf>
  </person>
</personen>
```

APIs für XML-Dokumente

SAX: Anwendung

```
<?xml version="1.0" standalone="no" ?> → startDocument()
<?xml-stylesheet type="text/xsl" ... ?> processingInstruction()

<personen>
  <person>
    <name>
      <vorname>Alan</vorname>
      <nachname>Turing</nachname>
    </name>
    <geburtstag>23. Juni 1912</geburtstag>
    <beruf>Mathematiker</beruf>
    <beruf>Informatiker</beruf>
  </person>

  <person>
    <name>
      <vorname>Judea</vorname>
      <nachname>Pearl</nachname>
    </name>
    <geburtstag>unknown</geburtstag>
    <beruf>Informatiker</beruf>
  </person>
</personen>
```

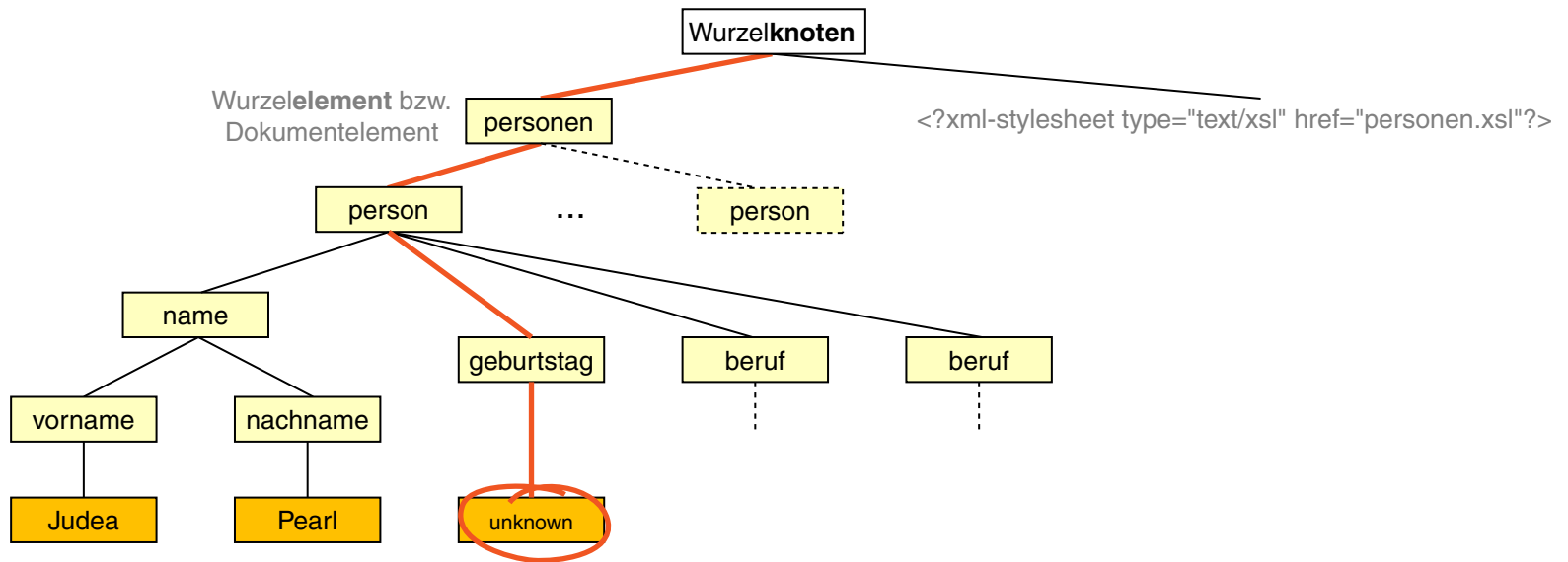
startElement()
startElement()
startElement()
startElement() characters() ...
startElement() characters() ...
endElement()
startElement() characters() ...
startElement() characters() ...
startElement() characters() ...
endElement()
...

APIs für XML-Dokumente

SAX: Anwendung (Fortsetzung)

Aufgabe:

1. Die Person „Judea Pearl“ finden.
2. Seinen Geburtstag ausgeben.



APIs für XML-Dokumente

SAX: Anwendung (Fortsetzung)

SAX-Parser instantiieren und XML-Ereignisstrom öffnen:

```
package documentlanguages.xmlparser.sax;

import java.io.*;
import org.xml.sax.*;
import org.xml.sax.helpers.*;

public class SAXParserExample {

    public void load(String filename, DefaultHandler handler)
        throws SAXException, IOException {
        XMLReader xr = XMLReaderFactory.createXMLReader();
        xr.setContentHandler(handler);
        xr.parse(filename);
    }

    public static void main(String[] args) throws SAXException, IOException{
        SAXParserExample spe = new SAXParserExample();
        DefaultHandler handler = new SAXParserExampleHandler("Judea", "Pearl");
        spe.load("./bin/documentlanguages/xmlparser/personen.xml", handler);
    }
}
```


APIs für XML-Dokumente

SAX: Anwendung (Fortsetzung)

Schnittstellenklasse mit eigenen Callback-Funktionen:

```
package documentlanguages.xmlparser.sax;

import org.xml.sax.*;
import org.xml.sax.helpers.*;

public class SAXParserExampleHandler extends DefaultHandler{

    boolean parseVorname, parseNachname, parseGeburtstag;
    String vorname, nachname, geburtstag;
    String targetFirstName, targetLastName;

    public SAXParserExampleHandler(String firstName, String lastName){
        targetFirstName = firstName;
        targetLastName = lastName;
    }

    public void startElement(String uri, String localName, String qName,...
    public void characters(char[] ch, int start, int length){...
    public void endElement(String uri, String localName, String qName){...
}
```

Bemerkungen:

- ❑ Die Klasse `SAXParserExampleHandler` ist von der Klasse `DefaultHandler` abgeleitet, die für jede Callback-Funktion eine Default-Implementierung enthält, die nichts tut. Somit müssen nur diejenigen Methoden überschrieben werden, die genau die Ereignisse verarbeiten, an denen man interessiert ist.

APIs für XML-Dokumente

SAX: Anwendung (Fortsetzung)

Verarbeitung von Elementstart-Ereignissen (= Zustandsmarkierung):

```
public void startElement(String uri, String localName, String qName,
    Attributes attributes) {

    if(qName.equals("vorname")) {parseVorname = true;}
    if(qName.equals("nachname")) {parseNachname = true;}
    if(qName.equals("geburtstag")) {parseGeburtstag = true;}

}
```

APIs für XML-Dokumente

SAX: Anwendung (Fortsetzung)

Verarbeitung von Character-Data-Ereignissen (= Einlesen):

```
public void characters(char[] ch, int start, int length) {  
  
    if(parseVorname) {  
        vorname = new String(ch, start, length);  
        parseVorname = false;  
    }  
    if(parseNachname) {  
        nachname = new String(ch, start, length);  
        parseNachname = false;  
    }  
    if(parseGeburtstag) {  
        geburtstag = new String(ch, start, length);  
        parseGeburtstag = false;  
    }  
}
```

APIs für XML-Dokumente

SAX: Anwendung (Fortsetzung)

Verarbeitung von Elementende-Ereignissen:

```
public void endElement(String uri, String localName, String qName){

    if (qName.equals("person"))
    // When leaving a <person>-element...
    {
        // If the names are correct, print the birthday.
        if (vorname.equals(targetFirstName) &&
nachname.equals(targetLastName))
        {
            System.out.println("[SAX] " + targetFirstName + " "
+ targetLastName + "'s Geburtstag: " + geburtstag);
        }
        // Reset person data.
        vorname = null; nachname = null; geburtstag = null;
    }
}
```

APIs für XML-Dokumente

SAX: Anwendung (Fortsetzung)

Aufruf in der Shell:

```
[user@pc WORKINGDIR]$ java -cp CLASSPATH  
documentlanguages.xmlparser.sax.SAXParserExample
```

```
[SAX] Judea Pearl's Geburtstag: unknown
```

Bemerkungen:

- ❑ Um sinnvoll auf Ereignisse reagieren zu können, muss sich der Zustand gemerkt werden, in dem sich der Parser befindet. Im Beispiel: tritt ein Character-Data-Ereignis ein, so soll sich die Zeichenkette nur dann gemerkt werden, falls vorher der Start-Tag eines `<vorname>`-, `<nachname>`- oder `<geburtstag>`-Elements geparsed wurde.

Stichwort: endlicher Automat

- ❑ Im Beispiel sind die Zustände des endlichen Automaten durch Variablen wie `parseVorname`, `parseNachname` oder `parseGeburtstag` codiert.

APIs für XML-Dokumente

XML Data Binding: Historie

- 2006 JSR 222. Java Specification Request für JAXB 2.0, der Java Architecture for XML Binding. [jcp.org]
- 2013 JAXB-Referenzimplementierung. [GlassFish]
- 2013 Castor. Open Source Data Binding Framework in Java. [codehaus.org]
- 2013 EclipseLink MOXy. XML Binding with JAXB and SDO. [eclipse.org]
- 2013 JAXB-Implementierung in Java 1.8 (JDK8). [Javadoc]

Bemerkungen:

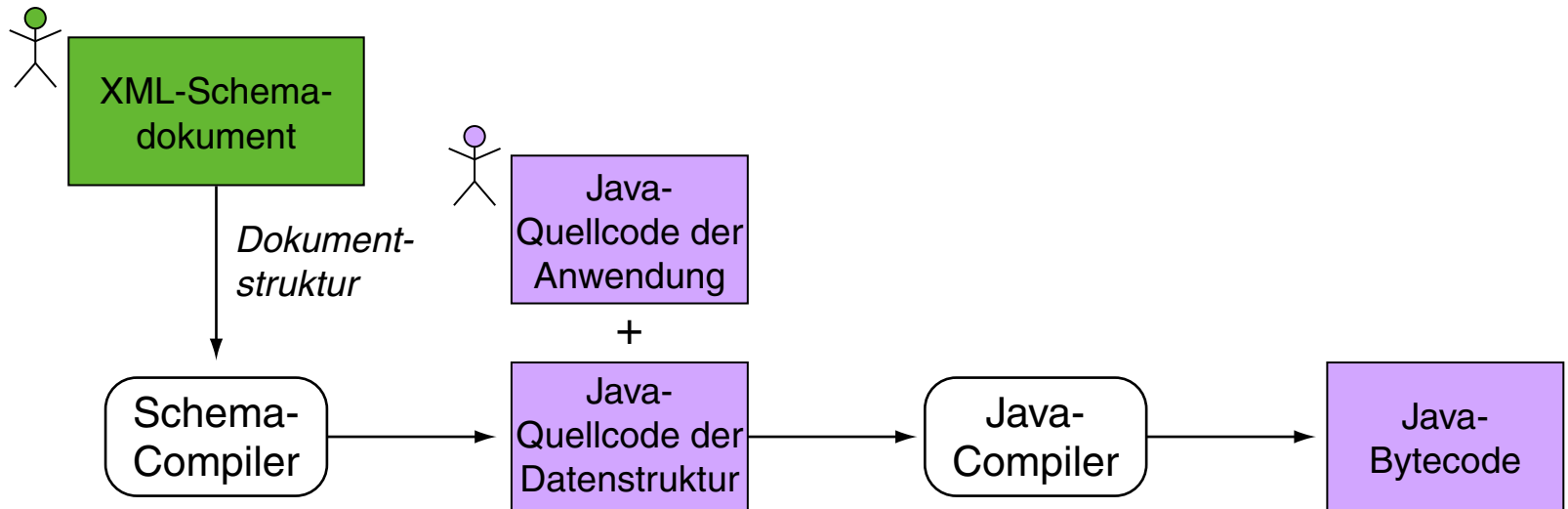
- ❑ Unter Data Binding versteht man die Abbildung einer gegebenen Datenstruktur (hier: XML-Schema) auf die Konzepte einer Zielsprache. Data Binding macht die Daten auf *Basis der Datenstrukturen der gewählten Zielsprache* verfügbar.
- ❑ Data Binding wird attraktiv, wenn Mechanismen für dessen Automatisierung existieren: die Konzepte der Zielsprache (Beschränkungen, Datenstruktur-Mapping, Setter / Getter-Methoden, etc.) werden aus Sicht des Programmierers transparent gehandelt.
- ❑ XML Data Binding: “A facility for compiling an XML schema into one or more Java classes which can parse, generate, and validate documents that follow the schema.” [jcp.org]
- ❑ DOM versus XML Data Binding:

“In the DOM approach, the parser creates a tree of objects that represents the content and organization of data in the document. The application can then navigate through the tree in memory to access the data it needs. DOM data, however, is contained in objects of a single type, linked according to the XML document’s structure, with individual node objects containing an element, an attribute, a CDATA section, etc. Values are invariably provided as strings.

Unmarshalling an XML document with the appropriate JAXB method also results in a tree of objects, with the significant difference being that the nodes in this tree correspond to XML elements, which contain attributes and the content as instance variables and refer to child elements by object references.” [GlassFish]

APIs für XML-Dokumente

XML Data Binding: Konzepte I [\[Konzepte II\]](#)

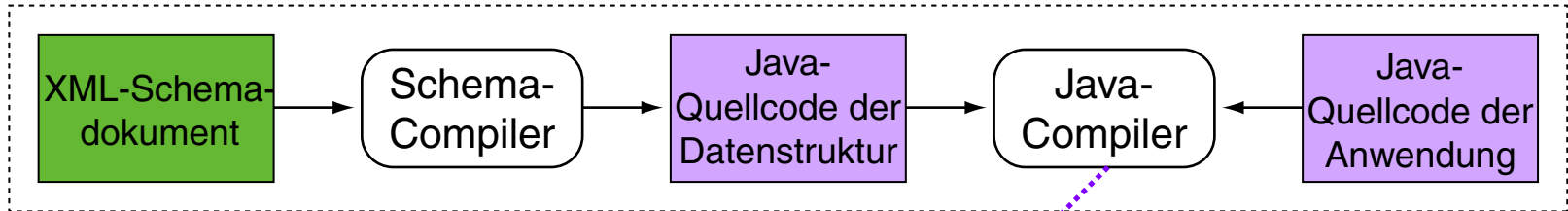


- Ein Schema-Compiler erzeugt aus dem XML-Schema Java-Klassen, die als Datentypen fungieren und den Zugriff und die Manipulation der Inhalte von Schema-validen XML-Dokumenten implementieren. [\[Oracle\]](#)
- Die eigene Anwendung setzt direkt auf den generierten Java-Klassen auf.

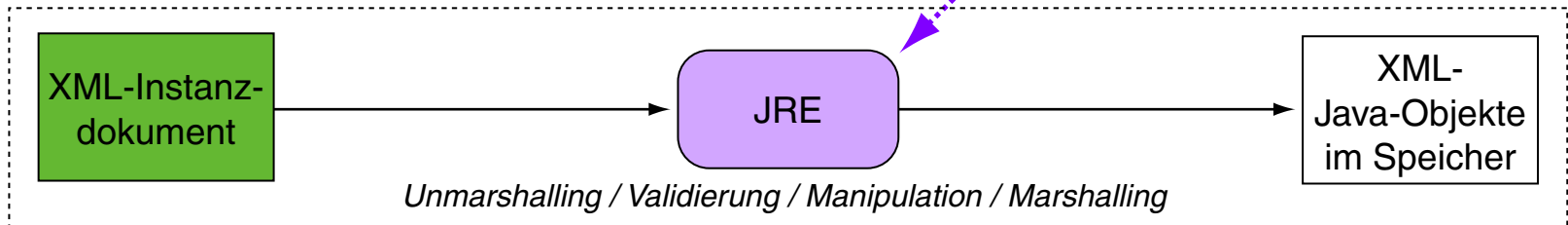
APIs für XML-Dokumente

XML Data Binding: Konzepte I [\[Konzepte II\]](#)

Programmerstellung



Programmausführung



- ❑ XML-Schema für Programmerstellung, Instanzdokument für Programmausführung.
- ❑ Unmarshalling: Lese- und Validierungsvorgang, der eine XML-Datei aus einem Eingabestrom liest und die notwendigen Speicherobjekte erzeugt.
- ❑ Marshalling: Schreiben der Speicherobjekte als XML-Datei.

APIs für XML-Dokumente

XML Data Binding: Konzepte I (Fortsetzung)

1. xjc.

XML-Schema-Compiler. Erzeugt Java-Klassen, die die Struktur der Daten gemäß eines XML-Schemadokuments abbilden.

2. JAXBContext.

Erzeugung einer Factory-Klasse für folgende Klassen:

- (a) `unmarshaller` Bildet einen XML-Stream mit Hilfe von Java-Objekten (Instanzen der von `xjc` erzeugten Klassen) im Speicher ab.
- (b) `marshaller` Serialisiert die gespeicherte Objektstruktur als XML-Stream.
- (c) `validator` Validiert eine gegebene XML-Datei gemäß dem zugrunde liegenden XML-Schema des JAXBContext-Objekts.

3. Getter- und Setter-Methoden.

Manipulation von Element- und Attributwerten.

APIs für XML-Dokumente

XML Data Binding: Anwendung

```
<?xml version="1.0" standalone="no" ?>
<?xml-stylesheet type="text/xsl" ... ?>

<personen>
  <person>
    <name>
      <vorname>Alan</vorname>
      <nachname>Turing</nachname>
    </name>
    <geburtstag>23. Juni 1912</geburtstag>
    <beruf>Mathematiker</beruf>
    <beruf>Informatiker</beruf>
  </person>

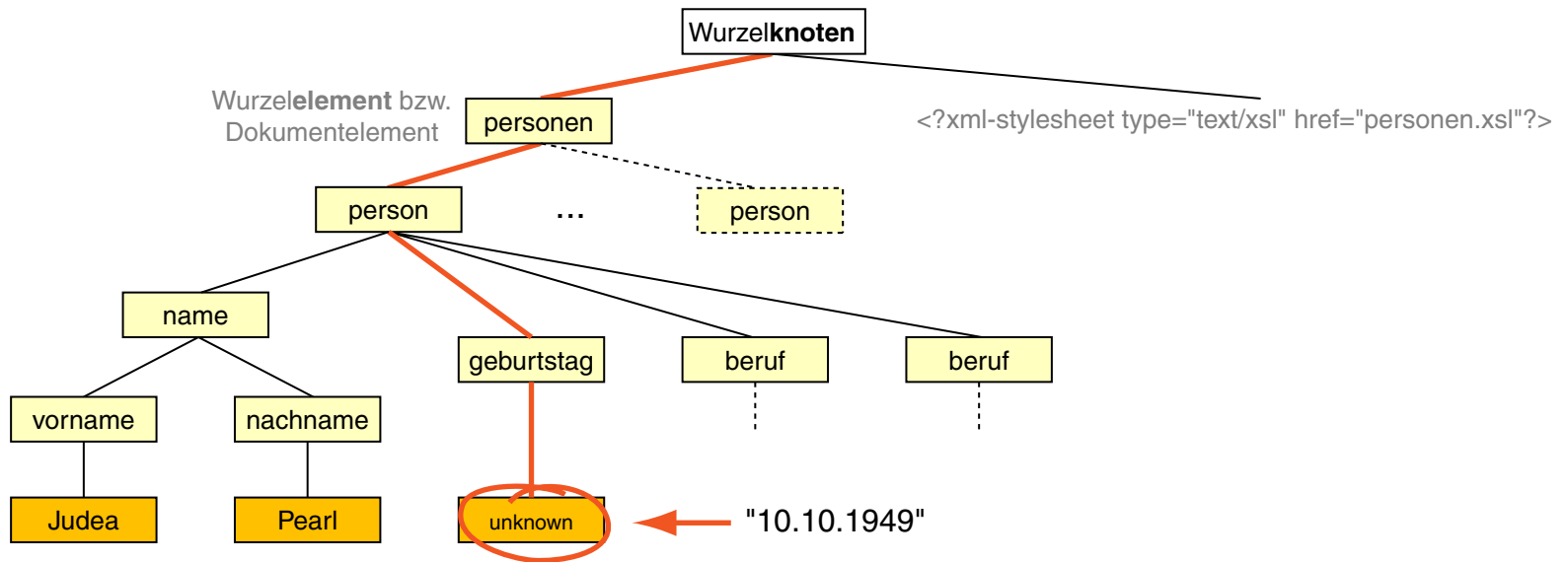
  <person>
    <name>
      <vorname>Judea</vorname>
      <nachname>Pearl</nachname>
    </name>
    <geburtstag>unknown</geburtstag>
    <beruf>Informatiker</beruf>
  </person>
</personen>
```

APIs für XML-Dokumente

XML Data Binding: Anwendung (Fortsetzung)

Aufgabe:

1. Die Person „Judea Pearl“ finden.
2. Seinen Geburtstag auf einen bestimmten Wert setzen.



APIs für XML-Dokumente

XML Data Binding: Anwendung (Fortsetzung)

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="https://www.w3.org/2001/XMLSchema" ...>
  <xs:complexType name="NameType">
    <xs:sequence>
      <xs:element name="vorname" type="xs:string" />
      <xs:element name="nachname" type="xs:string" />
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="PersonType">
    <xs:sequence>
      <xs:element name="name" type="NameType" />
      <xs:element name="geburtstag" type="xs:string" />
      <xs:element name="beruf" type="xs:string" minOccurs="0" .../>
    </xs:sequence>
  </xs:complexType>

  <xs:element name="personen">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="person" type="PersonType" minOccurs="0" .../>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

APIs für XML-Dokumente

XML Data Binding: Anwendung (Fortsetzung)

Generierung der Elementtypklassen inklusive von Getter- und Setter-Methoden (= Data Binding) mit dem Java-Schema-Compiler xjc [\[Konzepte I\]](#) :

1. Schema für die Datei `personen.xml`:

```
SOURCEDIR/documentlanguages/xmlparser/personen.xsd
```

2. Angabe der Target-Package-Option `-p` derart, dass die generierten Java-Klassen im gewünschten Package eingegliedert sind:

```
[user@pc SOURCEDIR] $ xjc  
-p documentlanguages.xmlparser.jaxb.generated.personen  
documentlanguages/xmlparser/personen.xsd
```


APIs für XML-Dokumente

XML Data Binding: Anwendung (Fortsetzung)

Generierung der Elementtypklassen inklusive von Getter- und Setter-Methoden (= Data Binding) mit dem Java-Schema-Compiler xjc [\[Konzepte I\]](#) :

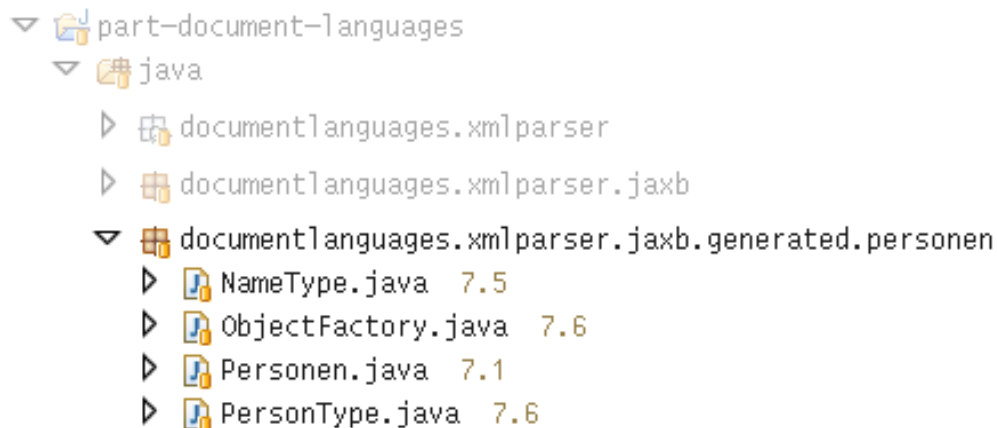
1. Schema für die Datei `personen.xml`:

`SOURCEDIR/documentlanguages/xmlparser/personen.xsd`

2. Angabe der Target-Package-Option `-p` derart, dass die generierten Java-Klassen im gewünschten Package eingegliedert sind:

```
[user@pc SOURCEDIR] $ xjc  
-p documentlanguages.xmlparser.jaxb.generated.personen  
documentlanguages/xmlparser/personen.xsd
```

3. Die entstandene Package-Struktur [\[Personen.java\]](#), [\[PersonType.java\]](#), [\[NameType.java\]](#) :



APIs für XML-Dokumente

XML Data Binding: Anwendung (Fortsetzung)

Java-Klasse:

```
package documentlanguages.xmlparser.jaxb;

import java.io.*;
import java.util.*;
import javax.xml.bind.*;

import documentlanguages.xmlparser.jaxb.generated.personen.*; // Konzepte I.
// import documentlanguages.xmlparser.jaxb.manual.personen.*; // Konzepte II.

public class JAXBParserExample1 {

    public Personen load(String filename){...
    public void setBirthday (Personen personen, ...
    public void save (Personen personen, String filename){...

    public static void main(String[] args) throws JAXBException,IOException {
        JAXBParserExample1 pe=new JAXBParserExample1();
        Personen personen=
            pe.load("./bin/documentlanguages/xmlparser/personen.xml");
        pe.setBirthday(personen, "Judea", "Pearl", "10.10.1949");
        pe.save(personen,
            "./bin/documentlanguages/xmlparser/personen-neu.xml");
    }
}
```

APIs für XML-Dokumente

XML Data Binding: Anwendung (Fortsetzung)

<personen>-Parser instantiieren, Dokument parsen und <personen>-Element im Speicher als Java-Objekt anlegen [[Javadoc](#)]:

```
public Personen load(String filename) throws FileNotFoundException,
                                   JAXBException{
    // Create JAXBContext.
    JAXBContext jc = JAXBContext.newInstance(Personen.class);
    // Create unmarshaller.
    Unmarshaller u = jc.createUnmarshaller();
    // Unmarshal an instance document into a tree of Java content objects
    // composed of classes from the xmlparser.generated.personen package.
    Personen personenElement = (Personen) u.unmarshal(new File(filename));
    return personenElement;
}
```

APIs für XML-Dokumente

XML Data Binding: Anwendung (Fortsetzung)

<geburtstag>-Element suchen und neu setzen:

```
public void setBirthday(Personen personen, String targetFirstName,
    String targetLastName, String birthday) {

    List<PersonType> personenListe = personen.getPerson();
    for (PersonType person : personenListe) {
        NameType name = person.getName();
        if (name.getNachname().equals(targetLastName) &&
            name.getVorname().equals(targetFirstName)) {
            System.out.println("[JAXB] Updating \"geburtstag\": "
                + person.getGeburtstag() + " -> " + birthday);
            person.setGeburtstag(birthday);
            break;
        }
    }
}
```

APIs für XML-Dokumente

XML Data Binding: Anwendung (Fortsetzung)

Geändertes `<personen>`-Element speichern:

```
public void save(Personen personen, String filename) throws IOException,
                                                    JAXBException{

    // Create JAXBContext.
    JAXBContext jc = JAXBContext.newInstance(Personen.class);
    // Create marshaller.
    Marshaller m=jc.createMarshaller();
    // Produce formatted output.
    m.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, Boolean.TRUE);
    // Write to file.
    m.marshal(personen, new File(filename));
}
```

APIs für XML-Dokumente

XML Data Binding: Anwendung (Fortsetzung)

Aufruf in der Shell:

```
[user@pc SOURCEDIR] $ xjc  
-p documentlanguages.xmlparser.jaxb.generated.personen  
documentlanguages/xmlparser/personen.xsd  
  
parsing a schema...  
compiling a schema...  
documentlanguages/xmlparser/jaxb/generated/personen/NameType.java  
documentlanguages/xmlparser/jaxb/generated/personen/ObjectFactory.java  
documentlanguages/xmlparser/jaxb/generated/personen/PersonType.java  
documentlanguages/xmlparser/jaxb/generated/personen/Personen.java
```

APIs für XML-Dokumente

XML Data Binding: Anwendung (Fortsetzung)

Aufruf in der Shell:

```
[user@pc SOURCEDIR] $ xjc  
-p documentlanguages.xmlparser.jaxb.generated.personen  
documentlanguages/xmlparser/personen.xsd
```

parsing a schema...

compiling a schema...

```
documentlanguages/xmlparser/jaxb/generated/personen/NameType.java  
documentlanguages/xmlparser/jaxb/generated/personen/ObjectFactory.java  
documentlanguages/xmlparser/jaxb/generated/personen/PersonType.java  
documentlanguages/xmlparser/jaxb/generated/personen/Personen.java
```

```
[user@pc SOURCEDIR] $ javac -d ../bin  
documentlanguages/xmlparser/jaxb/JAXBParserExample1.java
```

APIs für XML-Dokumente

XML Data Binding: Anwendung (Fortsetzung)

Aufruf in der Shell:

```
[user@pc SOURCEDIR]$ xjc  
-p documentlanguages.xmlparser.jaxb.generated.personen  
documentlanguages/xmlparser/personen.xsd
```

parsing a schema...

compiling a schema...

```
documentlanguages/xmlparser/jaxb/generated/personen/NameType.java  
documentlanguages/xmlparser/jaxb/generated/personen/ObjectFactory.java  
documentlanguages/xmlparser/jaxb/generated/personen/PersonType.java  
documentlanguages/xmlparser/jaxb/generated/personen/Personen.java
```

```
[user@pc SOURCEDIR]$ javac -d ../bin  
documentlanguages/xmlparser/jaxb/JAXBParserExample1.java
```

```
[user@pc WORKINGDIR]$ java -cp CLASSPATH  
documentlanguages.xmlparser.jaxb.JAXBParserExample1
```

```
[JAXB] Updating "geburtstag": unknown -> 10.10.1949
```


APIs für XML-Dokumente

XML Data Binding: Konzepte II (Manuelle Erstellung von Elementtypklassen)

Java-Klasse für das `<personen>`-Element [Konzepte I [xjc](#), [Java main](#)] :

```
package documentlanguages.xmlparser.jaxb.manual.personen;

import java.util.ArrayList;
import java.util.List;

import javax.xml.bind.annotation.XmlElement;
import javax.xml.bind.annotation.XmlRootElement;

@XmlRootElement (name="personen")
public class Personen {

    @XmlElement (name="person")
    private List<PersonType> personenListe;

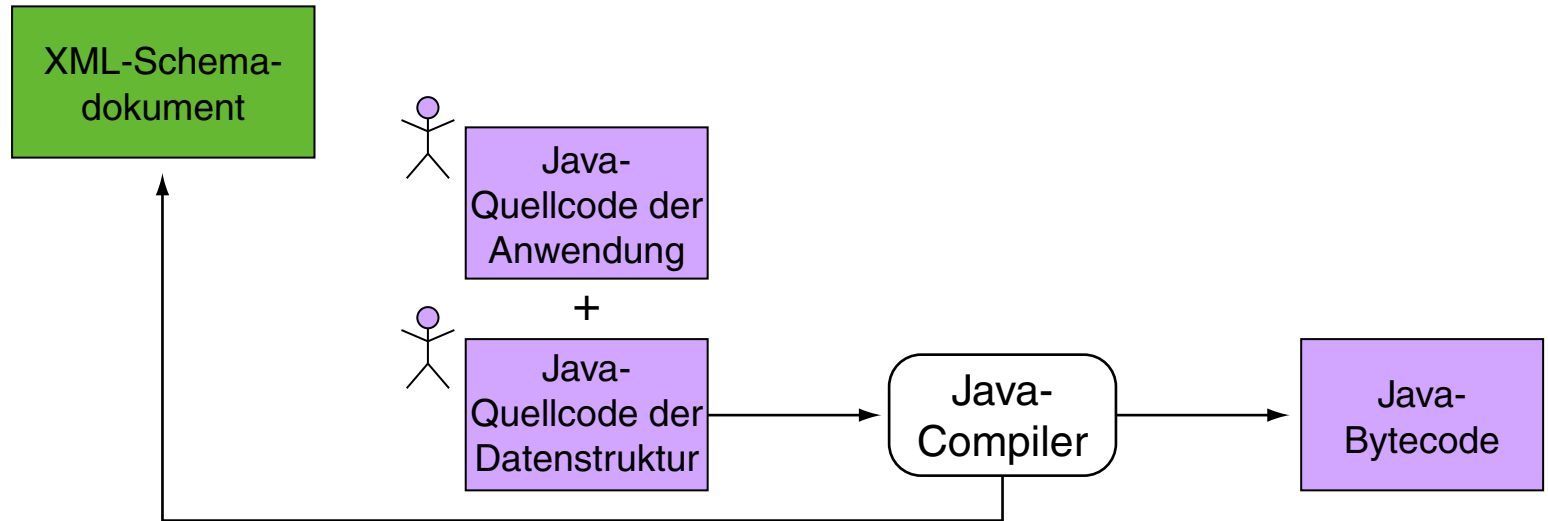
    public List<PersonType> getPersonenListe() {
        if (personenListe==null) {personenListe = new ArrayList<PersonType>();}
        return this.personenListe;
    }
}
```

Bemerkungen:

- ❑ Verwendung von Java Annotations: “Annotations, a form of metadata, provide data about a program that is not part of the program itself.” [\[Oracle\]](#)
- ❑ Hier dienen die Annotationen dazu, dem Java-Compiler mitzuteilen, dass XML-Elemente vom Typ `<person>` auf den Java-Datentyp `personenListe` gemappt werden. Vergleiche die manuell erstellte Personenklasse mit der via xjc automatisch generierten Personenklasse.
- ❑ Abstraktion über Datentypen mit Hilfe von Java Generics: “[. . .] allows a type or method to operate on objects of various types while providing compile-time type safety.” [\[Oracle\]](#)

APIs für XML-Dokumente

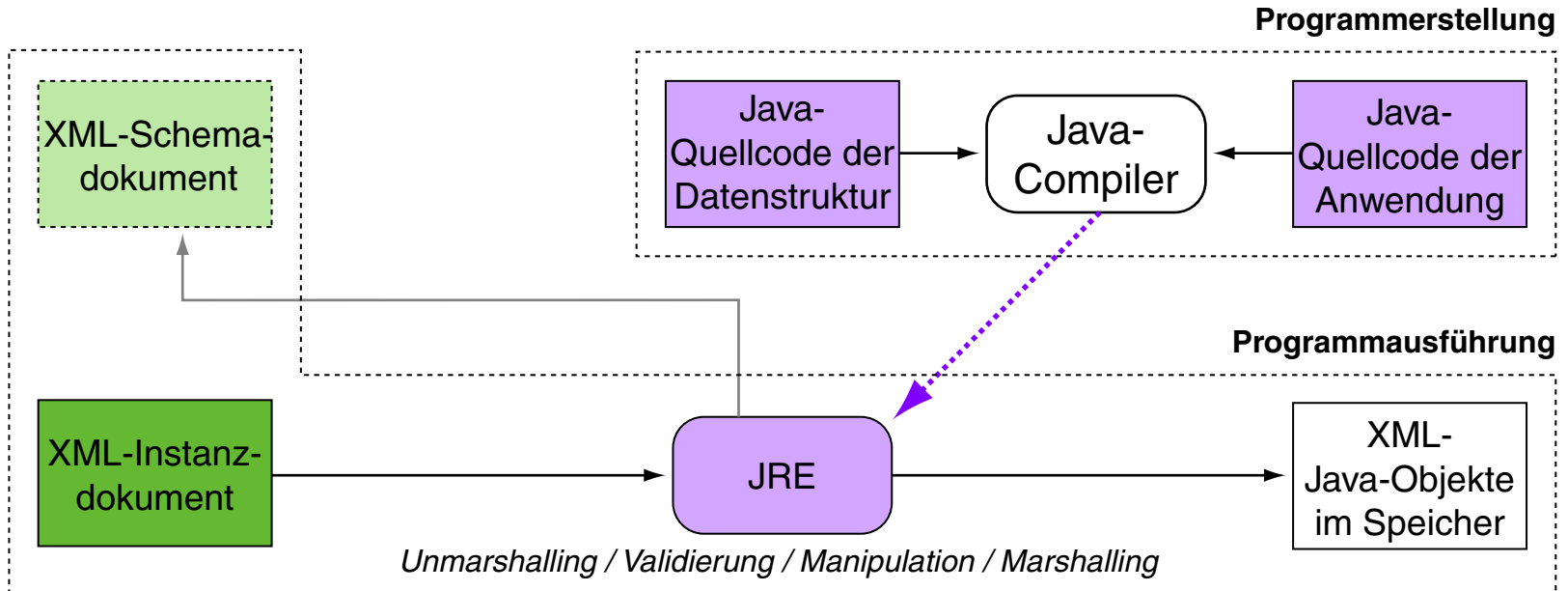
XML Data Binding: Konzepte II [\[Konzepte I\]](#)



- An die Stelle des Schema-Compilers tritt der Anwender, der annotierten Java-Quellcode der Datenstrukturen spezifiziert.

APIs für XML-Dokumente

XML Data Binding: Konzepte II [\[Konzepte I\]](#)



- Ein XML-Schema kann aus dem Java-Quellcode der Datenstruktur generiert werden.

APIs für XML-Dokumente

Diskussion der API-Technologien

DOM repräsentiert ein XML-Dokument als einen Objektbaum im Hauptspeicher.

- Das gesamte Dokument befindet sich (scheinbar) im Speicher.
- + Random-Access und einfache Manipulation von Dokumentbestandteilen
- + Laden und Speichern ist in der API (Level 3) realisiert.
- DOM-Objekte eignen sich nur bedingt als Datenstrukturen einer Applikation

SAX repräsentiert ein XML-Dokument als einen Strom von Ereignissen.

- Jedes Dokument-Token begegnet einem genau einmal.
- Kein Random-Access auf die Bestandteile eines Dokuments.
- Programmierer ist selbst verantwortlich für die Speicherung
- + Es kann flexibel bei der Speicherung von Inhalten entschieden werden.

APIs für XML-Dokumente

Diskussion der API-Technologien

DOM repräsentiert ein XML-Dokument als einen Objektbaum im Hauptspeicher.

- Das gesamte Dokument befindet sich (scheinbar) im Speicher.
- + Random-Access und einfache Manipulation von Dokumentbestandteilen
- + Laden und Speichern ist in der API (Level 3) realisiert.
- DOM-Objekte eignen sich nur bedingt als Datenstrukturen einer Applikation

SAX repräsentiert ein XML-Dokument als einen Strom von Ereignissen.

- Jedes Dokument-Token begegnet einem genau einmal.
- Kein Random-Access auf die Bestandteile eines Dokuments.
- Programmierer ist selbst verantwortlich für die Speicherung
- + Es kann flexibel bei der Speicherung von Inhalten entschieden werden.

APIs für XML-Dokumente

Diskussion der API-Technologien (Fortsetzung)

XML Data Binding repräsentiert ein XML-Dokument als eine Menge von Klassen zum Dokument-Handling.

- Hinsichtlich der Speicherung ist es mit DOM vergleichbar, eher besser.
- + Generierung von Datentypen (Klassen) und Zugriffsmethoden für XML-Elemente; die Navigation durch die Baumstruktur entfällt.
- + Die generierten Klassen sind direkt in der Applikation verwendbar.
- + Hinsichtlich Speicherplatz und Laufzeit ist der Ansatz optimal.
- Änderung des XML-Schemas erfordert die Neugenerierung der Klassen.
- Als Teil des Java Community Process ist es kein offener Standard.

Bemerkungen:

- ❑ DOM ist vorzuziehen, falls viele Manipulationen zu machen sind und falls (fremder) Scripting-Code einfachen Zugriff haben soll. Stichwort: Browser
- ❑ SAX ist vorzuziehen, falls Effizienz eine Rolle spielt oder falls die Verarbeitung hauptsächlich datenstromorientiert ist.
- ❑ DOM und SAX lassen sich in *einer* Applikation kombinieren: ein SAX-Ereignisstrom kann als Eingabe für DOM genutzt werden.
- ❑ Die Java API for XML Processing, JAXP, stellt alle notwendigen Technologien für DOM, SAX und StAX bereit. [\[Oracle\]](#)
- ❑ Die Java Architecture for XML Binding, JAXB, stellt die Data-Binding-Technologien bereit. [\[Oracle\]](#)

APIs für XML-Dokumente

Quellen zum Nachlernen und Nachschlagen im Web: Konzepte

- ❑ W3C. *What is the Document Object Model?*
www.w3.org/TR/REC-DOM-Level-1/introduction.html
- ❑ W3C. *DOM FAQ.*
www.w3.org/DOM/faq.html
- ❑ W3C. *DOM4 Candidate Recommendation.*
www.w3.org/TR/dom
- ❑ W3 Schools. *XML DOM Tutorial*
www.w3schools.com/dom

APIs für XML-Dokumente

Quellen zum Nachlernen und Nachschlagen im Web: Anwendung

- ❑ Oracle. *Java API for XML Processing, JAXP*.
docs.oracle.com/javase/tutorial/jaxp
- ❑ Oracle. *Java API für das Document Object Model, DOM*.
docs.oracle.com/javase/8/docs/api
- ❑ Oracle. *Java API für SAX2-Transformationen*.
docs.oracle.com/javase/8/docs/api

- ❑ GlassFish. *Metro Web Service Stack*.
metro.java.net
- ❑ GlassFish. *Java Architecture for XML Binding, JAXB*.
jaxb.java.net
- ❑ GlassFish. *Java API for XML Web Services, JAX-WS*.
jax-ws.java.net