Web- und XML-Technologien

BA-INF 133, Sommersemester 2018

Dr. Stefan Lüttringhaus-Kappel
Institut für Informatik, Universität Bonn
stefan@iai.uni-bonn.de

19. Juli 2018

Inhaltsverzeichnis

1 Organisatorisches

1.1 Vorlesung

Vorlesung

Vorlesungstermine

- Montags *16:15 Uhr* bis ca. 17:45 Uhr
- im Hörsaal 7, HSZ

Information und Kommunikation: eCampus

- Skript (Folieninhalte + Links + Erläuterungen + . . .)
- Literatur / Links
- Übungsaufgaben
- Diskussionsforum
- Mailingliste
- . . .
- Weitere Informationen zu eCampus gibt es in der ersten Übungsstunde.
- Hier anmelden: https://ecampus.uni-bonn.de/

1.2 Übung

Übung

Übungstermine

- Freitags *14:15 Uhr* bis ca. 15:45 Uhr
- Globalübung im Hörsaal 7, HSZ

Übungsinhalte

- Vertiefung, Verständnisaufgaben
- *Selbststudium*, eigene Recherchen
- Kurzreferate

- Open-Source-Werkzeuge
- Programmieraufgaben und Programmierprojekt (in Kleingruppen)

Wichtig

- Praktische Java-Kenntnisse sind notwendig
- Die erfolgreiche Teilnahme an den Übungen ist Voraussetzung für die Zulassung zur Prüfung.

1.3 Prüfung

Prüfung

Leistungspunkteprüfung

- Schriftliche Prüfung (Klausur)
- Termin: voraussichtlich ?. August 2018
- Nachklausur voraussichtlich ?. September 2018

Wichtig

- Das ist der Planungsstand.
- Ohne Gewähr! Termine können sich noch ändern.

Voraussetzungen für die Zulassung

1. Studium nach Bachelor-PO (oder im Nebenfach), *und*
2. erfolgreiche Übungsteilnahme (50% der zu erreichenden Punkte *und* Programmierprojekt mit Präsentation)

Überschneidungen

Überschneidungen

Die Leistungspunkte aus dieser Veranstaltung können nicht zusammen angerechnet werden mit denen aus *BA-INF 111 – Web-Technologien und Information Retrieval*, zuletzt im Wintersemester 2010/11.

2 XML-Dokumente

Literatur

Elliotte R. Harold, W. Scott Means.
XML in a Nutshell.
3. Auflage, O'Reilly, englisch (2004) oder deutsch (2005).
Siehe auch http://www.cafeconleche.org/books/xian3/

Charles F. Goldfarb and Paul Prescod.
Charles F. Goldfarb's XML Handbook, Fifth Edition.
Prentice Hall, 2003. Z. Z. leider vergriffen. Siehe auch www.xmlhandbook.com
(Vorsicht: nicht mehr aktuell)

2.1 Einleitung

Motivation und Hintergründe

Anwendungsgebiete (Repräsentation von Inhalten)

- XML für Dokumente
- XML für Daten
- XML für das WWW

Anwendungsgebiete (Verarbeitung)

- Transport
- Speicherung
- Rechnen, Transformieren

Anforderungen

- Autoren: (relativ) freie Textform
- Programmierer: rigide Strukturen
- Trennung von Inhalt und Layout

XML als Dokumentenformat

- Bücher, Artikel, technische Dokumentationen, Verträge, Webseiten, ...
- Dokumente sind *semistrukturiert*: Flexibilität bei Reihenfolge, Tiefe, Größe, Anzahl der Elemente
- Beispiele : TEI, DocBook, OpenDocument, XHTML, ...
- XML-Dokumente sind sowohl von Menschen als auch von Maschinen lesbar.

Das Erbe von SGML

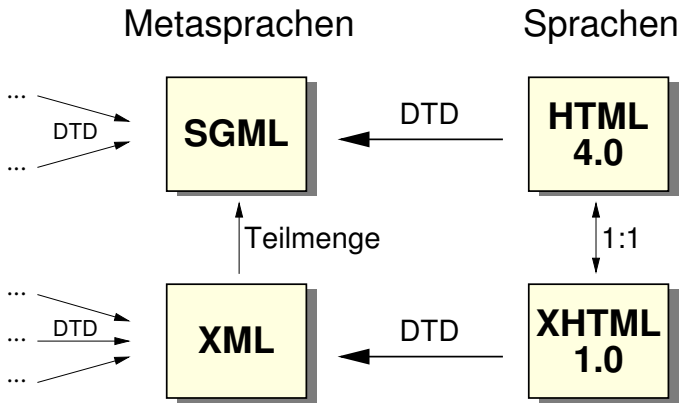
SGML (Standard Generalized Markup Language)

- ab 1986 ISO-Standard
- Einsatz z. B. im *Large-Scale Electronic Publishing*
- Trennung von Inhalt und Formatierung (*Declarative Markup*)
- ein Quelltext ⇒ mehrere Ausgabemedien (*Single Source*)
- Metasprache u. a. für *HTML*

Sprachen und Metasprachen (1)

- XML ist *eine* Sprache
- XML definiert *viele* Sprachen
- XML ist eine *gemeinsame Syntax* für viele Sprachen
- XML enthält einen *Definitionsmechanismus* für Sprachen (DTD)

Beispiel 2.1.



Ein einfaches Beispiel

Beispiel 2.2 (XHTML).

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html
    PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
           "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html lang="en" xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>Virtual Library</title>
  </head>
  <body>
    <p>Moved to <a href="http://example.org/">example.org</a>.</p>
  </body>
</html>
```

Ein einfaches Beispiel

Beispiel 2.3 (HTML5).

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>Virtual Library</title>
  </head>
  <body>
    <p>Moved to <a href="http://example.org/">example.org</a>.</p>
  </body>
</html>
```

HTML5 und XML

HTML5 kann (muss aber nicht) in XML-Syntax vorliegen.

Sprachen und Metasprachen (2)

- XML ist eine *Metasprache*
- XML erlaubt die Definition neuer anwendungsbezogener Sprachen
- SGML ist auch eine Metasprache, aber sehr komplex
- Beobachtung: Eine kleine Teilmenge von SGML reicht für fast alle Anwendungen aus.
- Das Ziel der XML-Entwicklung wurde erreicht:
- XML ist leichter zu erlernen.
- XML ist leichter zu implementieren.

2.2 XML-Strukturen

XML-Strukturen

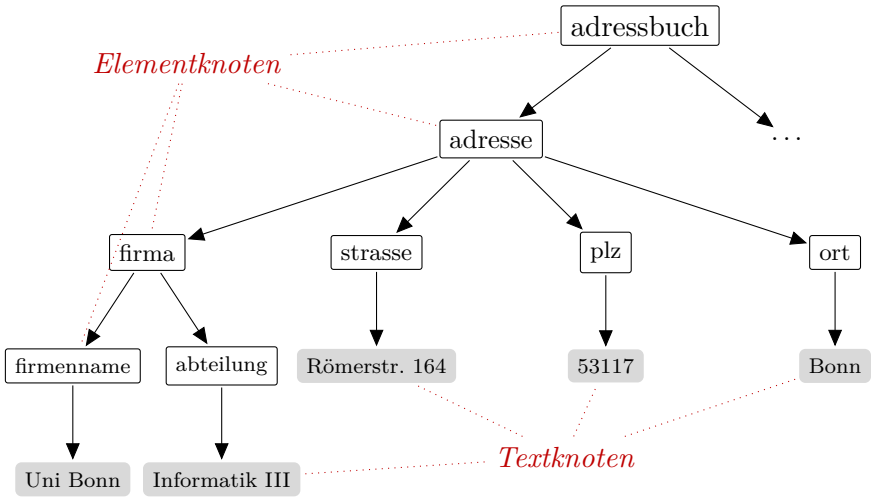
Konzept — wichtig!

- Ein XML-Dokument ist ein Baum!

Dokumente sind hierarchisch strukturiert

- *Elemente* strukturieren Dokumente.
- Elemente können wiederum *Subelemente* enthalten.
- Elemente können *Textknoten* enthalten.
- Elemente können *Attribute* haben.

XML als Baum



Datenstrukturen in XML

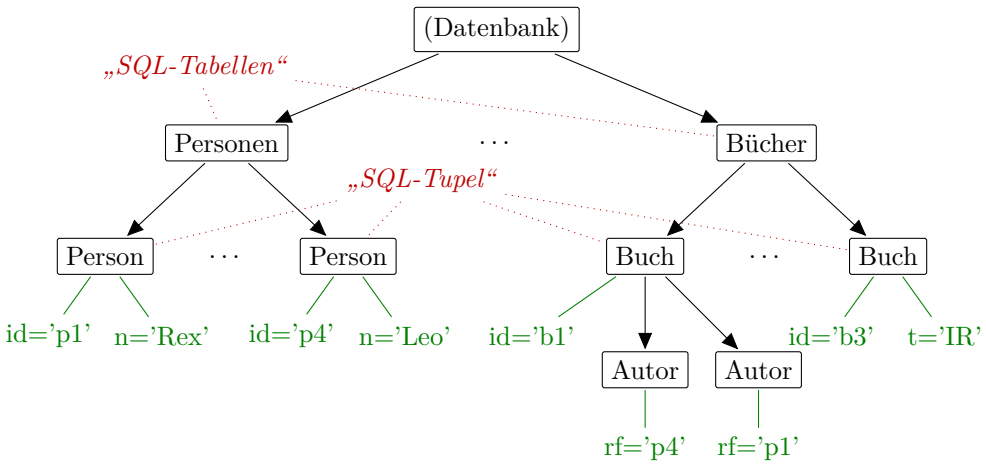
Abbildung von Datenstrukturen auf Bäume

- Viele Datenstrukturen können durch Bäume dargestellt werden.
- Beispiele folgen auf den nächsten Folien.
- Das *Hierarchische Datenbankmodell* basiert ebenfalls auf Bäumen.

Das hierarchische Datenbankmodell

Das hierarchische Datenbanksystem IMS[1] (seit ca. 1977) von IBM wird noch immer weiterentwickelt und ist sogar u. a. um XQuery- und SOA-Fähigkeiten erweitert worden, siehe z. B. http://www.heise.de/ix/news/meldung/97275.

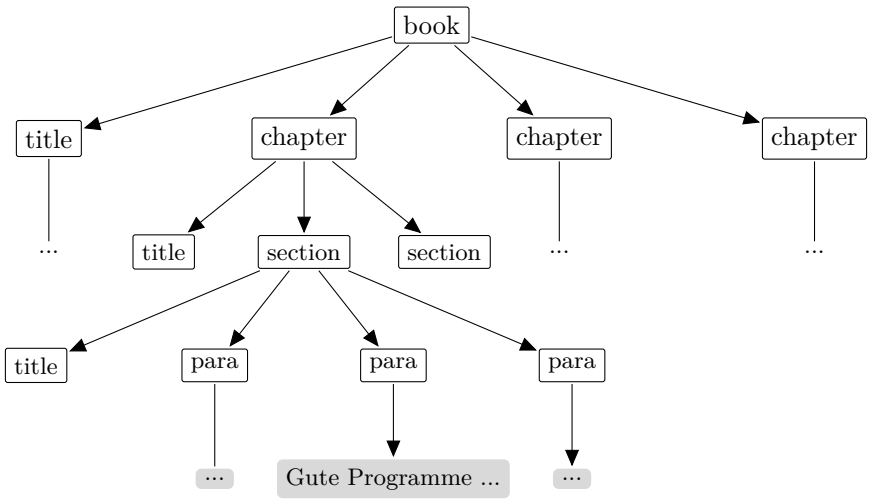
Beispiel: relationale Datenbank als Baum



Wir sind hier aber nicht auf die First-Normal-Form beschränkt.
Es sind z. B. auch Mengen oder Listen als Werte darstellbar.

Achtung: Es geht hier um Modellierungsaspekte und um mögliche Export- und Transportformate für Datenbankinhalte. Wir wollen nicht etwa die relationalen Datenbanken ersetzen!

Beispiel: semistrukturierte Dokumente (1)

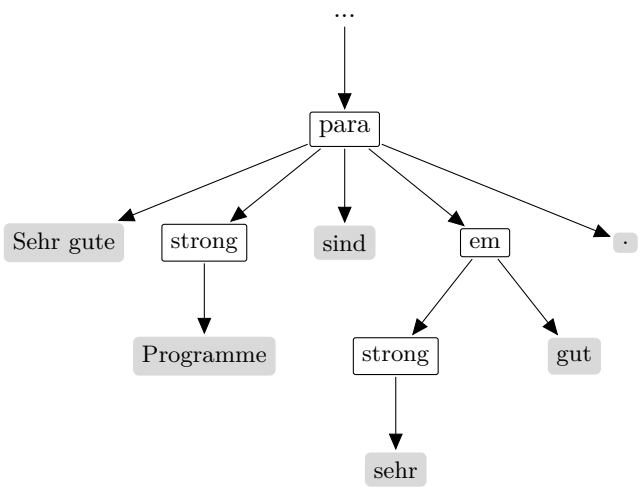


Die oberen Ebenen eines Buches sind stark strukturiert.

[1] http://www-306.ibm.com/software/data/ims/

Beispiel: semistrukturierte Dokumente (2)

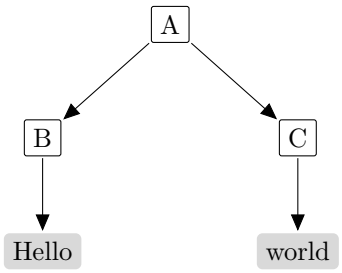
„Sehr gute *Programme* sind *sehr gut*.“



Im Fließtext gibt es variable Strukturen.

XML-Datenmodelle

Ein XML-Dokument ist ein Baum



- *Serialisierung*: XML-Syntax
- *Mathematisches Modell*: XML Information Set
- *Progammierschnittstelle (API)*: DOM

XML Information Set (Second Edition)
W3C Recommendation 4 February 2004
http://www.w3.org/TR/xml-infoset

Document Object Model (DOM)
http://www.w3.org/DOM/

2.3 XML Spezifikationen

Das World Wide Web Consortium (W3C)

W3C

- Gegründet im Oktober 1994 durch Tim Berners-Lee, den „Erfinder des WWW“
- Heute mehr als 300 Mitglieder: Industrie, Forschungsinstitute und Universitäten

Was macht das W3C?

- *. . . lead the World Wide Web to its full potential by developing common protocols that promote its evolution and ensure its interoperability.*
- Sitemap:
  http://www.w3.org/Consortium/siteindex.html
- Bisherige Empfehlungen:
  http://www.w3.org/standards/

W3C — Spezifikationsprozess

1. Working Draft (WD)
2. Last Call Working Draft
3. Candidate Recommendation (CR)
4. Proposed Recommendation (PR)
5. *Recommendation* (REC)

Daneben Notes und Requirements

XML Spezifikationen

Extensible Markup Language (XML) 1.0

- Extensible Markup Language (XML) 1.0 W3C Recommendation 10. Februar 1998
- XML 1.0 (Fifth Edition)[2] W3C Recommendation 26. November 2008

Extensible Markup Language (XML) 1.1

- Extensible Markup Language (XML) 1.1 W3C Recommendation 4. Februar 2004
- Gegenüber XML 1.0 nur kleinere Änderungen bzgl. Unicode-Verarbeitung
- XML 1.1 (Second Edition)[3] W3C Recommendation 16. August 2006

[2] https://www.w3.org/TR/xml/
[3] https://www.w3.org/TR/xml11/

Entwurfsziele für XML

[Aus der XML-1.0-Spezifikation in der Übersetzung von Mintert und Behme]

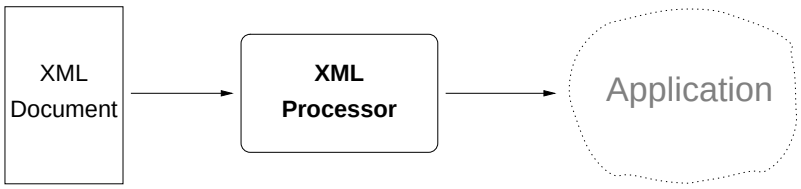
1. XML soll sich im Internet auf einfache Weise nutzen lassen.
2. XML soll ein breites Spektrum von Anwendungen unterstützen.
3. XML soll zu SGML kompatibel sein.
4. Es soll einfach sein, Programme zu schreiben, die XML-Dokumente verarbeiten.
5. Die Zahl optionaler Merkmale in XML soll minimal sein, idealerweise Null.
6. XML-Dokumente sollten für Menschen lesbar und angemessen verständlich sein.
7. Der Entwurf von XML soll formal und präzise sein.
8. XML-Dokumente sollen leicht zu erstellen sein.
9. Knappheit von XML-Markup ist von minimaler Bedeutung.
10. Der XML-Entwurf sollte zügig abgefasst sein.

Aufbau der XML-Definition

Logische Struktur Deklarationen, Elemente, Kommentare, Verarbeitungsanweisungen, usw.

Physische Struktur Zusammensetzen eines Dokuments aus *Entities*

XML-Prozessor (Parser) Verhalten, Umfang der Informationsweitergabe, Fehlerbehandlung, ...



2.4 Grundlagen

2.4.1 Syntax

XML — Syntax (1)

Serialisierung

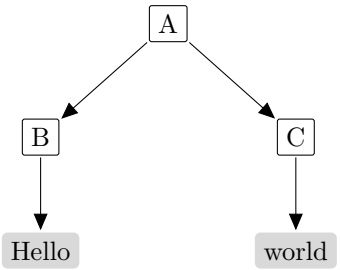
- Konkrete Syntax, um die Struktur zu *serialisieren*?

- Traversierung des Baumes
- *Tags* begrenzen die Elemente.
- kombinierte *Pre-Order*- und *Post-Order*-Traversierung
  - *Start-Tags* werden Pre-Order erzeugt
  - *End-Tags* werden Post-Order erzeugt

Die Baumstruktur wird also durch die *Verschachtelung* der Tags realisiert.

Beispiel: Traversierung

XML-Baum



XML-Text

```
<A>
  <B>Hello</B>
  <C>world</C>
</A>
```

Beispiel: XML-Dokument

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE adressbuch SYSTEM "abook.dtd">

<adressbuch>
  <adresse>
    <firma>
      <firmenname>Uni Bonn</firmenname>
      <abteilung>Informatik III</abteilung>
    </firma>
    <strasse>Römerstraße 164</strasse>
    <plz>53117</plz>
    <ort>Bonn</ort>
  </adresse>

  <adresse>
    ...
  </adresse>
</adressbuch>
```

XML — Syntax (2)

Formale Syntaxdefinition

1. Kontextfreie *Grammatik* (EBNF-ähnlich)
2. Frei formulierte *Zusatzbedingungen* (Constraints)

XML-Grammatik (Auszug)

```
 [1] document ::= ( prolog element Misc* )
                  - ( Char* RestrictedChar Char* )
[39] element ::= EmptyElemTag | STag content ETag
[40] STag ::= '<' Name (S Attribute)* S? '>'
[42] ETag ::= '</' Name S? '>'
```

Well-formedness Constraint (Beispiel)

Element Type Match The `Name` in an element's end-tag MUST match the element type in the start-tag.

Validity Constraint (Beispiel)

Element Valid An element is valid if there is a declaration matching `elementdecl` where the `Name` matches the element type, and one of the following holds: ...

XML — Syntax (3)

Lexikalische Grundlage

- **Zeichensatz:** Unicode (mehr dazu später)
- **Namen**: Buchstaben, Ziffern, Sonderzeichen `. - _ :`

Reservierte Zeichen

Die Zeichen `<` und `&` müssen außerhalb von Markup, Kommentaren usw. durch die vordefinierten *Entity-Referenzen* `<` bzw. `&` dargestellt werden.

Beispiel 2.4. Falsch:

```
a < b && a > 0
```

Richtig:

```
a &lt; b &amp;&amp; a > 0
```

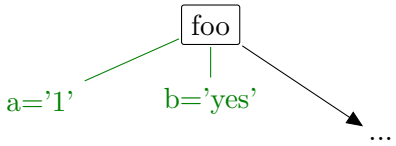
2.4.2 Elemente und Attribute

Syntax für Elemente und Attribute

```
[39] element ::= EmptyElemTag | STag content ETag
[40] STag ::= '<' Name (S Attribute)* S? '>'
[41] Attribute ::= Name Eq AttValue
[42] ETag ::= '</' Name S? '>'
```

- *Elemente* werden durch *Tags* begrenzt.
- Alle *Attribute* werden im Start-Tag gelistet.
- Die Reihenfolge der Attribute ist nicht relevant.
- Jeder Attributname kommt maximal einmal vor.

Beispiel 2.5.



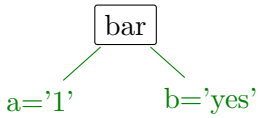

```
<foo a='1' b='yes'>...</foo>
```

Syntax für leere Elemente

```
 [3] S ::= (#x20 | #x9 | #xD | #xA)+
[25] Eq ::= S? '=' S?
[41] Attribute ::= Name Eq AttValue
[44] EmptyElemTag ::= '<' Name (S Attribute)* S? '/>'
```

- Für leere Elemente gibt es eine spezielle Syntax.
- Leere Elemente können Attribute haben, aber keine Kindknoten.

Beispiel 2.6.




```
<bar a='1' b='yes'/>        <bar b='yes' a="1" />

<bar a='1' b="yes"></bar>
```

- Alle drei Schreibweisen sind völlig äquivalent.

2.4.3 Unicode

Literatur

Unicode Home Page[4]
Unicode Character Database[5]

Unicode-Zeichensatz

- ISO/IEC 10646 und Unicode definieren einen universellen Zeichensatz.
- *Universal Character Set (UCS)*
- Bis zu 2^{16} bzw. 2^{32} Zeichen (UCS-2 bzw. UCS-4)
- `Links`
  - Datenbank aller Zeichen[6]
  - Code Charts[7]

Unicode-Zeichenkategorien

Die Zeichen sind mittels verschiedener *Kategorien* klassifiziert:

`Lu`	Letter, Uppercase ABCÄÖÜ	`Pi`	Punctuation, Initial quote
`Ll`	Letter, Lowercase abcdéαβγ	`Pf`	Punctuation, Final quote
`Lt`	Letter, Titlecase	`Po`	Punctuation, Other ! " % &
`Lm`	Letter, Modifier	`Sm`	Symbol, Math + < = >
`Lo`	Letter, Other	`Sc`	Symbol, Currency $
`Mn`	Mark, Nonspacing	`Sk`	Symbol, Modifier
`Mc`	Mark, Spacing Combining	`So`	Symbol, Other
`Me`	Mark, Enclosing	`Zs`	Separator, Space ␣
`Nd`	Number, Decimal Digit 01234	`Zl`	Separator, Line
`Nl`	Number, Letter	`Zp`	Separator, Paragraph
`No`	Number, Other	`Cc`	Other, Control
`Pc`	Punctuation, Connector _	`Cf`	Other, Format
`Pd`	Punctuation, Dash	`Cs`	Other, Surrogate
`Ps`	Punctuation, Open ([{	`Co`	Other, Private Use
`Pe`	Punctuation, Close)] }	`Cn`	Other, Not Assigned

Ein Zeichensatz – viele Kodierungen

Probleme

- Die Unicode-Zeichen benötigen i. Allg. 32 Bit.

[4] http://www.unicode.org/
[5] http://www.unicode.org/ucd/
[6] http://www.unicode.org/Public/UNIDATA/UnicodeData.txt
[7] http://www.unicode.org/charts/index.html

- Speicherung braucht bis zu 4x mehr Platz als nötig.

Die Lösung

Kodierung: Abbildung von Unicode-Zeichen auf Byte-Folgen

Einige vollständige Kodierungen

- ISO-10646-UCS-4 (Identische Kodierung)
- UTF-8, UTF-16

Einige unvollständige Kodierungen

- ASCII
- ISO-8859-1 (Latin-1, westeuropäisch), . . . , ISO-8859-16
- Codepage 1252, . . .

UCS Transformation Formats (UTF)

8-Bit- / 16-Bit-Architekturen: UTF-8 (RFC 2279[8]) / UTF-16

UTF-8 Kodierung

UCS-4 range (hex.)	UTF-8 octet sequence (binary)
00000000-0000007F	*0xxxxxxx*
00000080-000007FF	*110xxxxx 10xxxxxx*
00000800-0000FFFF	*1110xxxx 10xxxxxx 10xxxxxx*
00010000-001FFFFF	11110xxx 10xxxxxx 10xxxxxx 10xxxxxx
00200000-03FFFFFF	111110xx 10xxxxxx ... 10xxxxxx (5 Bytes)
04000000-7FFFFFFF	1111110x 10xxxxxx ... 10xxxxxx (6 Bytes)

Beispiel 2.7.

UCS: `A<NOT IDENTICAL TO><ALPHA>.` Glyphen: $A \not\equiv A.$
Hex.: (0041, 2262, 0391, 002E)

2262 → 0010 0010 0110 0010 → 1110 0010 1000 1001 1010 0010 → E2 89 A2

0391 → 0000 0011 1001 0001 → 1100 1110 1001 0001 → CE 91

UTF-8: `41 E2 89 A2 CE 91 2E`

[8] http://tools.ietf.org/html/rfc2279

UTF-8

Eigenschaften von UTF-8

- Kodierung in 1–3 Bytes (UCS-2), bzw. 1–6 Bytes (UCS-4)
- Die 7 Bit US-ASCII-Zeichen bleiben erhalten.
- Die US-ASCII-Zeichen kommen ansonsten als Bytes nicht vor ⇒ Kompatibilität mit Filesystemen und Software.
- Zeichengrenzen sind leicht zu finden.
- Die Konvertierung ist einfach.

UTF-16 (RFC 2781[9]) ähnlich, aber mit 16-Bit-Wörtern anstelle der Bytes

2.4.4 Zeichen, Namen, Texte

Unicode-Zeichenklassen in XML 1.1

Erlaubte Zeichen

```
 [1] document ::= ( prolog element Misc* )
                  - ( Char* RestrictedChar Char* )

 [2] Char ::= [#x1-#xD7FF] | [#xE000-#xFFFD] | [#x10000-#x10FFFF]

[2a] RestrictedChar ::= [#x1-#x8] | [#xB-#xC] | [#xE-#x1F]
                        | [#x7F-#x84] | [#x86-#x9F]
```

Bestimmte Unicode-Zeichen sind in XML-Dokumenten nicht erlaubt.

Namen

Namen in XML

```
 [4] NameStartChar ::= ":" | [A-Z] | "_" | [a-z] | [#xC0-#xD6]
                           | [#xD8-#xF6] | [#xF8-#x2FF]
                           | [#x370-#x37D] | [#x37F-#x1FFF]
                           | [#x200C-#x200D] | [#x2070-#x218F]
                           | [#x2C00-#x2FEF] | [#x3001-#xD7FF]
                           | [#xF900-#xFDCF] | [#xFDF0-#xFFFD]
                           | [#x10000-#xEFFFF]

[4a] NameChar ::= NameStartChar | "-" | "." | [0-9] | #xB7
```

[9] http://tools.ietf.org/html/rfc2781

```
                          | [#x0300-#x036F] | [#x203F-#x2040]

 [5] Name ::= NameStartChar (NameChar)*
 [6] Names ::= Name (#x20 Name)*
```

Alle Unicode-Buchstaben, *nicht nur A–Z, a–z*

Beispiele 2.8. `book` `Book:` `_book` `::__` `a.b` `a-b` `a0` `b1.2` `html:body`

Werte

Attributwerte und Referenzen

```
[10] AttValue ::=   '"' ([^<&"] | Reference)* '"'
                  | "'" ([^<&'] | Reference)* "'"

[66] CharRef ::= '&#' [0-9]+ ';' | '&#x' [0-9a-fA-F]+ ';'

[67] Reference ::= EntityRef | CharRef

[68] EntityRef ::= '&' Name ';'
```

Beispiele 2.9 (Attributwerte). `"ab"` `'ab'` `"sag: 'ja'"` `"ALPHA: &#x391;"` `"<book>"`

Character Data, CDATA Sections

Character Data

```
[14] CharData ::= [^<&]* - ([^<&]* ']]>' [^<&]*)
```

- Verwendung in Textknoten
- die Zeichen `<` und `&` dürfen nicht auftreten
- die Zeichenkette `]]>` darf nicht auftreten

CDATA Sections

```
[18] CDSect  ::= CDStart CData CDEnd
[19] CDStart ::= '<![CDATA['
[20] CData   ::= (Char* - (Char* ']]>' Char*))
[21] CDEnd   ::= ']]>'
```

Beispiel 2.10 (CDATA Section). `<![CDATA[<greeting>Hello, world!</greeting>]]>`

Beispiel 2.11 (äquivalent als Character Data). `<greeting>Hello, world!</greeting>`

Kommentare, Verarbeitungsanweisungen

Kommentare

```
[15] Comment ::= '<!--' ((Char - '-') | ('-' (Char - '-')))* '-->'
```

Beispiel 2.12 (Kommentar). `<!-- declarations for <head> & <body> -->`

Der Parser *darf*, muss aber nicht, den Inhalt von Kommentaren an die Anwendung weitergeben.

Verarbeitungsanweisungen

```
[16] PI ::= '<?' PITarget (S (Char* - (Char* '?>' Char*)))? '?>'
[17] PITarget ::= Name - (('X' | 'x') ('M' | 'm') ('L' | 'l'))
```

Beispiele 2.13 (Verarbeitungsanweisungen). `<?foo?>`

```
<?foo ...?>
<?test asdf gh jkl!" §$% &/()=?>
<?xml-stylesheet type="text/xml" href="abook.xsl"?>
```

Inhalt von Elementen

```
[39] element ::= EmptyElemTag | STag content ETag
[43] content ::= CharData? ((element | Reference
                | CDSect| PI | Comment) CharData?)*
```

Beispiel 2.14 (Inhalt).

```
<a>
  <b>Copyright &#xA9; 2007</b>
  <!-- Test -->
  <c/>
  &rights;
</a>
```

2.4.5 Wohlgeformtheit

Prolog, XML-Deklaration

Beispiel 2.15 (XML 1.1).

```
<?xml version="1.1" encoding="ISO-8859-1"?>
<greeting>Viele Grüße</greeting>
```

Beispiel 2.16 (XML 1.1, UTF-8).

```
<?xml version="1.1" ?>
<greeting>Viele Grüße</greeting>
```

Beispiel 2.17 (XML 1.0, UTF-8).

```
<greeting>Viele Grüße</greeting>
```

Prolog, XML-Deklaration

Top-Level-Grammatik

```
 [1] document ::= ( prolog element Misc* )
                - ( Char* RestrictedChar Char* )

[22] prolog ::= XMLDecl Misc* (doctypedecl Misc*)?
[23] XMLDecl ::= '<?xml' VersionInfo EncodingDecl? SDDecl? S? '?>'
[24] VersionInfo ::= S 'version' Eq ("'" VersionNum "'" | '"' VersionNum '"')
[26] VersionNum ::= '1.1'
[27] Misc ::= Comment | PI | S

[80] EncodingDecl ::= S 'encoding' Eq ('"' EncName '"' | "'" EncName "'" )
[81] EncName ::= [A-Za-z] ([A-Za-z0-9._] | '-')*
```

- Es gibt genau ein Wurzelelement. (Baum!)
- Fehlt die XML-Deklaration, ist es XML 1.0
- Fehlt die Encoding-Deklaration, muss die Kodierung UTF-8 sein

XML-Prozessoren

- Alle XML-Prozessoren müssen UTF-8 und UTF-16 verarbeiten können.
- Entities in UTF-16 müssen mit `#xFEFF` (Byte Order Mark) beginnen.
- Andere Kodierungen können unterstützt werden.

Wohlgeformtheit

Definition 2.18 (Wohlgeformte XML-Dokumente)**.** Ein textuelles Objekt ist ein *wohlgeformtes XML-Dokument* (well-formed XML document), wenn

1. es als Gesamtheit betrachtet aus dem Symbol `document` ableitbar ist,
2. es alle Wohlgeformtheits-Constraints der XML-Spezifikation erfüllt, und
3. jedes seiner *Parsed Entities*, welches direkt oder indirekt referenziert wird, wohlgeformt ist.

Wesentliche Punkte:

- *„Klammerstruktur“*
- Genau ein *Wurzelelement*

2.5 Document Type Definition (DTD)

Dokumenttypen, Vokabulare

Syntax und Vokabulare

- Anwendungen „verstehen“ nur spezifische Sprachen
- XML-Sprachen haben eine gemeinsame *Grundsyntax*
- Das *Vokabular* (Element- und Attributnamen) wird separat festgelegt.

Definition des Vokabulars

- Implizites Vokabular (Parser prüft nur auf Wohlgeformtheit) *oder*
- Document Type Definition (DTD) *oder*
- XML Schema

Dokumenttyp per DTD

Document Type Declaration

- steht vor dem ersten Element des Dokuments
- verweist auf eine *Document Type Definition (DTD)* (intern und/oder extern)

Document Type Definition (DTD)

- Elementdeklarationen: Welche *Elemente*, wo und in welcher Reihenfolge?
- Attributdeklarationen: Welche *Attribute* sind zulässig oder notwendig?
- *Entity*- und Notation-Deklarationen

2.5.1 Gültigkeit

Gültige Dokumente

Definition 2.19 (Gültige Dokumente)**.** Ein wohlgeformtes XML-Dokument, das eine *Document Type Declaration* besitzt und den dort spezifizierten Einschränkungen genügt, heißt gültig (valid).

Beispiele 2.20.

- Nicht wohlgeformt ⇒ nicht gültig
- Keine DTD angegeben ⇒ nicht gültig
- Gültig ⇒ wohlgeformt

Document Type Declaration und DTD

```
[28] doctypedecl ::= '<!DOCTYPE' S Name (S ExternalID)?
                     S? ('[' intSubset ']' S?)? '>'

[28a] DeclSep ::= PEReference | S

[28b] intSubset ::= (markupdecl | DeclSep)*

[29] markupdecl ::= elementdecl | AttlistDecl
             | EntityDecl | NotationDecl | PI | Comment
```

- Dazu gehören einige *Well-formedness Constraints* und *Validity Constraints*

Beispiel 2.21 (Validity constraint: Root Element Type). The Name in the document type declaration must match the element type of the root element.

Beispiel 2.22 (Beispiel: externe DTD).

```
<?xml version="1.0"?>
<!DOCTYPE greeting SYSTEM "hello.dtd">
<greeting>Hello, world!</greeting>
```

Beispiel 2.23 (Beispiel: externe DTD).

```
<?xml version="1.0"?>
<!DOCTYPE html
  PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">...</html>
```

Beispiel 2.24 (interne DTD).

```
<?xml version="1.0"?>
<!DOCTYPE greeting [
  <!ELEMENT greeting (#PCDATA)>
]>
<greeting>Hello, world!</greeting>
```

Beispiel 2.25 (weder interne noch externe DTD).

```
<?xml version="1.0"?>
<!DOCTYPE greeting>
<greeting>Hello, world!</greeting>
```

2.5.2 Elemente

Elementtyp-Deklarationen

```
[45] elementdecl ::= '<!ELEMENT' S Name S contentspec S? '>'
[46] contentspec ::= 'EMPTY' | 'ANY' | Mixed | children
```

- Elemente dürfen höchstens einmal deklariert werden.

Beispiele 2.26.

```
<!ELEMENT br EMPTY>
<!ELEMENT container ANY>
<!ELEMENT doc (header, body, footer) >
```

Arten von Content

- *Element Content*: nur Elemente
- *Mixed Content*: auch Character Data
- *EMPTY*: jeglicher Inhalt verboten
- *ANY*: beliebiger Inhalt erlaubt

Elementtyp-Deklarationen — *Element Content*

```
[47] children ::= (choice | seq) ('?' | '*' | '+')?
[48] cp       ::= (Name | choice | seq) ('?' | '*' | '+')?
[49] choice   ::= '(' S? cp ( S? '|' S? cp )+ S? ')'
[50] seq      ::= '(' S? cp ( S? ',' S? cp )* S? ')'
```

Beispiele 2.27 (DTD).

```
<!ELEMENT spec (front, body, back?)>
<!ELEMENT div1 (head, (p | list | note)*, div2*)>
```

Beispiel 2.28 (Instanz).

```
<div1>
  <head>Hello World</head>
  <p>Some text...</p><list>...</list>
  <p>More text...</p>
</div1>
```

- Whitespace zwischen den Elementen ist erlaubt

Elementtyp-Deklarationen — Eindeutigkeit

Determinismus

- Inhaltsmodelle ohne *Look Ahead* zu parsen

Beispiel 2.29. `<a><b>...</b><d>...</d></a>`
`<a><b>...</b><c>...</c></a>`

Beispiel DTD (falsch!)

```
<!ELEMENT a ((b, c) | (b, d))>
```

Beispiel 2.30 (richtig). `<!ELEMENT a (b, (c | d))>`

Elementtyp-Deklarationen — *Mixed Content*

```
[51] Mixed ::= '(' S? '#PCDATA' (S? '|' S? Name)* S? ')*'
               | '(' S? '#PCDATA' S? ')'
```

Beispiele 2.31 (DTD).

```
<!ELEMENT p  (#PCDATA|a|ul|b|i|em)*>
<!ELEMENT em (#PCDATA)>
```

Beispiel 2.32 (Instanz).

```
<p>Hello <em>World</em>.</p>
```

2.5.3 Attribute

Attributlisten

Attributlisten

```
[52] AttlistDecl ::= '<!ATTLIST' S Name AttDef* S? '>'
[53] AttDef      ::= S Name S AttType S DefaultDecl
```

Attributtypen

```
[54] AttType       ::= StringType | TokenizedType | EnumeratedType
[55] StringType    ::= 'CDATA'
```

Beispiel 2.33 (Definition).

```
<!ATTLIST artikel
          nummer      CDATA  #IMPLIED
          lagerplatz  CDATA  #IMPLIED>
```

Beispiel 2.34 (Instanzen).

```
<artikel/> <artikel nummer="42"/>
<artikel nummer="49" lagerplatz="C3A76"/>
```

Attributtypen — Token-Typen

```
[56] TokenizedType ::= 'ID' | 'IDREF' | 'IDREFS'
                     | 'ENTITY' | 'ENTITIES'
                     | 'NMTOKEN' | 'NMTOKENS'
```

```
[5]    Name  ::= NameStartChar (NameChar)*
[6]    Names  ::= Name (#x20 Name)*

[7]    Nmtoken  ::= (NameChar)+
[8]    Nmtokens ::= Nmtoken (#x20 Nmtoken)*
```

Validity Constraints (Auswahl)

- Werte vom Typ `ID` und `IDREF`: ableitbar aus `Name`
- Werte vom Typ `IDREFS`: ableitbar aus `Names`
- Ein Wert vom Typ `ID` darf max. einmal im *Dokument* vorkommen.
- Der Typ `ID` darf pro Element max. einmal vorkommen.
- Der Wert eines `IDREF`-Attributs muss in einem `ID`-Attribut eines Elements im Dokument vorkommen.
- Analog alle Werte eines `IDREFS`-Attributs
- Werte vom Typ `'NMTOKEN'`: ableitbar aus `Nmtoken`
- Werte vom Typ `'NMTOKENS'`: ableitbar aus `Nmtokens`

Attributtypen — Token-Typen

Beispiele 2.35 (DTD).

```
<!ATTLIST para xml:lang NMTOKEN #IMPLIED>


<!ATTLIST motor
          nummer        ID      #REQUIRED
          lieferant     IDREF   #REQUIRED
          auto-modelle  IDREFS  #IMPLIED>
```

Beispiel 2.36 (Instanz).

```
<motor nummer="M13462" lieferant="L43" auto-modelle="A1 A7 A58"/>
<motor nummer="M13471" lieferant="L2"/>
```

Validity Constraints: die Werte `L43`, `A1`, `A7`, `A58` und `L2` müssen im selben Dokument in Attributen des Typs `ID` definiert sein.

```
<zulieferer zcode="L43">...</zulieferer>
...
```

Aufzählungstypen

```
[57] EnumeratedType ::= NotationType | Enumeration
[59] Enumeration    ::= '(' S? Nmtoken (S? '|' S? Nmtoken)* S? ')'
```

Beispiel 2.37.

```
<!ATTLIST list type (bullets|ordered|glossary) #IMPLIED>
```

Beispiel 2.38.

```
<list type="ordered">...</list>
```

Defaultwerte von Attributen

```
[60] DefaultDecl ::= '#REQUIRED' | '#IMPLIED' | (('#FIXED' S)? AttValue)
```

Beispiele 2.39.

```
<!ATTLIST termdef
          id      ID      #REQUIRED
          name    CDATA   #IMPLIED>

<!ATTLIST form method CDATA #FIXED "POST">
```

Normalisierung von Attributwerten

Normalisierung durch den XML-Prozessor (alle Typen):

- Character-Referenzen durch Zeichen ersetzen
- Entity-Referenzen rekursiv durch Text ersetzen
- *Whitespace*-Zeichen durch `#x20` ersetzen, aber `#xD#xA` durch ein einzelnes `#x20`

Zusätzlich für alle Typen außer `CDATA`:

- `#x20` am Anfang und Ende des Wertes werden entfernt
- Folgen von `#x20` durch ein einzelnes `#x20` ersetzen

Beispiel 2.40.

`a="␣␣&#65;`

`B␣␣␣C␣"`

wird zu `a="␣␣A␣␣B␣␣␣C␣"` beim Typ `CDATA`, und zu `a="A␣B␣C"` beim Typ `NMTOKENS`

2.5.4 Entities

Entities — Begriffe

Entity separat gespeicherter Teil des Dokuments

Document Entity Startpunkt des XML-Prozessors

Parsed Entity Ersetzungstext, Teil des Dokuments; Zugriff über Entity-Referenzen, z. B. `&anhang;`

Unparsed Entity Beliebige externe (binäre) Daten. Nur die Namen werden an die Anwendung weitergereicht.

Parameter Entity Parsed Entity zur Verwendung innerhalb der DTD.

- Fast alle Entities haben Namen, außer dem *Dokument-Entity* und dem externen DTD-Teil.

Character- und Entity-Referenzen

Beispiel 2.41 (Character- und Entity-Referenzen).

```
Type <key>less-than</key> (&#x3C;) to save options.
This document was prepared on &docdate; and
is classified &security-level;.
```

Beispiel 2.42 (Parameter-Entity Referenz).

```
<!-- declare the parameter entity "ISOLat2"... -->
<!ENTITY % ISOLat2 SYSTEM "http://www.xml.com/iso/isolat2-xml.ent" >

<!-- ... now reference it. -->
%ISOLat2;

<!-- parameter entity references in an element declaration -->
<!ELEMENT %name.para; %content.para; >
```

Interne Entities

```
[70] EntityDecl ::= GEDecl | PEDecl
[71]   GEDecl ::= '<!ENTITY' S Name S EntityDef S? '>'
[72]   PEDecl ::= '<!ENTITY' S '%' S Name S PEDef S? '>'
[73]  EntityDef ::= EntityValue | (ExternalID NDataDecl?)
[74]      PEDef ::= EntityValue | ExternalID
```

Interne Entities

- Der Rumpf ist ein `EntityValue` (ähnlich `AttValue`).
- Der gesamte Inhalt ist bereits in der Deklaration enthalten.
- Das interne Entity ist ein *Parsed Entity*.

Beispiel 2.43.

```
<!ENTITY Pub-Status "This is a pre-release of the specification.">
```

Externe Entities

```
[75]  ExternalID ::= 'SYSTEM' S SystemLiteral
             | 'PUBLIC' S PubidLiteral S SystemLiteral
```

Externe Entities

- Das `SystemLiteral` heißt *System Identifier*.
- System Identifier ist ein URI, über das das Entity geholt werden kann.
- Über den optionalen *externen Identifier* `PubidLiteral` kann der XML-Prozessor eine alternative URI generieren.

Beispiel 2.44.

```
<!ENTITY open-hatch SYSTEM "http://www.xml.com/boilerplate/OH.xml">
```

External Parsed Entities

Text Declaration

```
[77]  TextDecl ::= '<?xml' VersionInfo? EncodingDecl
                   S? '?>'
```

Well-Formed External Parsed Entity

```
[78]  extParsedEnt ::= TextDecl? content
```

Zum Vergleich

```
 [1] document ::= ( prolog element Misc* )
                  - ( Char* RestrictedChar Char* )
[39] element ::= EmptyElemTag | STag content ETag
[43] content ::= CharData? ((element | Reference
                 | CDSect| PI | Comment) CharData?)*
```

Beispiel 2.45 (External Parsed Entity).

```
<?xml version="1.0" encoding="UTF-8"?>
Hello world! <a/><b/>
```

Entities — Konstruktion des Ersetzungstexts

Beispiel 2.46 (Deklarationen).

```
<!ENTITY % pub    "&#xc9;ditions Gallimard" >
<!ENTITY rights "All rights reserved" >
<!ENTITY book   "La Peste: Albert Camus, &#xA9; 1947 %pub;. &rights;" >
```

Beispiel 2.47 (Ersetzungstext für Entity `book`).

```
La Peste: Albert Camus, © 1947 Éditions Gallimard. &rights;
```

Konforme XML-Prozessoren (Parser)

Nicht-validierende Prozessoren

- Wohlgeformtheit wird geprüft, alle Fehler gemeldet.
- Nur das Dokument-Entity muss gelesen werden, externe Entities dürfen gelesen werden.

- Fehler in externen Entities werden also nicht von allen nicht-validierenden Prozessoren gefunden.

Validierende Prozessoren

- Wohlgeformtheit und Gültigkeit werden geprüft, alle Fehler gemeldet.
- Dazu müssen die gesamte DTD und alle externen Parsed Entities gelesen und verarbeitet werden.
- Vorhersagbares Verhalten des validierenden Prozessors

2.6 XML Namespaces

Namespace-Spezifikationen

Namespaces in XML

- Namespaces in XML 1.0 (Third Edition)[10] W3C Recommendation 8. Dezember 2009

  (Originalspezifikation: Januar 1999)
- Namespaces in XML 1.1 (Second Edition)[11] W3C Recommendation 16. August 2006

- Namen in XML-Dokumenten sind *global*.
- *Namenskonflikte* beim Kombinieren verschiedener „Module“ zu komplexen XML-Dokumenten
- Namespaces: Qualifizierte Element- und Attributnamen in XML-Dokumenten
- Ein Software-Modul kann am Namespace diejenigen Elemente und Attribute erkennen, die es verarbeiten soll. (Beispiel: XHTML mit eingebettetem SVG)

Namespaces — Begriffe

XML-Namespace

- Sammlung von Element- und Attributnamen (nicht explizit dargestellt)
- URI (als weltweit eindeutiges Namensschema)

Idee

- Realisierung der Namespaces als disziplinierte eingeschränkte Verwendung von XML 1.0 oder 1.1

[10] https://www.w3.org/TR/xml-names/
[11] https://www.w3.org/TR/xml-names11/

Namespaces — Begriffe

Qualifizierter Name

- XML-Name,
- enthält genau einen Doppelpunkt (`:`),
- besteht aus *Namespace-Präfix*, dem Doppelpunkt (`:`) und einem *lokalen Teil*.
- Beispiele: *`html:h1`*, *`unibn:vorlesung`*

- Präfixe werden an Namespace-URIs *gebunden*
- Der Umweg über den Namespace-Präfix ist notwendig, weil URIs i. allg. keine zulässigen XML-Namen sind.

Beispiel 2.48.

```
<x xmlns:edi='http://ecommerce.org/schema'>
  <edi:price units='Euro'>32.18</edi:price>
</x>
```

Grammatik für XML mit Namespaces

Die Namespace-Spezifikation modifiziert und erweitert die XML-Grammatik (und Constraints), so dass eine *echte Teilsprache* von XML 1.1 definiert wird:

- Alle *Elementnamen* und alle *Attributnamen* enthalten entweder keinen oder genau einen Doppelpunkt (`:`), und
- alle Entity-Namen, PI-Targets und Notationsnamen enthalten keine Doppelpunkte (`:`).
- Verwendete Präfixe müssen deklariert sein.
- ...

Namespace-Deklaration

Präfix-Bindung

- Attribute mit dem speziellen Präfix *`xmlns`*
- Eine Namespace-Deklaration gilt bereits für das Element, in dessen Start-Tag sie erfolgt.
- Eine Namespace-Deklaration gilt auch für den gesamten Unterbaum, solange der Präfix nicht umdeklariert wird.

Beispiel 2.49.

```
<html:html xmlns:html='http://www.w3.org/1999/xhtml'>
  <html:head>...</html:head>
  <html:body>...</html:body>
</html:html>
```

Qualifizierte Namen

Der Präfix ist nur ein Platzhalter für den Namespace-URI. Applikationen sollen die Referenz auflösen, also mit dem Namespace-URI arbeiten, nicht mit dem Präfix.

Beispiel 2.50.

```
<p:game xmlns:p='http://foo.bar.com'>Skat</p:game>
<q:game xmlns:q='http://foo.bar.com'>Poker</q:game>
<game xmlns='http://foo.bar.com'>Doppelkopf</game>
```

Aus Anwendungssicht liegt hier dreimal ein Element `game` aus dem Namespace `http://foo.bar.com` vor.

Namespaces bei Attributen

Beispiel 2.51.

```
<x xmlns:edi='http://ecommerce.org/schema'>
  <lineItem edi:taxClass="exempt">Baby food</lineItem>
</x>
```

Anwendung von Namespaces

Beispiel 2.52 (mehrere Namespaces).

```
<?xml version="1.1"?>
<bk:book xmlns:bk='urn:loc.gov:books'
         xmlns:isbn='urn:ISBN:0-395-36341-6'>
    <bk:title>Cheaper by the Dozen</bk:title>
    <isbn:number>1568491379</isbn:number>
</bk:book>
```

Namespace Defaults

Ein *Default Namespace* gilt für alle Elemente, die keinen expliziten Namespace-Präfix besitzen.

Beispiel 2.53.

```
<?xml version="1.1"?>
<!-- elements are in the HTML namespace,
     in this case by default -->
```

```
<html xmlns='http://www.w3.org/1999/xhtml'>
  <head><title>Frobnostication</title></head>
  <body>
    <p>Moved to
      <a href='http://frob.example.com'>here</a>.
    </p>
  </body>
</html>
```

- Default Namespaces gelten nicht für Attributnamen!

Namespace Defaults

Beispiel 2.54.

```
<?xml version="1.1"?>
<!-- unprefixed element types are from "books" -->
<book xmlns='urn:loc.gov:books'
      xmlns:isbn='urn:ISBN:0-395-36341-6'>
    <title>Cheaper by the Dozen</title>
    <isbn:number>1568491379</isbn:number>
</book>
```

Namespace Defaults

Beispiel 2.55 (Umdeklaration).

```
<?xml version="1.1"?>
<!-- initially, the default namespace is "books" -->
<book xmlns='urn:loc.gov:books'
      xmlns:isbn='urn:ISBN:0-395-36341-6'>
    <title>Cheaper by the Dozen</title>
    <isbn:number>1568491379</isbn:number>
    <notes>
      <!-- make HTML the default namespace
           for some commentary -->
      <p xmlns='http://www.w3.org/1999/xhtml'>
          This is a <i>funny</i> book!
      </p>
    </notes>
</book>
```

Namespace Defaults

Beispiel 2.56 (leere Namespace-URI beim Default Namespace).

```
<?xml version='1.1'?>
<Beers>
 <!-- the default namespace inside tables is that of HTML -->
 <table xmlns='http://www.w3.org/1999/xhtml'>
  <th><td>Name</td><td>Origin</td><td>Description</td></th>
```

```
  <tr>
   <!-- no default namespace inside table cells -->
   <td><brandName xmlns="">Huntsman</brandName></td>
   <td><origin xmlns="">Bath, UK</origin></td>
   <td>
    <details xmlns=""><class>Bitter</class><hop>Fuggles</hop>
     <pro>Wonderful hop, light alcohol, good summer beer</pro>
     <con>Fragile; excessive variance pub to pub</con>
    </details>
   </td>
  </tr>
 </table>
</Beers>
```

Attribute dürfen nicht mehrfach auftreten!

Das gilt auch, wenn unterschiedliche Präfixe an denselben Namespace gebunden sind:

Beispiel (Fehler!)

```
<x xmlns:n1="http://www.w3.org"
   xmlns:n2="http://www.w3.org" >
  <bad a="1"    a="2" />
  <bad n1:a="1" n2:a="2" />
</x>
```

Beispiel 2.57 (korrekt).

```
<x xmlns:n1="http://www.w3.org"
   xmlns="http://www.w3.org" >
  <good a="1"  b="2" />
  <good a="1"  n1:a="2" />
</x>
```

(Default Namespaces gelten nicht für Attributnamen.)

Reservierte Namespaces und Präfixe

- Der Präfix `xml` ist immer an den Namespace `http://www.w3.org/XML/1998/namespace` gebunden.

  Beide dürfen nicht anderweitig gebunden werden.

  Der Präfix `xml` darf, muss aber nicht deklariert werden.

- Der Präfix `xmlns` ist immer an den Namespace `http://www.w3.org/2000/xmlns/` gebunden und darf nicht deklariert werden.

  Der Namespace `http://www.w3.org/2000/xmlns/` darf nicht an andere Präfixe gebunden werden.

- Alle anderen Präfixe, die mit `xml` beginnen (in beliebiger Groß- oder Kleinschreibung), sind reserviert.

Qualifizierte Namen und Namespaces in der DTD

Problem: In DTD-Deklarationen müssen Elementnamen und Attributnamen mit einem festen Präfix versehen werden.

Beispiel 2.58 (DTD).

```
<!ELEMENT h:haus (z:zimmer*) >
<!ELEMENT z:zimmer (#PCDATA)>

<!ATTLIST h:haus xmlns:h CDATA #IMPLIED>
<!ATTLIST z:zimmer xmlns:z CDATA #FIXED "http://n2.de">
```

DTDs sind für Sprachen mit Namespaces nicht gut geeignet.

Namespaces in XML 1.1

1. Erweiterung von URIs auf IRIs

   Beispiel 2.59 (IRI). (Internationalized Resource Identifier)

   ```
   http://www.example.org/rosé
   ```

2. Präfixe können „undeklariert“ werden.

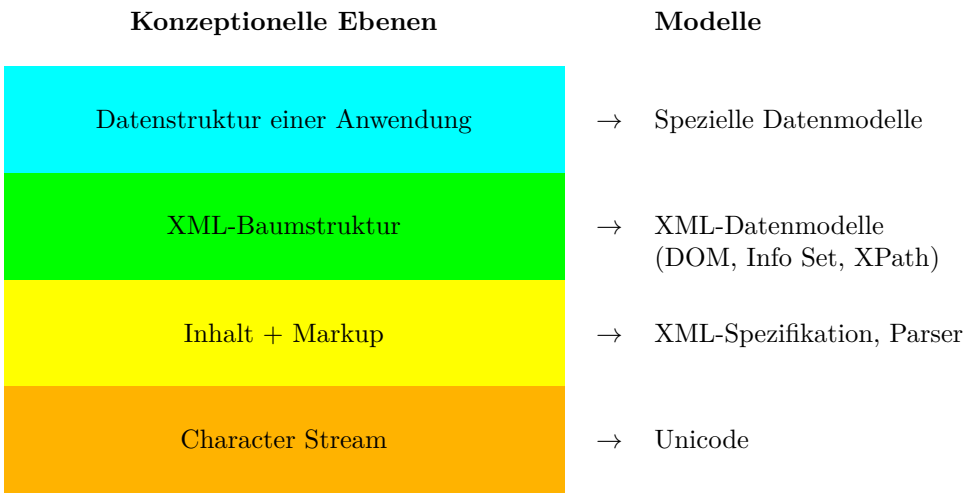
   Beispiel 2.60.

   ```
   <x xmlns:n1="http://www.w3.org">
     <n1:a/>
     <x xmlns:n1="">
       <n1:a/> <!-- illegal; the prefix n1 is not bound -->
       <x xmlns:n1="http://www.w3.org">
         <n1:a/>
       </x>
     </x>
   </x>
   ```

XML — Zusammenfassung

- *Bäume!*
- Gemeinsame *Syntax* für Anwendungssprachen
- *DTD* definiert Vokabular einer Anwendungssprache
- *Namespaces* modularisieren
- Semantik: *Infosets* (Bäume)

XML — Zusammenfassung



3 XML Schema

3.1 Einleitung

XML Schema

XML Schema 1.0 — W3C Recommendations

- Mai 2001, Second Editions Oktober 2004
- **XML Schema Part 0: Primer** Second Edition[12]
  „non-normative“, gut zu lesende Einführung mit vielen Beispielen
- **XML Schema Part 1: Structures** Second Edition[13]
- **XML Schema Part 2: Datatypes** Second Edition[14]

XML Schema 1.1 — W3C Recommendations

- April 2012
- **W3C XML Schema Definition Language (XSD) 1.1 Part 1: Structures**[15]
- **W3C XML Schema Definition Language (XSD) 1.1 Part 2: Datatypes**[16]
- weitgehend kompatibel, einige Erweiterungen

[12] http://www.w3.org/TR/xmlschema-0/
[13] http://www.w3.org/TR/xmlschema-1/
[14] http://www.w3.org/TR/xmlschema-2/
[15] http://www.w3.org/TR/xmlschema11-1/
[16] http://www.w3.org/TR/xmlschema11-2/

DTD: beschränkte Möglichkeiten

Beispiel 3.1 (DTD).

```
<!ELEMENT staff (employee)*>
<!ELEMENT employee (name, salary)>
<!ATTLIST employee
          id   ID    #REQUIRED
          boss IDREF #IMPLIED>
<!ELEMENT name   (#PCDATA)>    <!ELEMENT salary (#PCDATA)>
```

Beispiel 3.2 (Instanz).

```
<staff>
  <employee id="p12634-2" boss="p00008-5">
    <name>Joe Miller</name>
    <salary>3100</salary>
  </employee>
  <employee id="p00008-5">
    <name>Sam Smith</name>
    <salary>12000</salary>
  </employee>
</staff>
```

XML Schema versus DTD — Motivation

Eigenschaften DTD

- DTD ist *der* Schema-Mechanismus für SGML und der erste Schema-Mechanismus für XML.
- DTD definiert *Content Model* und in beschränktem Umfang die Datentypen von Attributen.

Nachteile DTD

- DTD benutzt eine spezielle Syntax (nicht XML).
- *Namespaces* werden nicht unterstützt.
- Nur sehr beschränkte *Datentypen*; nur für Attributwerte, nicht für den Elementinhalt.
- Asymmetrien zwischen Element- und Attributtypen
- Constraints nur mit den *Typen* `ID` und `IDREF`, eingeschränkter Typ (nur `Name`)
- Parameter Entities können nicht die vielfältigen Beziehungen zwischen Datentypen modellieren.
- ...

Eigenschaften von XML Schema

- XML-Syntax (mit Namespace)
- Vielfältige Datentypen: diverse Zahlen, Boolean, Datum, Zeit, URIs, ..., für Elemente *und* Attribute
- Benutzerdefinierbare Typen
- Namespace-Unterstützung
- Vererbung
- Verteilte Schemas
- Äquivalenz (Substituierbarkeit) von Typen
- Allgemeine Schlüssel- und Fremdschlüsselbedingungen
- XML Schema Dokumente sind selbst durch ein XML Schema beschrieben. (Es gibt auch eine DTD für XML Schema.)

3.2 Beispiel

Ein erstes Beispiel: Instanzdokument

Beispiel 3.3 („Purchase Order“ Instanz — `po.xml`).

```
<?xml version="1.0"?>
<purchaseOrder orderDate="1999-10-20">
    <shipTo country="US">
        <name>Alice Smith</name>
        <street>123 Maple Street</street>
        <city>Mill Valley</city>
        <state>CA</state>
        <zip>90952</zip>
    </shipTo>
    <billTo country="US">
        <name>Robert Smith</name>
        <street>8 Oak Avenue</street>
        <city>Old Town</city>
        <state>PA</state>
        <zip>95819</zip>
    </billTo>
    <comment>Hurry, my lawn is going wild!</comment>

    <items>
        <item partNum="872-AA">
            <productName>Lawnmower</productName>
            <quantity>1</quantity>
            <USPrice>148.95</USPrice>
            <comment>Confirm this is electric</comment>
        </item>
```

```
        <item partNum="926-AA">
            <productName>Baby Monitor</productName>
            <quantity>1</quantity>
            <USPrice>39.98</USPrice>
            <shipDate>1999-05-21</shipDate>
        </item>
    </items>
</purchaseOrder>
```

Ein erstes Beispiel: XML Schema

Beispiel 3.4 („Purchase Order“ Schema — `po.xsd`).

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

 <xsd:annotation>
  <xsd:documentation>
   Purchase order schema for Example.com. Copyright 2000 Example.com.
  </xsd:documentation>
 </xsd:annotation>

 <xsd:element name="purchaseOrder" type="PurchaseOrderType"/>

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

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

 <xsd:complexType name="USAddress">
  <xsd:sequence>
   <xsd:element name="name"   type="xsd:string"/>
   <xsd:element name="street" type="xsd:string"/>
   <xsd:element name="city"   type="xsd:string"/>
   <xsd:element name="state"  type="xsd:string"/>
   <xsd:element name="zip"    type="xsd:decimal"/>
  </xsd:sequence>
  <xsd:attribute name="country" type="xsd:NMTOKEN" fixed="US"/>
 </xsd:complexType>

 <xsd:complexType name="Items">
  <xsd:sequence>
   <xsd:element name="item" minOccurs="0" maxOccurs="unbounded">
    <xsd:complexType>
     <xsd:sequence>
      <xsd:element name="productName" type="xsd:string"/>
      <xsd:element name="quantity">
       <xsd:simpleType>
```

```
        <xsd:restriction base="xsd:positiveInteger">
         <xsd:maxExclusive value="100"/>
        </xsd:restriction>
       </xsd:simpleType>
      </xsd:element>
      <xsd:element name="USPrice"  type="xsd:decimal"/>
      <xsd:element ref="comment"   minOccurs="0"/>
      <xsd:element name="shipDate" type="xsd:date" minOccurs="0"/>
     </xsd:sequence>
     <xsd:attribute name="partNum" type="SKU" use="required"/>
    </xsd:complexType>
   </xsd:element>
  </xsd:sequence>
 </xsd:complexType>

 <!- Stock Keeping Unit, a code for identifying products ->
 <xsd:simpleType name="SKU">
  <xsd:restriction base="xsd:string">
   <xsd:pattern value="\d{3}-[A-Z]{2}"/>
  </xsd:restriction>
 </xsd:simpleType>

</xsd:schema>
```

3.3 Einfache Typen

Datentypen in XML Schema

Simple Types

- Viele „built-in“-Typen und
- daraus abgeleitete (durch Einschränkung: Teilmenge)
- können in Element- *und* Attributdeklarationen verwendet werden.

Complex Types

- definieren *Elementinhalt* (*Content Model*) und *Attribute*,
- können nur in Elementdeklarationen verwendet werden,
- können *Vererbungsmechanismen* verwenden.

Typdefinitionen

- Eigene Typen definierbar zur Verwendung in Element- und Attributdeklarationen.

Einfache „built-in“ Datentypen (Auswahl)

Simple Type	Example(s)	Notes
string	`Hello world!`	
normalizedString	`Hello world!`	
boolean	`true, false, 1, 0`	
decimal	`-1.23, 0, 123.4, 1000.00`	arbitrary precision decimal numbers; minimum of 18 decimal digits
integer	`-1, 0, 1, 126789`	minimum of 18 decimal digits
nonNegativeInteger	`0, 1, 126789`	
positiveInteger	`1, 126789`	
nonPositiveInteger	`-126789, -1, 0`	
negativeInteger	`-126789, -1`	
date	`1999-05-31`	ISO format
time	`13:20:00.000, 13:20:00.000-05:00`	
dateTime	`1999-05-31T↩ 13:20:00.000-05:00`	May 31st 1999 at 1.20pm Eastern Standard Time which is 5 hours behind Co-Ordinated Universal Time
gYear	`1999`	1999
gYearMonth	`1999-05`	May 1999
gMonthDay	`-05-31`	every May 31st
gDay	`--31`	every 31st day
duration	`P1Y2M3DT10H30M12.3S`	1 year, 2 months, 3 days, 10 hours, 30 minutes, 12.3 seconds

Einfache „built-in“ Datentypen (Auswahl, Forts.)

Simple Type	Example(s)	Notes
ID		XML 1.0 ID attribute type
IDREF		XML 1.0 IDREF attribute type
language	`en-GB, en-US, fr`	valid values for xml:lang as defined in XML 1.0
NMTOKEN	`US`	XML 1.0 NMTOKEN attribute type
NMTOKENS	`US UK`	XML 1.0 NMTOKENS attribute type
Name	`shipTo`	XML 1.0 Name type
QName	`po:Address`	XML Namespace QName
NCName	`USAddress`	XML Namespace NCName
anyUri	`http://www.example.com`	
float	`-INF, -1E4, -0, 0, 12.78E-2, 12, INF, NaN`	equivalent to single-precision 32-bit floating point, NaN is „not a number“
double	`-INF, -1E4, -0, 0, 12.78E-2, 12, INF, NaN`	equivalent to double-precision 64-bit floating point
long	`-1, 12678967543233`	64 bit signed
int	`-1, 126789675`	32 bit signed
short	`-1, 12678`	16 bit signed
byte	`-1, 126`	8 bit signed
unsignedLong	`0, 12678967543233`	
unsignedInt	`0, 1267896754`	
unsignedShort	`0, 12678`	
unsignedByte	`0, 126`	
binary	`100010`	

Ableitung durch Einschränkung (Restriction)

Beispiel 3.5 (Element with integer content, Range 1-99).

```
<xsd:element name="quantity">
   <xsd:simpleType>
      <xsd:restriction base="xsd:positiveInteger">
         <xsd:maxExclusive value="100"/>
      </xsd:restriction>
   </xsd:simpleType>
</xsd:element>
```

Beispiel 3.6 (Simple Type `myInteger`, Range 10000-99999).

```
<xsd:simpleType name="myInteger">
   <xsd:restriction base="xsd:integer">
      <xsd:minInclusive value="10000"/>
      <xsd:maxInclusive value="99999"/>
   </xsd:restriction>
```

```
</xsd:simpleType>
```

Beispiel 3.7 (Simple Type `SKU`, e. g. `123-XY`, `007-JB`).

```
<!-- Stock Keeping Unit (product code) -->
<xsd:simpleType name="SKU">
   <xsd:restriction base="xsd:string">
      <xsd:pattern value="\d{3}-[A-Z]{2}"/>
   </xsd:restriction>
</xsd:simpleType>
```

Einschränkung durch Facetten

Einfache Typen durch *Facetten* einschränken:

1. length
2. minLength
3. maxLength
4. pattern
5. enumeration
6. whiteSpace
7. maxInclusive
8. maxExclusive
9. minInclusive
10. minExclusive
11. totalDigits
12. fractionDigits

Einschränkung durch Facetten

Beispiel 3.8 (Enumeration Facet).

```
<xsd:simpleType name="USState">
   <xsd:restriction base="xsd:string">
      <xsd:enumeration value="AK"/>
      <xsd:enumeration value="AL"/>
      <xsd:enumeration value="AR"/>
      <!-- and so on ... -->
   </xsd:restriction>
</xsd:simpleType>
```

Die Facette ***`enumeration`*** kann auf alle einfachen Typen außer `boolean` angewendet werden.

Listentypen

- Listentypen sind einfache Typen.
- „built-in“ Listentypen: `NMTOKENS`, `IDREFS` und `ENTITIES`

Beispiel 3.9 (Liste von `myInteger`).

```
<xsd:simpleType name="listOfMyIntType">
   <xsd:list itemType="myInteger"/>
</xsd:simpleType>

<xsd:element name="mylist" type="listOfMyIntType"/>
```

Beispiel 3.10 (Instanz).

```
<mylist>20003 15037 95977 95945</mylist>
```

Listentypen

Beispiel 3.11 (List Type US States).

```
<xsd:simpleType name="USStateList">
   <xsd:list itemType="USState"/>
</xsd:simpleType>
```

Beispiel 3.12 (List Type for Six US States).

```
<xsd:simpleType name="SixUSStates">
   <xsd:restriction base="USStateList">
      <xsd:length value="6"/>
   </xsd:restriction>
</xsd:simpleType>
```

Beispiel 3.13 (Instanz von `SixUSStates`).

```
PA NY CA NY LA AK
```

Assertions (XML Schema 1.1)

Weitere Einschränkung des Wertebereichs auf Basis eines booleschen Ausdrucks (XPath 2.0)

Beispiel 3.14.

```
<xs:simpleType name="evenNumber">
  <xs:restriction base="xs:integer">
    <xs:minInclusive value="0" />
    <xs:maxInclusive value="100" />
    <xs:assertion test="$value mod 2 = 0" />
  </xs:restriction>
</xs:simpleType>
```

Vereinigungstypen

- Vereinigungstypen sind einfache Typen,
- basierend auf anderen einfachen Typen.

Beispiel 3.15 (Union Type for Zipcodes).

```
<xsd:simpleType name="zipUnion">
  <xsd:union memberTypes="USState listOfMyIntType"/>
</xsd:simpleType>

<xsd:element name="zips" type="zipUnion"/>
```

Beispiel 3.16 (Instanzen).

```
<zips>CA</zips>
<zips>95630 95977 95945</zips>
<zips>AK</zips>
```

3.4 Komplexe Typen

Komplexe Typen

- Komplexe Typen haben Unterelemente oder Attribute.
- Einfache Typen zu komplexen Typen erweitern:

Beispiel 3.17 (Attribut deklarieren).

```
<xsd:element name="intlPrice">
  <xsd:complexType>
    <xsd:simpleContent>
      <xsd:extension base="xsd:decimal">
        <xsd:attribute name="curr" type="xsd:string"/>
      </xsd:extension>
    </xsd:simpleContent>
  </xsd:complexType>
</xsd:element>
```

Beispiel 3.18 (Instanz).

```
<intlPrice curr='EUR'>423.46</intlPrice>
```

Komplexe Typen: Leerer Inhalt

Beispiel 3.19 (Komplexer Typ ohne Inhalt).

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

Beispiel 3.20 (Instanz).

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

Content Models: `sequence`, `choice` und `group`

- `sequence` und `choice` entsprechen den Operatoren `,` bzw. `|` in einer DTD.
- Wiederholungsanzahl durch die Attribute *`minOccurs`* und *`maxOccurs`* — diese entsprechen den Operatoren *`* + ?`* in einer DTD.
- Default-Wert von `minOccurs` `maxOccurs` ist jeweils 1.
- `group`-Elemente können Namen erhalten und damit wiederverwendet werden.

Content Models: `sequence`, `choice` und `group`

Beispiel 3.21 (Verschachtelte `choice`- und `sequence`-Gruppen).

```
<xsd:complexType name="PurchaseOrderType">
  <xsd:sequence>
    <xsd:choice>
      <xsd:group   ref="shipAndBill" />
      <xsd:element name="singleUSAddress" type="USAddress" />
    </xsd:choice>
    <xsd:element ref="comment" minOccurs="0"/>
    <xsd:element name="items"  type="Items" />
  </xsd:sequence>
  <xsd:attribute name="orderDate" type="xsd:date" />
</xsd:complexType>

<xsd:group name="shipAndBill">
  <xsd:sequence>
    <xsd:element name="shipTo" type="USAddress" />
    <xsd:element name="billTo" type="USAddress" />
  </xsd:sequence>
</xsd:group>
```

Content Models: `all`

- `all` nur als einziges Kind eines Content Models
- In `all` sind nur Elemente zugelassen (keine Gruppen).

- zulässige Werte der `minOccurs`- und `maxOccurs`-Attribute: 0 od. 1 (beliebig in Schema 1.1)

Beispiel 3.22.

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

Komplexe Typen: Mixed Content

Beispiel 3.23 (Ausschnitt aus einem Brief).

```
<letterBody>
<salutation>Dear Mr.<name>Robert Smith</name>.</salutation>
Your order of <quantity>1</quantity> <productName>Baby
Monitor</productName> shipped from our warehouse on
<shipDate>1999-05-21</shipDate>. ....
</letterBody>
```

Beispiel 3.24 (Ausschnitt aus dem zugehörigen Schema).

```
<xsd:element name="letterBody">
  <xsd:complexType mixed="true">
    <xsd:sequence>
      <xsd:element name="salutation">
        <xsd:complexType mixed="true">
          <xsd:sequence>
            <xsd:element name="name" type="xsd:string"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="quantity"
                   type="xsd:positiveInteger"/>
      <xsd:element name="productName"
                   type="xsd:string"/>
      <xsd:element name="shipDate"
                   type="xsd:date" minOccurs="0"/>
      <!-- etc -->
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Attributgruppen

Beispiel 3.25 (Attributgruppe).

```
<xsd:element name="item" minOccurs="0" maxOccurs="unbounded">
 <xsd:complexType>
  <xsd:sequence>...</xsd:sequence>
  <!-- attributeGroup replaces individual declarations -->
  <xsd:attributeGroup ref="ItemDelivery"/>
 </xsd:complexType>
</xsd:element>

<xsd:attributeGroup name="ItemDelivery">
  <xsd:attribute name="partNum"  type="SKU"/>
  <xsd:attribute name="weightKg" type="xsd:decimal"/>
  <xsd:attribute name="shipBy">
    <xsd:simpleType>
     <xsd:restriction base="xsd:string">
      <xsd:enumeration value="air"/> ...
     </xsd:restriction>
    </xsd:simpleType>
  </xsd:attribute>
</xsd:attributeGroup>
```

3.5 Namespaces

Namespaces, Schemas & Qualification

Vokabulare

- Vokabular eines Schemas: Element- und Attributdeklarationen, Typdefinitionen
- *Target Namespaces*: Unterscheidung und Modularisierung der Vokabulare

Namespaces in XML Schema

- Ein Schema kann *einen Target Namespace* haben.
- Dieser gilt zunächst für die globalen Elemente, Attribute und Typen.
- Die Qualifizierung von lokalen Elementen und/oder und Attributen kann erzwungen werden.

Namespaces, Schemas & Qualification

Beispiel 3.26 (Purchase Order Schema with Target Namespace).

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:po="http://www.example.com/PO1"
        targetNamespace="http://www.example.com/PO1"
        elementFormDefault="unqualified"
        attributeFormDefault="unqualified">
```

```
  <element name="purchaseOrder" type="po:PurchaseOrderType"/>
  <element name="comment"       type="string"/>

  <complexType name="PurchaseOrderType">
    <sequence>
      <element name="shipTo"    type="po:USAddress"/>
      <element name="billTo"    type="po:USAddress"/>
      <element ref="po:comment" minOccurs="0"/> ...
    </sequence> ...
  </complexType>

  ...
</schema>
```

Namespaces, Schemas & Qualification

Beispiele 3.27 (A Purchase Order with Unqualified Locals).

```
<?xml version="1.0"?>
<apo:purchaseOrder xmlns:apo="http://www.example.com/PO1"
                   orderDate="1999-10-20">
  <shipTo country="US">
    <name>Alice Smith</name>
    <street>123 Maple Street</street>
    ...
  </shipTo>
  <billTo country="US">
    <name>Robert Smith</name>
    <street>8 Oak Avenue</street>
    ...
  </billTo>
  <apo:comment>Hurry, my lawn is going wild!</apo:comment>
  ...
</apo:purchaseOrder>
```

Namespace nur bei den *globalen Elementen* `purchaseOrder` und `comment`.

Namespaces, Schemas & Qualification

Beispiel 3.28 (Modifications to `po1.xsd` for Qualified Locals).

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:po="http://www.example.com/PO1"
        targetNamespace="http://www.example.com/PO1"
        elementFormDefault="qualified"
        attributeFormDefault="unqualified">

  <element name="purchaseOrder" type="po:PurchaseOrderType"/>
  <element name="comment"       type="string"/>

  <complexType name="PurchaseOrderType">
    ...
```

```
  </complexType>
  ...
</schema>
```

Übliche Variante in XML-Sprachen, z. B. in XHTML, XSLT, XSL-FO, XML Schema, SVG, SMIL, ...

Namespaces, Schemas & Qualification

Beispiel 3.29 (A Purchase Order with Explicitly Qualified Locals).

```
<?xml version="1.0"?>
<apo:purchaseOrder xmlns:apo="http://www.example.com/PO1"
                   orderDate="1999-10-20">
  <apo:shipTo country="US">
    <apo:name>Alice Smith</apo:name>
    <apo:street>123 Maple Street</apo:street>
    ...
  </apo:shipTo>
  <apo:billTo country="US">
    <apo:name>Robert Smith</apo:name>
    <apo:street>8 Oak Avenue</apo:street>
    ...
  </apo:billTo>
  <apo:comment>Hurry, my lawn is going wild!</apo:comment>
  ...
</apo:purchaseOrder>
```

Namespaces, Schemas & Qualification

Beispiel 3.30 (A Purchase Order with Default Qualified Locals).

```
<?xml version="1.0"?>
<purchaseOrder xmlns="http://www.example.com/PO1"
               orderDate="1999-10-20">
  <shipTo country="US">
    <name>Alice Smith</name>
    <street>123 Maple Street</street>
    ...
  </shipTo>
  <billTo country="US">
    <name>Robert Smith</name>
    <street>8 Oak Avenue</street>
    ...
  </billTo>
  <comment>Hurry, my lawn is going wild!</comment>
  ...
</purchaseOrder>
```

Namespaces, Schemas & Qualification

Beispiel 3.31 (Requiring Qualification of Single Attribute).

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:po="http://www.example.com/PO1"
        targetNamespace="http://www.example.com/PO1"
        elementFormDefault="qualified"
        attributeFormDefault="unqualified">
   ...
  <element name="secure">
    <complexType>
      <sequence>
        <!-- element declarations -->
      </sequence>
      <attribute name="publicKey" type="base64Binary"
                 form="qualified"/>
    </complexType>
  </element>
</schema>
```

Namespaces, Schemas & Qualification

Beispiel 3.32 (Instance with a Qualified Attribute).

```
<?xml version="1.0"?>
<purchaseOrder xmlns="http://www.example.com/PO1"
               xmlns:po="http://www.example.com/PO1"
               orderDate="1999-10-20">
  ...
  <secure po:publicKey="GpM7">
    ...
  </secure>
</purchaseOrder>
```

3.6 Globale und lokale Elementdeklarationen

Globale vs. lokale Elementdeklarationen

Positionsabhängige Typen?

- ein Elementname, verschiedene Typen an verschiedenen Positionen im Dokument?
- DTD: nein (nur globale Deklaration)
- XML Schema: ja (lokale oder globale Deklaration)

Beispiel 3.33 (Schema mit globalen Elementen, wie bei DTDs).

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:po="http://www.example.com/PO1"
        targetNamespace="http://www.example.com/PO1">
  <element name="purchaseOrder" type="po:PurchaseOrderType"/>
  <element name="shipTo" type="po:USAddress"/>
  <element name="billTo" type="po:USAddress"/>
```

```
  <element name="comment" type="string"/>
  <element name="name" type="string"/>
  <element name="street" type="string"/>
  <complexType name="PurchaseOrderType">...</complexType>
... </schema>
```

Globale vs. lokale Elementdeklarationen

- Lokale Elemente gleichen Namens können verschiedene Typen haben.

Beispiel 3.34 (mit lokalen Elementen).

```
<buch>
  <titel>Ein schönes Buch</titel>     <!-- string -->
  <autor>
    <titel>Dr.</titel>           <!-- enumeration -->
    <vorname>Albert</vorname>
    <nachname>Autor</nachname>
  </autor>
  ...
</buch>
```

3.7 Verteilte Schemas

Verteilte Schemas

Verteilte Schemas

- Schema aus mehreren Dokumenten zusammensetzen ⇒ Wiederverwendbarkeit
- Bei `include` kann nur ein Target Namespace verwendet werden.
- `import` von Schemas ermöglicht die Verwendung mehrerer Target Namespaces.
- Die eingefügten Typen können auch *redefiniert* (`xs:redefine`) oder *überschrieben* (`xs:override`, ab Schema 1.1) werden.

Verteilte Schemas

Beispiel 3.35 (The International Purchase Order Schema).

```
<schema targetNamespace="http://www.example.com/IPO"
         xmlns="http://www.w3.org/2001/XMLSchema"
         xmlns:ipo="http://www.example.com/IPO">

  <include schemaLocation = "http://www.example.com/schemas/address.xsd"/>

  <element name="purchaseOrder" type="ipo:PurchaseOrderType"/>
  <element name="comment" type="string"/>
```

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

3.8 Ableiten von Typen durch Erweiterung

Ableiten von Typen durch Erweiterung

- Content Model eines komplexen Typs erweitern
- Die *Content Models* des Ausgangstyps und der Erweiterung werden hintereinander gesetzt.

Beispiel 3.36 (Addresses for IPO schema — `address.xsd`).

```
<schema targetNamespace="http://www.example.com/IPO"
        xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:ipo="http://www.example.com/IPO">

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

 <complexType name="USAddress">
  <complexContent>
   <extension base="ipo:Address">
    <sequence>
     <element name="state" type="ipo:USState"/>
     <element name="zip"   type="positiveInteger"/>
    </sequence>
   </extension>
  </complexContent>
 </complexType>

 <complexType name="UKAddress">
  <complexContent>
   <extension base="ipo:Address">
    <sequence>
     <element name="postcode" type="ipo:UKPostcode"/>
    </sequence>
```

```
    <attribute name="exportCode" type="positiveInteger"
                                 fixed="1"/>
   </extension>
  </complexContent>
 </complexType>
 ...
</schema>
```

Verwendung abgeleiteter Typen

Beispiel 3.37 (International Purchase order, `ipo.xml`).

```
<ipo:purchaseOrder
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:ipo="http://www.example.com/IPO"
      orderDate="1999-12-01">
  <shipTo exportCode="1" xsi:type="ipo:UKAddress">
    <name>Helen Zoe</name>
    <street>47 Eden Street</street>
    <city>Cambridge</city>
    <postcode>CB1 1JR</postcode>
  </shipTo>
  <billTo xsi:type="ipo:USAddress">
    <name>Robert Smith</name>
    <street>8 Oak Avenue</street>
    <city>Old Town</city>
    <state>PA</state>
    <zip>95819</zip>
  </billTo>
  <items>...</items>
</ipo:purchaseOrder>
```

Schema-Informationen in Instanzen

XML Schema Namespace for Instances

- (Typ)-Informationen im Dokument
- *http://www.w3.org/2001/XMLSchema-instance*
- Attribute `xsi:type`, `xsi:nil`, `xsi:schemaLocation`, ...

3.9 Ableiten von Typen durch Restriktion

Ableiten komplexer Typen durch Restriktion

- Content Model eines komplexen Typs einschränken
- Modifizierte Deklaration komplett angeben

Beispiel 3.38 (Typ `RestrictedPurchaseOrderType` durch Restriktion aus `PurchaseOrderType`).

```
<complexType name="RestrictedPurchaseOrderType">
  <complexContent>
    <restriction base="ipo:PurchaseOrderType">
      <sequence>
        <element name="shipTo" type="ipo:Address"/>
        <element name="billTo" type="ipo:Address"/>
        <element ref="ipo:comment" minOccurs="1"/>
        <element name="items"  type="ipo:Items"/>
      </sequence>
    </restriction>
  </complexContent>
</complexType>
```

3.10 Typäquivalenz

Äquivalenzklassen

Substituierbarkeit

- Entsprechend deklarierte Element können *substituiert* werden.
- Der Elementtyp muß identisch mit oder abgeleitet aus dem Typ des „Head“- Elements sein.

Beispiel 3.39 (Declaring Elements Substitutable for comment).

```
<element name="shipComment" type="string"
    substitutionGroup="ipo:comment" />
<element name="customerComment" type="string"
    substitutionGroup="ipo:comment" />
```

Beispiel 3.40 (Snippet of `ipo.xml` with Substituted Elements).

```
<item partNum="833-AA">
  <productName>Lapis necklace</productName>
  <quantity>1</quantity>
  <USPrice>99.95</USPrice>
  <ipo:shipComment>Use gold wrap if possible</ipo:shipComment>
  <ipo:customerComment>
    Want this for the holidays!
  </ipo:customerComment>
  <shipDate>1999-12-05</shipDate>
</item>
```

3.11 Abstrakte Elemente und Typen

Abstrakte Elemente und Typen

Beispiel 3.41 (Abstrakter Typ `Vehicle`).

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        targetNamespace="http://cars.example.com/schema"
        xmlns:target="http://cars.example.com/schema">
  <complexType name="Vehicle" abstract="true"/>

  <complexType name="Car">
    <complexContent>
      <extension base="target:Vehicle"/>
    </complexContent>
  </complexType>

  <complexType name="Plane">
    <complexContent>
      <extension base="target:Vehicle"/>
    </complexContent>
  </complexType>

 <element name="transport" type="target:Vehicle"/>
</schema>
```

Abstrakte Elemente und Typen

Beispiel (falsche Anwendung)

```
<transport xmlns="http://cars.example.com/schema" />
```

Beispiel 3.42 (richtige Anwendung).

```
<transport xmlns="http://cars.example.com/schema"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:type="Car"/>
```

Auch möglich: Ableitungsmöglichkeiten einschränken (Attribut `final`)

3.12 Constraints

Constraints in XML Schema

- (In DTD: nur `ID`-/`IDREF`-Attributtypen)
- In XML Schema: Constraints über beliebige, auch mehrere Felder
- Element- und Attributinhalte über (eingeschränkte) XPath-Ausdrücke spezifizierbar
- Eindeutigkeit: *xsd:unique*
- Schlüssel: *xsd:key*
- Fremdschlüssel: *xsd:keyref*

Constraints

Beispiel 3.43 (Quarterly Report, Daten).

```
<purchaseReport xmlns="http://www.example.com/Report"
                period="P3M" periodEnding="1999-12-31">
 <regions>
  <zip code="95819">
   <part number="872-AA" quantity="1"/>
   <part number="926-AA" quantity="1"/>
   <part number="833-AA" quantity="1"/>
   <part number="455-BX" quantity="1"/>
  </zip>
  <zip code="63143">
   <part number="455-BX" quantity="4"/>
  </zip>
 </regions>

 <parts>
  <part number="872-AA">Lawnmower</part>
  <part number="926-AA">Baby Monitor</part>
  <part number="833-AA">Lapis Necklace</part>
  <part number="455-BX">Sturdy Shelves</part>
 </parts>
</purchaseReport>
```

Constraints: Schlüssel und Fremdschlüssel

Beispiel 3.44 (Schlüssel und Fremdschlüssel).

```
<element name="purchaseReport">
  <complexType>
    <sequence>
      <element name="regions" type="r:RegionsType"/>
      <element name="parts"   type="r:PartsType"/>
    </sequence>
    <attribute name="period"       type="duration"/>
    <attribute name="periodEnding" type="date"/>
  </complexType>
  ...
  <key name="pNumKey">
    <selector xpath="r:parts/r:part"/>
    <field xpath="@number"/>
  </key>
  <keyref name="dummy2" refer="r:pNumKey">
    <selector xpath="r:regions/r:zip/r:part"/>
    <field xpath="@number"/>
  </keyref>
</element>
```

Constraints: Eindeutige Werte

unique

Innerhalb einer durch einen `XPath`-Ausdruck gewählten Knotenmenge darf jeder Wert (bzw. jedes Wertetupel) nur einmal vorkommen.

Beispiel 3.45 (`unique`). „Jeder ZIP-Code soll nur einmal gelistet werden“:

```
<unique name="dummy1">
  <selector xpath="r:regions/r:zip"/>
  <field xpath="@code"/>
</unique>
```

Constraints: Wertetupel

Beispiel 3.46 (Kombinierter Schlüssel mit zwei Komponenten).

```
  <xs:key name="regKey">
    <xs:selector xpath=".//vehicle"/>
    <xs:field xpath="@state"/>
    <xs:field xpath="@plateNumber"/>
  </xs:key>
```

Assertions (XML Schema 1.1)

Validierung auf Basis von booleschen XPath-Ausdrücken

`xs:assert` bei Complex Types

Beispiel 3.47.

```
<xs:complexType name="intRange">
  <xs:attribute name="min" type="xs:int"/>
  <xs:attribute name="max" type="xs:int"/>
  <xs:assert test="@min le @max"/>
</xs:complexType>
```

Beispiel 3.48 (Instanz).

```
<family>
  <name>Müller</name>
  <ages min="3" max="85" />
</family>
```

Beispiel 3.49 (Schema).

```
<xs:complexType name="arrayType">
  <xs:sequence>
    <xs:element name="entry" minOccurs="0"
                             maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:attribute name="length" type="xs:int"/>
  <xs:assert test="@length eq fn:count(./entry)"/>
</xs:complexType>
```

Beispiel 3.50 (Instanz).

```
<array length="3">
  <entry>...</entry>
  <entry>...</entry>
  <entry>...</entry>
<array>
```

Beispiel 3.51.

```
<xs:element name="Transportation">
  <xs:complexType>
    <xs:choice>
      <xs:element name="airplane" type="xs:string" />
      <xs:element name="boat" type="xs:string" />
      <xs:element name="car" type="xs:string" />
    </xs:choice>
    <xs:attribute name="mode" type="modeType"
                              use="required"/>
    <xs:assert test="
        if (@mode eq 'air') then airplane
        else if (@mode eq 'water') then boat
        else if (@mode eq 'ground') then car
        else false()"/>
  </xs:complexType>
</xs:element>
```

`xs:assertion` bei Simple Types

Beispiel 3.52.

```
<xs:simpleType name="evenNumber">
  <xs:restriction base="xs:integer">
    <xs:minInclusive value="0" />
    <xs:maxInclusive value="100" />
    <xs:assertion test="$value mod 2 = 0" />
  </xs:restriction>
</xs:simpleType>
```

3.13 Verschiedenes

any-Element, any-Attribut

Beispiel 3.53 (`any`-Element).

```
<element name="purchaseReport">
 <complexType>
  <sequence>
   <element name="regions" type="r:RegionsType"/>
   <element name="parts" type="r:PartsType"/>
   <element name="htmlExample">
    <complexType>
     <sequence>
      <any namespace="http://www.w3.org/1999/xhtml"
           minOccurs="1" maxOccurs="unbounded"
```

```
            processContents="skip"/>
      </sequence>
     </complexType>
    </element>
  </sequence>
  <attribute name="period"       type="timeDuration"/>
  <attribute name="periodEnding" type="date"/>
 </complexType>
</element>
```

any-Element, any-Attribut

Beispiel 3.54 (Quarterly Report with HTML, `4Q99html.xml`).

```
<purchaseReport
   xmlns="http://www.example.com/Report"
   period="P3M" periodEnding="1999-12-31">
  <regions>...</regions>
  <parts>...</parts>
  <htmlExample>
    <table xmlns="http://www.w3.org/1999/xhtml"
           border="0" width="100%">
      <tr>
        <th align="left">Zip Code</th>
        <th align="left">Part Number</th>
        <th align="left">Quantity</th>
      </tr>
      <tr><td>95819</td><td> </td><td> </td></tr>
      <tr><td> </td><td>872-AA</td><td>1</td></tr>
      ...
    </table>
  </htmlExample>
</purchaseReport>
```

any-Element, any-Attribut

Beispiel 3.55 (`anyAttribute`).

```
<element name="htmlExample">
  <complexType>
    <sequence>
      <any namespace="http://www.w3.org/1999/xhtml"
           minOccurs="1" maxOccurs="unbounded"
           processContents="skip"/>
    </sequence>
    <anyAttribute namespace="http://www.w3.org/1999/xhtml"/>
  </complexType>
</element>
```

Beispiel 3.56 (An XHTML attribute in the htmlExample Element).

```
<htmlExample xmlns:h="http://www.w3.org/1999/xhtml"
          h:href="http://www.example.com/reports/4Q99.html">
```

```
  <h:p>Hello World.</h:p>
</htmlExample>
```

Open Content Models

(Ab Schema 1.1)
Beispiel 3.57.

```
<xs:complexType name="name">
  <xs:openContent>
    <xs:any namespace="##other" processContents="skip"/>
  </xs:openContent>
  <xs:sequence>
    <xs:element name="given" type="xs:string"/>
    <xs:element name="middle" type="xs:string"
                              minOccurs="0"/>
    <xs:element name="family" type="xs:string"/>
  </xs:sequence>
</xs:complexType>
```

A complex type definition that allows three explicitly declared child elements, in the specified order, and furthermore allows additional elements of any name from any namespace other than the target namespace to appear anywhere in the children.

Explizite Zuweisung eines Schema-Dokuments

Beispiel 3.58 (Attribut `schemaLocation`).

```
<purchaseReport
  xmlns="http://www.example.com/Report"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.example.com/Report
                      http://www.example.com/Report.xsd"
  period="P3M" periodEnding="1999-12-31"> ...
</purchaseReport>
```

Beispiel 3.59 (`schemaLocation` ohne Namespaces).

```
<p xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
      xsi:noNamespaceSchemaLocation='myschema.xsd'>
  ...
</p>
```

Beispiel 3.60 (Schema-validierend parsen mit Apache Xerces).

```
java -cp xercesImpl.jar:xercesSamples.jar sax.Counter -v -s -f test.xml
```

Parsen mit Apache Xerces

Download: Xerces2 Java Parser[17], Version 2.9 oder höher. Benötigt werden z. B. `Xerces-J-bin.2.9.1.tar.gz` oder `Xerces-J-bin.2.9.1.zip`

[17] http://xerces.apache.org/xerces2-j/index.html

Die darin enthaltenen Archive `xercesImpl.jar` und `xercesSamples.jar` müssen auf dem `CLASSPATH` liegen oder mit `-cp` beim Aufruf angegeben werden.

Ein Aufruf `java -cp .../xercesImpl.jar:.../xercesSamples.jar sax.Counter` (ohne weitere Argumente) gibt eine Hilfeseite aus, auf der man u. a. erfährt, wie man validierend parsen kann. (Pfade bitte anpassen. Unter Windows sind die Teilpfade mit `;` anstelle des `:` zu trennen.)

Validierung

Ein wohlgeformtes Instanzendokument wird gegen ein Schema geprüft:

Schema-Validierung Ist das Instanzendokument gültig bzgl. des Schemas?

Infoset Contributions Hinzufügen von Information, die im Instanzendokument nicht direkt enthalten ist, sondern im Schema (Defaultwerte und Typen).

XML Schema — Zusammenfassung

- vielfältige Datentypen
- benutzerdefinierbare Typen
- interessante Anwendungsmöglichkeiten
- umfangreich (aufwändig zu implementieren)
- Verwendung u. a. in XPath 2.0 und XQuery
- Verfügbare Anwendungen: Parser, XML-Editoren, Schema-Editoren, Maskengeneratoren, ...

4 Web-Technologien

Literatur

Jeffrey C. Jackson; Web Technologies: A Computer Science Perspective; Prentice Hall 2006. Leider schon teilweise veraltet, zurzeit völlig überteuert, aber gut fürs Verständnis.

Materialien

Originalquellen

- World Wide Web Consortium: http://www.w3.org/ Gegründet im Okt. 1994 durch Tim Berners-Lee, den „Erfinder des WWW“

- Web Hypertext Application Technology Working Group (WHATWG)[18] „HTML Living Standard“, etc.
- Internet Engineering Task Force: `http://www.ietf.org/` HTTP/2, etc.

Weitere Quellen

- Mozilla Developer Network — Web technology for developers[19]
- The Web platform: Browser technologies[20]
- Wikipedia (Übersicht, Links)
- Bücher...

Schlechte Quelle

- Oft bei den ersten Google-Treffern, aber nicht zu empfehlen: `www.w3schools.com` hat mit dem W3C nichts zu tun, ist nicht immer korrekt und oft unvollständig!

4.1 Internet

Internet seit Mitte der 70er-Jahre

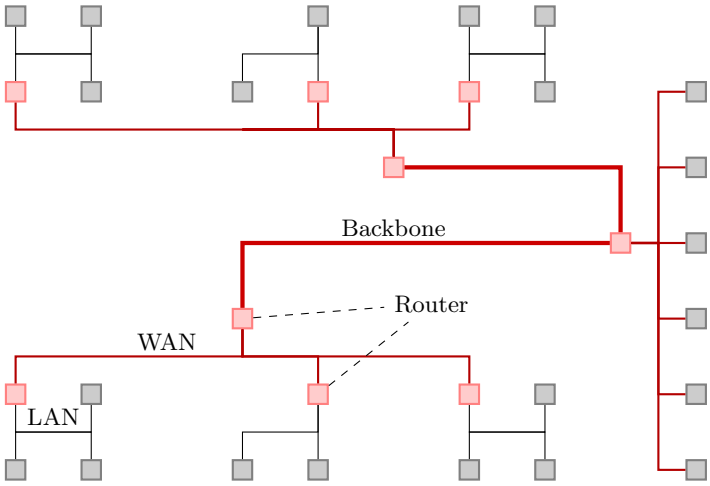
Schichtenmodell (TCP/IP)

- Anwendungsschicht
  - HTTP, FTP, SMTP, USENET, ...
- Transportschicht
  - zuverlässige Duplexverbindung: TCP
  - für andere Aufgaben z. B. UDP
- Vermittlungsschicht (Paketvermittlung: IP)
  - *IP-Nummern* als Adressen, z. B. `131.220.8.1`
- Netzzugangsschicht (z. B. Ethernet)

- IP-Paketvermittlung (unzuverlässig)
- Höhere Schichten bauen auf den unterliegenden auf
- Client/Server-Architektur
- Domain Name System (DNS) für lesbare Namen, z. B. `www.informatik.uni-bonn.de`

[18] `http://www.whatwg.org/`
[19] `https://developer.mozilla.org/en-US/docs/Web`
[20] `https://platform.html5.org/`

Netzzugangs- und Vermittlungsschicht



4.2 World Wide Web

Anfänge der Hypertext-Systeme

Vannevar Bush: As We May Think (1945)

- ein visionärer Artikel!
- analoge Speichermaschine „Memex“ (Mikrofilme, Kamera, Elektromechanik)
- noch keine Vernetzung mit weiteren Geräten
- lokale Links zwischen Dokumenten, Browser
- Informationen und Links jederzeit einfügbar

> *Wholly new forms of encyclopedias will appear, ready made with a mesh of associative trails running through them, ...*

Theodor H. Nelson

- *Project Xanadu*, das erste *Hypertext*-Projekt

  > *By hypertext I mean non-sequential writing – text that branches and allows choices to the reader, best read at an interactive screen. (1965)*

- Nichtlineare Texte und Grafiken
- Verbunden durch *Links*
- Der Leser kann von Dokument zu Dokument springen.

Historische Quellen

- Vannevar Bush
  As We May Think[21]
  The Atlantic Monthly, July 1945
  — ein visionärer Artikel

- Theodor H. Nelson
  Getting it out of our system
  In *Information Retrieval: A Critical Review*, G. Schechter, ed. Thomson Books, Washington D.C., 1967, 191-210
  — „By *hypertext* I mean non-sequential writing – text that branches and allows choices to the reader, best read at an interactive screen." (1965)

  http://en.wikipedia.org/wiki/Ted_Nelson
  http://en.wikipedia.org/wiki/Project_Xanadu

- Tim Berners-Lee
  Information Management: A Proposal[22]
  CERN March 1989, May 1990

  — Der Anfang des *World Wide Web*

Siehe auch: Memex: Das "'dritte Auge"' hat Geburtstag[23]
— enthält eine kurze historische Zusammenfassung und weitere Links

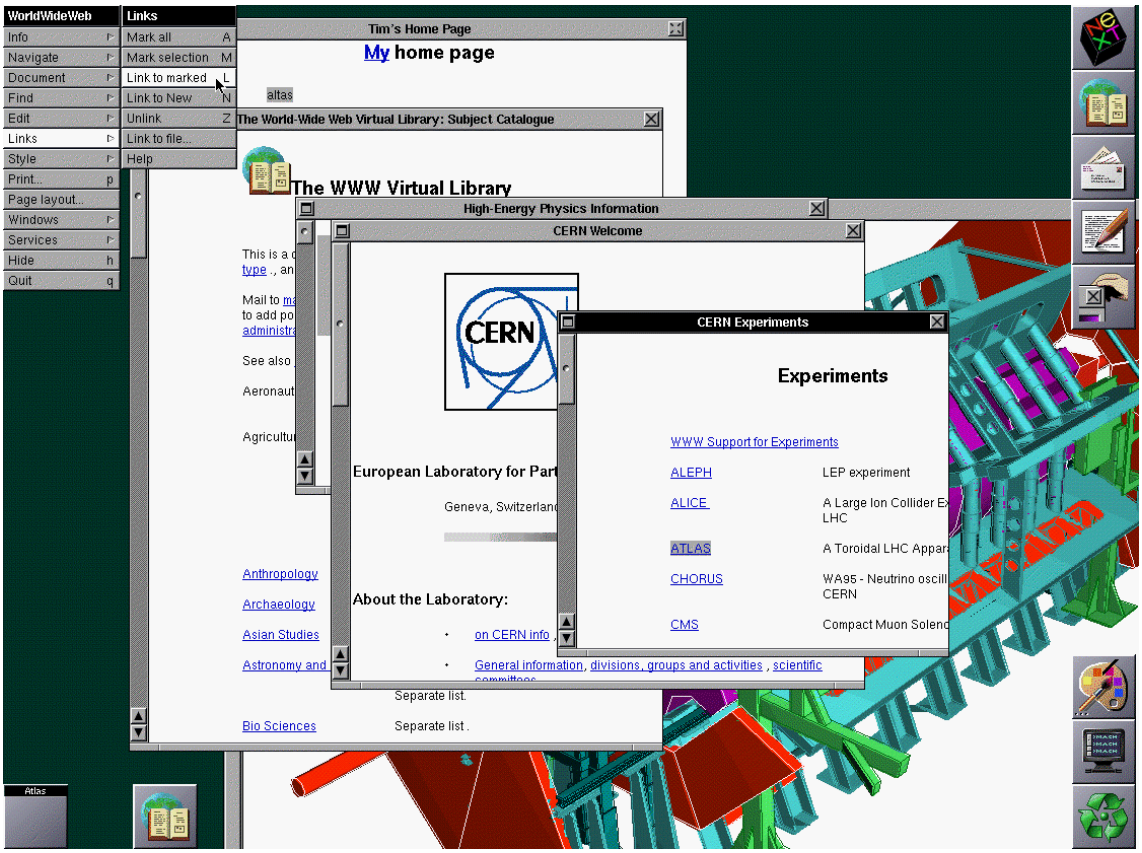
Entstehung des WWW

Tim Berners-Lee (ab 1989, CERN): Anforderungen

- WWW = Verknüpfung aus Hypertext und Internet
- Dezentrale Server
- Plattformunabhängig
- Einbindung vorhandener Daten (Files, Datenbanken, Manual-Pages, ...)
- *Private Links* (Bookmarks, aber auch Annotationen, Links *aus* Seiten, ...)
- ...

[21] http://www.theatlantic.com/doc/194507/bush
[22] http://www.w3.org/History/1989/proposal.html
[23] http://www.heise.de/newsticker/meldung/Memex-Das-dritte-Auge-hat-Geburtstag-118054.html

„Tim's Editor“



Quelle zum Screenshot (1993)

Tim Berners-Lee
The WorldWideWeb browser
http://www.w3.org/People/Berners-Lee/WorldWideWeb.html

WWW: Ein Netz über dem Netz

Zwei Netze

- Das Internet verbindet Rechner (Clients und Server)
- Das WWW verbindet Dokumente auf den Servern

WWW: ein großer weltweiter Hypertext

Hypertext als Graph

- Knoten: Dokumente, Teile von Dokumenten, beliebige Daten

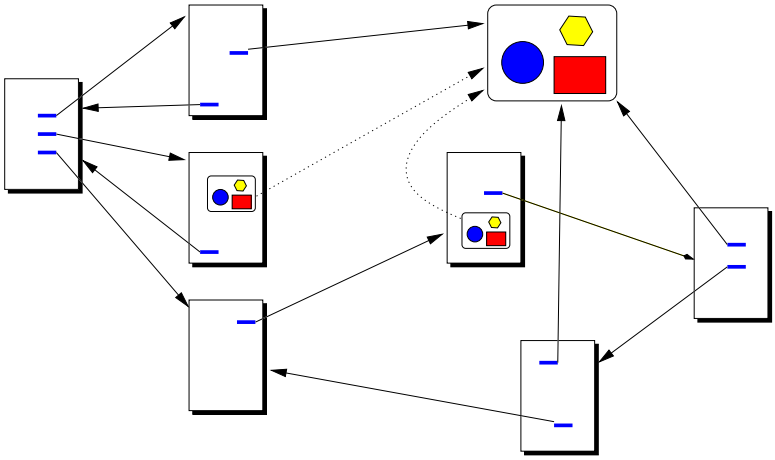
- Kanten: Hyperlinks, zur Navigation oder zur Einbettung

HTML-Hyperlinks haben nur eingeschränkte Funktionalität (vgl. Ansätze von V. Bush, T. Nelson und T. Berners-Lee).

Hyperlinks mit erweiterter Funktionalität

Z. B. XLink[24]

Struktur des WWW



Die drei wesentlichen Komponenten des WWW

1. Einheitliche *Adressen* für *Ressourcen* (Dokumente) im Internet: URI, URL, URN
2. Ein *Zugriffsprotokoll* für diese Ressourcen: HTTP
3. Eine *Markup*-Sprache für *Hypertext-Dokumente*: HTML

4.3 Adressen (URI)

WWW-Adressen

Uniform Resource Locator (URL)

Bestandteile eines URL

- Protokoll für den Zugriff

[24] http://www.w3.org/XML/Linking

- Name oder IP-Nummer des Servers
- Optional: eine Portnummer, Default für HTTP ist 80
- Pfad der Ressource auf dem Server
- Optional: ein Fragmentname innerhalb der Ressource

Beispiele 4.1 (URL). • `http://www.w3.org:80/Protocols/#Specs`

- `http://www.uni-bonn.de/Aktuelles.html`

Absolute und relative URLs

Beispiel 4.2 (absolute URL). • `http://www.acme.com/support/intro.html`

Beispiele 4.3 (relative URLs). • `suppliers.html`

- `../icons/logo.gif`

Relative URLs werden z. B. vom Browser bzgl. der *Basis-URL* *`http://www.acme.com/support/`* aufgelöst:

Beispiele 4.4 (Aufgelöste URLs). • `http://www.acme.com/support/suppliers.html`

- `http://www.acme.com/icons/logo.gif`

WWW-Adressen, weitere Formen

Beispiele 4.5 (Verschiedene Protokolle). • `http://www.w3.org/` (→ HTML-Seite)

- `http://www.w3.org/Icons/w3c_main` (→ Bild)
- `ftp://gd.tuwien.ac.at/pub/infosys/servers/`
- `mailto:stefan@iai.uni-bonn.de`

Adressierungskonzepte

- Oberbegriff: Uniform Resource Identifier (URI)
- Konkrete Adresse: Uniform Resource Locator (URL)
  - Beispiele: `http://...`, `ftp://...`, ...
  - „Wo ist es?“
- Logische Bezeichung: Uniform Resource Name (URN)
  - `urn:ietf:rfc:2141`, `urn:isbn:0-201-08372-8`
  - „Wie heißt es?“
  - Dauerhafter, ortsunabhängiger Bezeichner.
  - Muss nicht auflösbar sein.

Material zu URIs

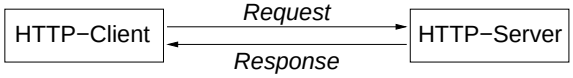
- Uniform Resource Locators (RFC1738)[25]
- URN Syntax (RFC2141)[26]
- Uniform Resource Identifiers (RFC 2396): Generic Syntax[27]
- Uniform Resource Identifiers (URIs), URLs, and Uniform Resource Names (URNs): Clarifications and Recommendations (RFC 3305)[28]

4.4 Transport: HTTP/1.1

- http://www.w3.org/Protocols/

Hypertext Transfer Protocol – HTTP/1.1

- Request/Response Protokoll der Anwendungsschicht
- Setzt auf TCP/IP auf



Austausch von Nachrichten

- **Anfrage** (Request): Methode + URI
- **Antwort** (Response): Metadaten (Typ, Datum, ...) + Dokument (Folge von *Bytes*)

Das HTTP-Nachrichtenformat ist angelehnt an *MIME (Multipurpose Internet Mail Extensions)*, RFC 2045 ff. MIME ist aber kein Bestandteil der HTTP-Definition.

HTTP — ein einfaches Beispiel ...

```
$ telnet www.uni-bonn.de 80
Trying 131.220.14.100...
Connected to www.uni-bonn.de.
Escape character is '^]'.
GET /studium HTTP/1.1
Host: www.uni-bonn.de

HTTP/1.1 301 Moved Permanently
```

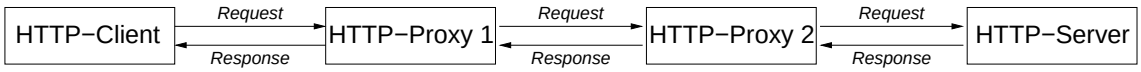
[25] http://tools.ietf.org/html/rfc1738
[26] http://tools.ietf.org/html/rfc2141
[27] http://tools.ietf.org/html/rfc2396
[28] http://tools.ietf.org/html/rfc3305

```
Server: nginx
Date: Tue, 16 May 2017 09:15:33 GMT
Content-Type: text/html
Content-Length: 178
Connection: keep-alive
Location: https://www.uni-bonn.de/studium

<html>
<head><title>301 Moved Permanently</title></head>
<body bgcolor="white">
<center><h1>301 Moved Permanently</h1></center>
<hr><center>nginx</center>
</body>
</html>
Connection closed by foreign host.
```

HTTP — zweiter Versuch

```
$ openssl s_client -quiet -connect www.uni-bonn.de:443
depth=3 C = DE, O = Deutsche Telekom AG, OU = T-TeleSec Trust Center, ...
...
GET /studium HTTP/1.1
Host: www.uni-bonn.de

HTTP/1.1 200 OK
Server: nginx
Date: Tue, 16 May 2017 09:17:15 GMT
Content-Type: text/html;charset=utf-8
Content-Length: 180287
Connection: keep-alive
Vary: Accept-Encoding
Content-Language: de
Expires: Sat, 1 Jan 2000 00:00:00 GMT
X-Ua-Compatible: IE=edge
Set-Cookie: I18N_LANGUAGE="de"; Path=/; secure; HttpOnly
X-Cache-Action: HIT
X-Cache-Hits: 8
Accept-Ranges: bytes
Strict-Transport-Security: max-age=15552001
X-Content-Type-Options: nosniff

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="de"
      lang="de">
...
</html>
```

HTTP — Terminologie (1)



Akteure

Server (*Origin Server*)

- stellt die Ressource unmittelbar zur Verfügung.

Proxy („Vertreter“)

- leitet HTTP-Requests und -Responses weiter,
- speichert evtl. die Responses (*Cache*),
- wird auch in Firewalls eingesetzt.

User Agent (Client)

- sendet den ursprünglichen Request,
- verarbeitet die Daten der Response.

HTTP — Terminologie (2)

Resource (Quelle)

- ein Datenobjekt oder ein Service
- über einen URI erreichbar
- kann in verschiedenen Repräsentationen vorliegen, zum Beispiel mehrere Sprachen, Datenformate, Auflösungen, ...

Entity (Ding, Objekt, Einheit)

- die „Nutzlast“ der Nachricht (Request oder Response) → *Entity-Body*
- inklusive Meta-Informationen → *Entity-Header Fields*

Datumsformate in HTTP — Beispiele (RFC 1123)

```
Expires: Thu, 11 Oct 2007 16:53:36 GMT
```

Oder relative Angabe in Sekunden:
`Expires: 120`

HTTP — Request

Aufbau HTTP-Request

- Erste Zeile (*Request-Line*): *Methode*, URI, Protokollversion
- Weitere *Header Lines*

- Zeilen im *Header* werden stets mit CR-LF abgeschlossen.
- Der *Header* endet mit einer Leerzeile (CR-LF).

Beispiel 4.6 (HTTP-Request).

```
GET / HTTP/1.1
Host: www.w3.org
Connection: keep-alive
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
DNT: 1
Accept-Encoding: gzip, deflate, sdch, br
Accept-Language: en-US,en;q=0.8
If-None-Match: "a0a5-54f724c72a0c0;89-3f26bd17a2f00-gzip"
If-Modified-Since: Sun, 14 May 2017 02:00:11 GMT
```

HTTP-Kommunikation beobachten

- Mit `telnet` auf der Kommandozeile, siehe oben
- Mit Firebug[29] oder Live HTTP Headers[30] für den Mozilla Firefox kann man Requests und Responses direkt mitlesen.

HTTP — Request — Methoden (1)

- Komplette Funktionalität für Informationssysteme:
  - Auffinden von Seiten
  - Einfügen
  - Updates
  - Löschen
  - ...
- HTTP stellt verschiedene Methoden zur Verfügung, die jeweils auf den angegebenen URI anzuwenden sind.
- GET, HEAD, POST, PUT, DELETE ...
- auch Basis von *REST* (Representational State Transfer)
  - Konzept zum Zugriff auf verteilte Informationssysteme
  - Adressierbarkeit
  - Zustandslosigkeit
  - Wohldefinierte Operationen

[29] https://addons.mozilla.org/de/firefox/addon/1843
[30] https://addons.mozilla.org/de/firefox/addon/3829

Quellen zu REST

- Roy Thomas Fielding: Architectural Styles and the Design of Network-based Software Architectures. Dissertation; University of California, Irvine, 2000.
- RESTful Web Services Cookbook von Subbu Allamaraju, O'Reilly 2010.
- http://en.wikipedia.org/wiki/Representational_state_transfer
  (siehe dort auch weiterführende Links)

HTTP — Request — Methoden (2)

Häufig genutzte Methoden

`GET` Die zum URI gehörige Information (Dokument o. ä.) soll als *Entity* in der Response-Nachricht mitgeschickt werden.

Kann durch eine `If-Modified-Since` Header-Zeile bedingt ausführbar gemacht werden.

`HEAD` Wie `GET`, sendet aber nur den *Header* zurück, nie den *Entity-Body*.

`PUT` Das im Request mitgeschickte *Entity* soll auf dem Server die Ressource ersetzen oder als neue Ressource angelegt werden.

HTTP — Request — Methoden (3)

Weitere Methoden

`DELETE` Auf dem Server soll die betroffene Repräsentation der Ressource gelöscht werden.

`POST` Das im Request mitgeschickte *Entity* soll vom Server im Kontext des URI verarbeitet werden.

Zum Beispiel werden Daten aktualisiert oder untergeordnete Ressourcen ergänzt.

`OPTIONS` fragt nach verfügbaren Optionen des angegebenen URIs.

...

HTTP — Request — URI

Beispiel: `http://www.w3.org/Consortium/contact`

- Ressourcen auf dem Ursprungsserver werden mit einem *absoluten Pfad* (aber ohne Host-Anteil) angesprochen:

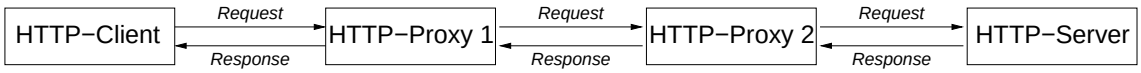
```
GET /Consortium/contact HTTP/1.1
Host: www.w3.org
```

- Ab HTTP/1.1 muß das Feld `Host` vorhanden sein.

  Es dient der Unterscheidung *virtueller Server* mit unterschiedlichen Namen, aber derselben IP-Nummer

- Ein Request an einen Proxy muss ein absolutes URI angeben:

```
GET http://www.w3.org/Consortium/contact HTTP/1.1
Host: www.w3.org
```



HTTP — Header Fields

Es gibt drei Kategorien von Header Fields:

- *Request Header Fields*: Informationen zur Anfrage
- *Entity Header Fields*: Metainformationen zu Nutzlast
- *Response Header Fields*: Informationen zur Antwort

HTTP — Request — Header Fields (1)

`If-Modified-Since`

Das Entity nur zurückliefern, wenn die Ressource neuer ist als der angegebene Zeitpunkt.

```
If-Modified-Since: Sun, 14 May 2017 02:00:11 GMT
```

`User-Agent`

Informationen über Browser, Betriebssystem, Versionen. Mögliche Datenschutz- und Sicherheitsprobleme!

```
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 ↩
(KHTML, like Gecko) Chrome/58.0.3029.81 Safari/537.36
```

`Referer`

URI der Seite, die den Link auf die angeforderte URI enthielt. Mögliche Datenschutzprobleme!

```
GET /Consortium/ HTTP/1.1
Host: www.w3.org
Referer: https://www.w3.org/standards/
```

HTTP — Request Header Fields (2)

`Authorization`

Geheime Zugangsdaten gemäß gesonderter Spezifikation.

Hier User-Id:Passwort für *Basic Authentication Scheme*, base64-codiert.

```
Authorization: Basic YWRtaW46Z2VoZWlt
```

`Range`

Nur den angegebenen Ausschnitt der angeforderten Ressource liefern.

Nützlich, wenn ein früherer Versuch nach teilweiser Übertragung abgebrochen wurde.

```
Range: bytes 0-499/1234
```

```
Range: bytes 500-1233/1234
```

HTTP — Entity Header Fields

Metainformationen über den *Message Body*

`Content-Length`

Länge des Entity im Message Body.

Fehlt diese, wird das Ende des *Message Body* durch Schließen der Verbindung angezeigt (`Connection: close`), oder es muß *Chunked Encoding* angewandt werden (`Transfer-Encoding: chunked`, s. u.)

```
Content-Length: 3495
```

`Content-Type`

„Medientyp“ des *Entity-Body* (analog zu MIME), Syntax: *type/subtype*.

```
Content-Type: text/html; charset=ISO-8859-1
```

Weitere Typen:

```
text/plain, image/gif, image/jpeg, audio/midi, video/mpeg, application/pdf, ...
```

HTTP — Response

Aufbau HTTP-Response

- Erste Zeile (*Statuszeile*): Protokollversion, *Status-Code*, Statustext
- Weitere *Header Lines*, danach eine Leerzeile (CR-LF)

Beispiel 4.7 (HTTP-Response).

```
HTTP/1.1 200 OK
Date: Tue, 16 May 2017 10:17:25 GMT
Content-Location: Home.html
Vary: negotiate,accept,Accept-Encoding
TCN: choice
Last-Modified: Sun, 14 May 2017 02:00:11 GMT
ETag: "a098-54f724c72a0c0;89-3f26bd17a2f00-gzip"
Accept-Ranges: bytes
Content-Encoding: gzip
Cache-Control: max-age=600
Expires: Tue, 16 May 2017 10:27:25 GMT
P3P: policyref="http://www.w3.org/2014/08/p3p.xml"
Content-Length: 10057
Content-Type: text/html; charset=utf-8
Strict-Transport-Security: max-age=15552000; includeSubdomains; preload
Content-Security-Policy: upgrade-insecure-requests

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
...
</html>
```

HTTP — Response — Status-Codes (Auswahl)

Informational 1xx

```
100 Continue
101 Switching Protocols
```

Successful 2xx

```
200 OK
201 Created
202 Accepted
204 No Content
205 Reset Content
206 Partial Content
```

Redirection 3xx

```
300 Multiple Choices
301 Moved Permanently
303 See Other
304 Not Modified
305 Use Proxy
307 Temporary Redirect
```

Client Error 4xx

```
400 Bad Request
401 Unauthorized
403 Forbidden
404 Not Found
405 Method Not Allowed
408 Request Timeout
410 Gone
415 Unsupported Media Type
```

Server Error 5xx

```
500 Internal Server Error
501 Not Implemented
502 Bad Gateway
503 Service Unavailable
504 Gateway Timeout
```

HTTP — Response — Status-Codes (2)

Beispiel 4.8 (Umleitung).

```
GET /XML HTTP/1.0
```

```
HTTP/1.0 301 Moved Permanently
Date: Wed, 01 Nov 2000 19:30:19 GMT
Server: Apache/1.3.12 (Unix)  (SuSE/Linux)
Location: http://wob.informatik.uni-bonn.de/Wob/de/view/...
Content-Type: text/html; charset=iso-8859-1
Age: 0
Connection: close

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<HTML><HEAD>
<TITLE>301 Moved Permanently</TITLE>
</HEAD><BODY>
<H1>Moved Permanently</H1>
The document has moved <A HREF="http://wob...
```

HTTP — Response — Status-Codes (3)

Beispiel 4.9 (Unsinnige Anfrage).

```
Hallo
```

```
HTTP/1.0 400 Bad Request
Server: Squid/2.2.STABLE5
Mime-Version: 1.0
Date: Wed, 01 Nov 2000 19:31:11 GMT
Content-Type: text/html
Content-Length: 826
Expires: Wed, 01 Nov 2000 19:31:11 GMT
Proxy-Connection: close

<HTML><HEAD>
...
```

HTTP — Response — Status-Codes (4)

Beispiel 4.10 (Anfrage, für die ein Passwort benötigt wird).

```
GET /internal/index.html HTTP/1.0
```

```
HTTP/1.0 401 Unauthorized
Date: Wed, 01 Nov 2000 19:28:36 GMT
Server: Apache/1.3.12 (Unix)  (SuSE/Linux)
WWW-Authenticate: Basic realm="Wob User"
Content-Length: 300
Content-Type: text/html
Age: 0
Connection: close

...
```

HTTP — Response — Status-Codes (5)

Beispiel 4.11 (Nichtexistente Ressource).

```
GET /foo.html HTTP/1.0
```

```
HTTP/1.0 404 Not Found
Date: Wed, 01 Nov 2000 19:24:59 GMT
Server: Apache/1.3.12 (Unix)  (SuSE/Linux)
Content-Type: text/html; charset=iso-8859-1
Age: 0
Connection: close

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<HTML><HEAD>
<TITLE>404 Not Found</TITLE>
```

```
</HEAD><BODY>
...
```

HTTP — Response Header Fields (1)

`Date`

Datum der Response-Nachricht (nicht der Ressource)

```
Date: Sun, 14 Oct 2007 15:04:21 GMT
```

`Last-Modified`

Zeitpunkt, an dem die Ressource zuletzt geändert wurde

```
Last-Modified: Fri, 12 Oct 2007 20:40:06 GMT
```

`Expires`

Zeitpunkt, an dem die Response ungültig (veraltet) sein wird

```
Expires: Sun, 14 Oct 2007 15:14:21 GMT
```

`Retry-After`

Vorgeschlagene Wartezeit vor erneutem Zugriff bei bestimmten Fehlercodes

```
Retry-After: 120
```

HTTP — Response Header Fields (2)

`Server`

Informationen über Server, Betriebssystem, Versionen. Mögliche Datenschutz- und Sicherheitsprobleme!

```
Server: IBM_HTTP_SERVER/1.3.28.1-PQ98444 ↩ Apache/1.3.28 (Unix) PHP/4.3.2
```

`Location`

URI einer anderen Ressource bei Umleitungen

```
Location: http://www.w3.org/pub/WWW/People.html
```

`WWW-Authenticate`

Bei `401 Unauthorized` die Aufforderung, den Request mit passenden Authentisierungsangaben (`Authorization:`) zu senden

```
WWW-Authenticate: Basic realm="SysAdmin"
```

HTTP — Persistente Verbindungen (1)

- Ursprünglich in HTTP/1.0: eine separate TCP-Verbindung für jede Anfrage
- Die meisten Seiten generieren mehrere Anfragen an denselben Server (Bilder, CSS, Scripts usw.).
- Auf- und Abbauen vieler Verbindungen erhöht Netzlast und Wartezeiten
- **Idee**: Mehrere Request/Response-Vorgänge über eine einzige TCP-Verbindung abwickeln.
- *Persistente Verbindungen* sind die Standardmethode in HTTP/1.1.

HTTP — Persistente Verbindungen (2)

Request- bzw. Response-Header Field

HTTP/1.1: Schließen der Verbindung ankündigen oder verlangen

`Connection: close`

HTTP/1.0 (Erweiterung): Offenhaltung der Verbindung anfordern

`Connection: Keep-Alive`

Chunked Transfer Coding

- Fehlt das Feld `Content-Length`, so kann das Ende einer Nachricht nur durch das Schließen der Verbindung signalisiert werden. Dieses ist bei persistenten Verbindungen unerwünscht.
- Idee: Den Message Body in einzelne Blöcke (chunks) mit expliziter Längenangabe unterteilen.
- Response Header Field: `Transfer-Encoding: chunked`

Transfer-Encoding

`Transfer-Encoding: chunked`

HTTP — Persistente Verbindungen (3)

Beispiel 4.12 (Chunked Transfer Coding).

```
HTTP/1.1 200 OK
Date: Thu, 18 Feb 1999 13:20:41 GMT
Server: Apache/1.3.1 (Unix)
Transfer-Encoding: chunked
Content-Type: text/plain

d
```

```
Erste Zeile,

16
zweite folgt sogleich

0

HTTP/1.1 200 OK
Date: Thu, 18 Feb 1999 13:20:43 GMT
...
```

HTTP — Persistente Verbindungen (4)

Pipelining

- Lange Paketlaufzeiten können immer noch zu unnötigen Wartezeiten führen.
- Abhilfe bringt das *Pipelining*.
- Client setzt mehrere Requests auf einer persistenten Verbindung ab, ohne auf die Antworten zu warten.
- Der Server liefert die zugehörigen Responses in der Reihenfolge der Requests.
- Beide Seiten dürfen die Verbindung jederzeit abbrechen.
- Verbindungsstatus muss also laufend geprüft werden.

HTTP — Sicherheit und Datenschutz

Sicherheitslücken

- Offene Übertragung der Anfragen und Antworten
- Fehlende Authentisierung des Servers gegenüber dem Client (und umgekehrt)

Lösung

- Kryptografische Verfahren
- Secure Sockets Layer (SSL) / Transport Level Security (TLS)
- Vermeidet Abhören durch *Verschlüsselung* der Transportschicht
- Sichert die Integrität der Verbindung
- Authentisierung (insbes.) der Server durch *Zertifikate*

4.5 HTTP/2

Material zu HTTP/2

- https://http2.github.io/
- http://blog.scottlogic.com/2014/11/07/http-2-a-quick-look.html
- http://undertow.io/blog/2015/04/27/An-in-depth-overview-of-HTTP2.html
- http://en.wikipedia.org/wiki/HTTP/2
- http://heise.de/-2650979

HTTP/2

- HTTP/2 ist ein neues Übertragungsverfahren für HTTP.
- Die HTTP-Methoden, Status-Codes und deren Semantik bleiben unverändert.
- Die APIs für HTTP/1.x können weiterverwendet werden.
- Der Fokus liegt auf Performanz:
  - vom Endbenutzer wahrgenommene Wartezeit,
  - Belastung von Netzwerk- und Server-Ressourcen.

HTTP/2 — Spezifikation

Hypertext Transfer Protocol version 2

- http://httpwg.org/specs/rfc7540.html
- Status: Internet Standard
- Weitere Infos: https://http2.github.io/

HTTP/2 — Motivation

Ausgangslage

- Webseiten laden üblicherweise viele Ressourcen nach.
- Von einzelnen Servern kommen oft zahlreiche Ressourcen.

HTTP/1.x

- HTTP 1.0: sequentielle Requests, schlecht bei hoher Latenz im Netz.
- HTTP 1.1: persistente Verbindung und Pipelining.

- Heutige Browser öffnen parallel mehrere TCP-Verbindungen. Das bindet Ressourcen auf den Servern.

Wesentliche Techniken

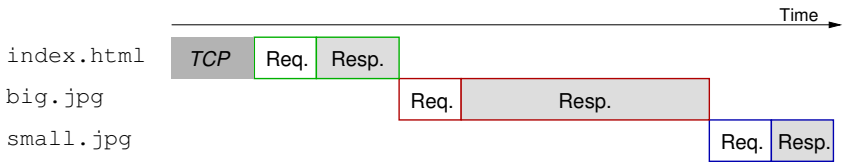
- Binary Protocol, Streams, Frames
- Multiplexing, *Single Connection*
- Header Compression
- Server Push

HTTP/2 — Streams

- Logisch: eine einzelne Request/Response-Kommunikation, *Stream-Id*
- Physisch: bidirektionale Folge von *Frames*
- Streams erlauben gleichzeitige (multiplexed) Kommunikationvorgänge auf einer einzigen TCP-Verbindung.

HTTP — Timing

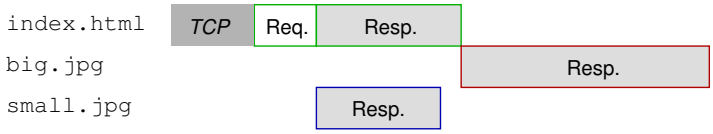
HTTP/1.0 mit `Connection: Keep-Alive`



HTTP/1.1 mit Pipelining

HTTP/2 mit Multiplexing

HTTP/2 mit Multiplexing und Push

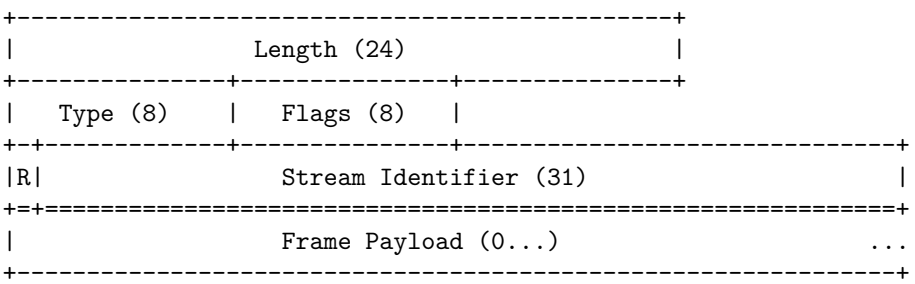


HTTP/2 — Frames

Frames

- Jeder Teil der Komunikation zwischen Client und Server wird in einen „binären Frame“ verpackt, bevor er übertragen wird.
- Frame-Typen: HEADERS, CONTINUATION, DATA, PUSH_PROMISE, RST_STREAM, SETTINGS, PING, GOAWAY, PRIORITY, WINDOW_UPDATE

Frame Format



HTTP/2 — Server Push

- Der Server kann Ressourcen senden, die der Client benötigen oder anfordern wird (Bilder, Skripte, ...).
- PUSH_PROMISE kündigt dies an.
- Abbruch mit RST_STREAM möglich.
- Der Client kann danach die nächste Seite auf derselben Connection laden.

HTTP/2 — Headers

HTTP/1.1

```
GET /resource HTTP/1.1
  Host: example.org
  Accept: image/jpeg
```

HTTP/2

```
HEADERS
    + END_STREAM
    + END_HEADERS
      :method = GET
      :scheme = https
      :path = /resource
      host = example.org
      accept = image/jpeg
```

HTTP/1.1

```
HTTP/1.1 304 Not Modified
  ETag: "xyzzy"
  Expires: Thu, 23 Jan ...
```

HTTP/2

```
HEADERS
    + END_STREAM
    + END_HEADERS
      :status = 304
      etag = "xyzzy"
      expires = Thu, 23 Jan ...
```

HTTP/1.1

```
POST /resource HTTP/1.1
  Host: example.org
  Content-Type: image/jpeg
  Content-Length: 123

  {binary data}
```

HTTP/2

```
HEADERS
    - END_STREAM
    - END_HEADERS
      :method = POST
      :path = /resource
      :scheme = https

CONTINUATION
    + END_HEADERS
      content-type = image/jpeg
      host = example.org
      content-length = 123

DATA
    + END_STREAM
  {binary data}
```

HTTP/1.1

```
HTTP/1.1 200 OK
  Content-Type: image/jpeg
  Content-Length: 123

  {binary data}
```

HTTP/2

```
HEADERS
    - END_STREAM
    + END_HEADERS
      :status = 200
      content-type = image/jpeg
      content-length = 123

DATA
    + END_STREAM
  {binary data}
```

HTTP/2 — Upgrading

Beispiel 4.13 (Upgrade von http:). `GET / HTTP/1.1`
`Host: server.example.com`
`Connection: Upgrade, HTTP2-Settings`
`Upgrade: h2c`
`HTTP2-Settings: <base64url encoding of HTTP/2 SETTINGS>`

Entweder:

```
HTTP/1.1 200 OK
Content-Length: 243
Content-Type: text/html
```

oder:

```
HTTP/1.1 101 Switching Protocols
Connection: Upgrade
Upgrade: h2c
```

ALPN bei https:

Application Layer Protocol Negotiation (ALPN): Eine Erweiterung von TLS, die das nach dem TLS Handshake zu nutzende Protokoll aushandelt.

HTTP/2 — Encryption

Ursprünglicher Plan

- Das Protokoll sollte nur verschlüsselte Übertragung unterstützen.

Kritik / Probleme

- Verschlüsselung benötigt Rechenzeit auf beiden Seiten.
- Externes Caching nicht mehr möglich ⇒ Ladezeiten steigen

⇒ Ziele verfehlt

Kompromiss

- Optional plain-text version
- Unsicher! Trend geht allgemein zur Verschlüsselung.
- Plain-text version wird z. Z. nicht von allen Browsern unterstützt.

HTTP/2 — Kritikpunkte

- Der Zeitplan war zu knapp, weswegen nur Googles SPDY Protokoll als Basis in Frage kam.
- Nicht alle Kritikpunkte an SPDY wurden behoben.
- Das Protokoll ist übermäßig komplex.
- Das Protokoll verletzt die Schichtung. (Dupliziert die Flusskontrolle der Transportschicht TCP.)
- Privatsphäre/Verschlüsselung (s. vorherige Folie)
- Letztendlich nur eine Übergangslösung auf dem Weg zu HTTP/3?

Links (insbesondere) zu den Kritikpunkten

- http://en.wikipedia.org/wiki/HTTP/2#Criticisms
- https://lists.w3.org/Archives/Public/ietf-http-wg/2014AprJun/0814.html
- https://lists.w3.org/Archives/Public/ietf-http-wg/2014AprJun/0807.html
- http://blog.scottlogic.com/2014/11/07/http-2-a-quick-look.html
- http://queue.acm.org/detail.cfm?id=2716278

4.6 Inhalte im WWW

HTML

HTML (Hypertext Markup Language)

- Markup-Sprache, ursprünglich definiert in SGML
- Idee: Struktur beschreiben (Textstruktur, Semantik)
  - Überschriften, Absätze, Aufzählungen, Betonungen und Hervorhebungen, ...
  - *Links*, *Bilder*, andere Medien, ...
- Präsentation separat beschreiben (CSS)
- Nicht nur visuell, alternative Präsentationen sind möglich, z. B. Audio
- Plattformunabhängig, barrierefrei (bei korrekter Verwendung)

HTML — Entwicklung

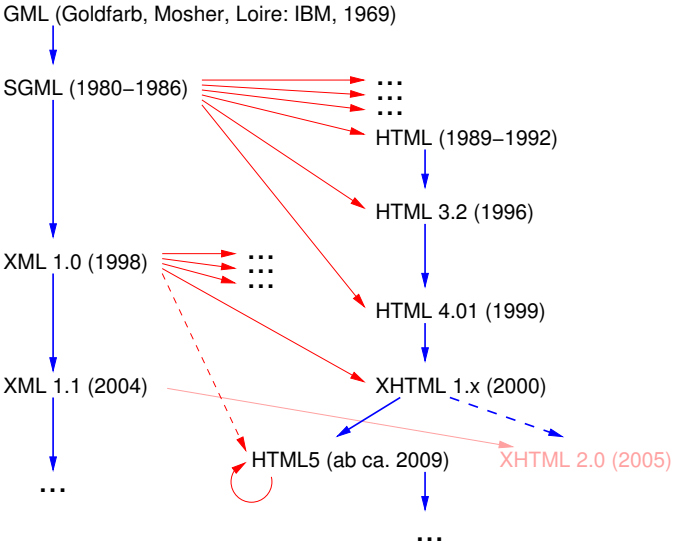
Probleme im realen WWW

- Proprietäre Browser-Erweiterungen, „Browser-Krieg“
- Formatierungsattribute
- Missbrauch von Sprachkonstrukten, z. B. „pixelgenaues“ Design mit Tabellen
- Neue Anwendungsfelder: E-Commerce, „Web 2.0“, z. B. „Office im Browser“, ... ⇒ *Web*-Anwendungen

Lösungen

- Richtlinien für „sauberen“ HTML-Einsatz, auch mit dem Ziel *Barrierefreiheit*
- Eingeschränktes Vokabular *strict* bei HTML 4
- Separate Formatierung mit CSS
- *Semantisches Markup* bei HTML5
- Wohldefinierte APIs für JavaScript

Markup-Sprachen



XHTML 1.0

Beispiel 4.14. `<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">`

```
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>
      HelloWorld.html
    </title>
  </head>
  <body>
    <p>
      Hello World!
    </p>
    <ol>
      <li>rot</li>
      <li>grün</li>
      <li>blau</li>
    </ol>
  </body>
</html>
```

XHTML ist eine XML-basierte Sprache.

HTML5

Beispiel 4.15. `<!DOCTYPE html>`

```
<html lang="de">
  <head>
    <title>Hallo Welt!</title>
  </head>
  <body>
```

```
    <h1>Hallo Welt!</h1>
    <ol>
      <li>rot/li>
      <li>grün
      <li>blau
    </ol>
  </body>
</html>
```

HTML5 basiert weder auf XML noch auf SGML.

Es gibt eine XML-Syntax für HTML5

Beispiel GML[31]

GML ist ein Vorläufer von SGML.

```
:frontm.
:titlep.
.se tl = 'Document Formatting Systems:  Survey, Concepts, and Issues'
.se ea = 'Extended Abstract'
:title.&tl. [&ea.]:fnref refid=funds.
:fn id-funds.
This research was supported in part by the National Science Foundation
under grant number MCS-7826285.
:efn.
:author.Alan Shaw
:author.Richard Furuta
:author.Jeffrey Scofleld
:address.
:aline.Department of Computer Science
:aline.University of Washington
:aline.Seattle, Washington 98195, U.S.A.
:eaddress.
:etitlep.
:abstract.
:p.Formatting, the final part of the document preparation process , is
concerned with the physical layout of a document for hard and soft
copy media....  Our aims are to characterize the formatting problem
and its relation to other aspects of document processing , to evaluate
several representative and seminal systems, and to describe some
issues and problems relevant to future systems.
:body.
:h2.The Formatting Problem
:p. In order to discuss formatters and their functions and to distinguish
formatting from other aspects of document preparation, it is
convenient to use an :hpl.object:ehpl. model o f documents [Shaw 80],
somewhat analogous to that in programming languages.
:p.A document is an object composed of a hierarchy of more primitive
objects ....
:h2.Representative and Seminal Systems
```

[31] Aus: Document Formatting Systems: Survey, Concepts, and Issues Richard Furuta, Jeffrey Scofield, and Alan Shaw. Computing Surveys,Vol. 14, No. 3, September 1982

```
:h3.Pure Formatters
:p.Some typical first generation formatters...
```

Technologien

Grundlegende Technologien für Web-Inhalte

- Semantische Beschreibung von Webseiten(inhalten), strukturierte Dokumente
- Stylesheets für die konkrete Darstellung
- JavaScript für client-seitige Programmierung
- Serverseitige Werkzeuge

Separation of Concerns

Immer sauber trennen:

1. Inhalte
2. Design, Formatierung
3. Verhalten (falls vorhanden)

4.7 HTML5

HTML5 — zwei parallele Ansätze

W3C

- **HTML5** – A vocabulary and associated APIs for HTML and XHTML[32] – W3C Recommendation 28 October 2014
- **HTML 5.2**[33] – W3C Recommendation, 14 December 2017
- Klassischer Spezifikationsprozess
- Feste Versionen: 5.0, 5.1, 5.2, ...

WHATWG

Web Hypertext Application Technology Working Group[34]

HTML Living Standard[35] – Last Updated 22 May 2017

[32] http://www.w3.org/TR/html5/
[33] https://www.w3.org/TR/html5/
[34] http://www.whatwg.org/
[35] http://www.whatwg.org/specs/web-apps/current-work/multipage/

- Kontinuierlicher Prozess
- Greift Entwicklungen der Browserhersteller auf.
- *... how to write a browser ... that handles all previous versions of HTML, as well as all the latest features.*

HTML5

Anwendungsgebiete

- Webseiten, strukturierte Dokumente
- *Web Applications*, *Rich Internet Applications*
  - Shopping
  - Office, Publishing
  - E-Mail, Chat, ...
  - Games
  - u. v. a. m.
- Plattformen: Desktop, Mobile, TV, Headless, ...

HTML5 — Semantische Beschreibung

Semantik

- *Was* will ich beschreiben.
- Elemente *repräsentieren* „Dinge“, sie haben eine innere Bedeutung (Semantik).
- **Nicht:** *Wie* soll es aussehen.
- Das Design erledigt später *CSS*.

Darstellung

- Webseiten brauchen nicht in jedem Browser gleich auszusehen.
- Webseiten können nicht in jedem Browser gleich aussehen.
- Webseiten sollen manchmal ihr Aussehen verändern.

Beispiel 4.16 (semantisches Markup). **Einleitung**

```
<h1>Einleitung</h1>
```

Beispiel (unklare Semantik)

Einleitung

```
<div class="h1">Einleitung</div>
```

Beispiel (noch schlimmer)

Einleitung

```
<div style="font-size: 120%; font-weight: bold;">Einleitung</div>
```

Beispiel 4.17 (Betonung). `<p>Cats <em>are</em> cute animals.</p>`

Beispiel 4.18 (Wichtig).
```
<p><strong>Warning.</strong> This dungeon is dangerous.
<strong>Avoid the ducks.</strong> Take any gold you find.
<strong><strong>Do not take any of the diamonds</strong>,
they are explosive and <strong>will destroy anything within
ten meters.</strong></strong> You have been warned.</p>
```

Beispiel 4.19 („Kleingedrucktes“). `<p><small>Copyright 2014</small></p>`

Beispiel 4.20 (Begriffe).
```
<p>The term <i>prose content</i> is defined above.</p>
<p>There is a certain <i lang="fr">je ne sais quoi</i> in
the air.</p>
```

Semantik — Vergleich mit HTML 4

HTML5 Text-level semantics

- The `em` element represents stress emphasis of its contents.
- The `strong` element represents strong importance, seriousness, or urgency for its contents.

HTML 4.01 Structured text

- **EM**: Indicates emphasis.
- **STRONG**: Indicates stronger emphasis.

HTML5 Text-level semantics

- The `i` element represents a span of text in an alternate voice or mood, or otherwise offset from the normal prose, such as a taxonomic designation, a technical term, an idiomatic phrase from another language, a thought, a ship name, or some other prose whose typical typographic presentation is italicized.

- The `b` element represents a span of text to be stylistically offset from the normal prose without conveying any extra importance, such as key words in a document abstract, product names in a review, or other spans of text whose typical typographic presentation is boldened.

HTML 4.01 Font style elements

- **I**: Renders as italic text style.
- **B**: Renders as bold text style.

HTML5 — Semantische Beschreibung

Die HTML-Elemente (in Gruppen)

- The root element: `html`
- Document metadata: `head`, `title`, `link`, `meta`, ...
- Sections: `body`, `article`, `section`, `header`, `nav`, ...
- Grouping Content: `p`, `ul`, `ol`, `main`, `div`, ...
- Text-level semantics: `a`, `em`, `strong`, `time`, `span`, ...
- Edits: `ins`, `del`
- Embedded content: `img`, `audio`, `video`, `math`, `svg`, ...
- Links: `a`, `area`
- Tabular data: `table`, `thead`, `tbody`, `tr`, `td`, ...
- Forms: `form`, `label`, `input`, `button`, ...
- Interactive elements: `details`, `summary`, `menu`, `dialog`, ...
- Scripting: `script`, `noscript`, `template`, `canvas`

HTML5 — Strukturbeispiel

Abgeleitet vom HTML5 Bones[36]

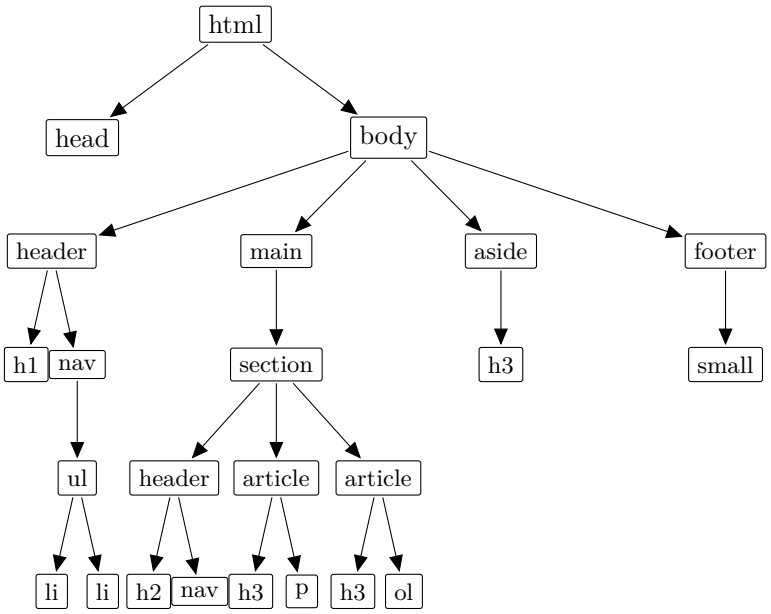
- Einfaches Template für HTML5-Seiten.
- Kommentare erläutern die Anwendung der HTML5-Elemente.

[36] http://www.html5bones.com/

- Ohne JavaScript

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <title>PAGE TITLE</title>
    <meta name="description" content="Just an example" />
    <meta name="viewport"
          content="width=device-width, initial-scale=1.0,
                   shrink-to-fit=no" />
    <link href="css/normalize.css" rel="stylesheet" media="all">
    <link href="css/styles.css" rel="stylesheet" media="all">
  </head>
  <body>
    <header role="banner">...</header>
    <div class="wrap">
      <main role="main">...</main>
      <aside role="complementary">...</aside>
    </div>
    <footer role="contentinfo">...</footer>
  </body>
</html>
```

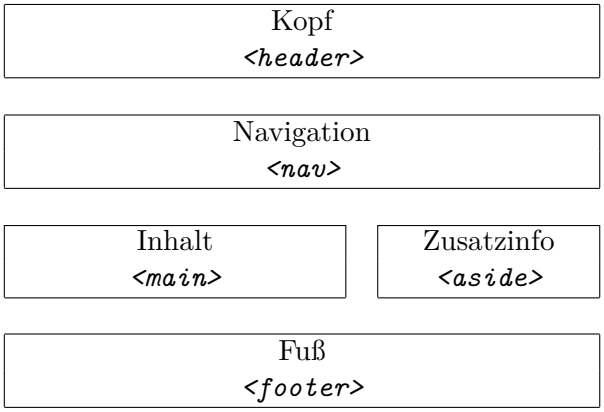
```
<!DOCTYPE html>
<html lang="en">
  <head>...</head>
  <body>
    <header role="banner">...</header>
    <div class="wrap">
      <main role="main">...</main>
      <aside role="complementary">
        <h3>Did you know?</h3>
        <p>
          The article and section elements cause a lot of confusion
          when people are trying to decide how and when to use them.
          The article: <a href="http://...">section or article?</a>
          might help you choose.
        </p>
      </aside>
    </div>
    <footer role="contentinfo">
      <small>
        Copyright &copy; <time datetime="2016">2016</time>
      </small>
    </footer>
  </body>
</html>
```



Das Beispiel als Baum (nicht komplett)

Semantische Elemente für die Seitenstruktur

Beispiel 4.21 (Seitenstruktur, logisch).



Das visuelle Layout kann auch ganz anders aussehen!

HTML5 — Sections

Elemente (Auswahl)

`body` Hauptinhalt

`header` Einleitendes Material (Überschrift, Navigation, ...)

`nav` Navigationsbereich

`section` Abschnitt (inhaltlich)

`h1`, `...`, `h6` Überschriften

`article` In sich geschlossener Teilinhalt

`aside` Zusatzinformationen

`footer` Fußbereich, Metadaten

address Kontaktinfos

HTML5 — Sections

The `article` element

- The `article` element represents a complete, or self-contained composition in a document, page, application, or site and that is, in principle, independently distributable or reusable, e. g. in syndication. This could be
  - a forum post,
  - a magazine or newspaper article,
  - a blog entry,
  - a user-submitted comment,
  - an interactive widget or gadget,
  - or any other independent item of content.

HTML5 — Sections

Beispiel 4.22 (The `article` element). `<article itemscope itemtype="http://schema.org/BlogPostin`

```
  <header>
    <h1 itemprop="headline">The Very First Rule of Life</h1>
    <p><time itemprop="datePublished" datetime="2009-10-09">
      3 days ago</time></p>
  </header>
  <p>If there's a microphone anywhere near you, ...</p>
  <p>...</p>
  <section>
    <h1>Comments</h1>
    <article id="c1" itemprop="comment" itemscope
             itemtype="http://schema.org/UserComments">
      <link itemprop="url" href="#c1">
      <footer><p>Posted by: ...</p>...</footer>
      <p>Yeah! Especially when talking about...</p>
    </article>
    <article>...</article>
  </section>
</article>
```

HTML5 — Sections

The `header` element

- The `header` element represents a group of introductory or navigational aids.

Beachte: `header` $\neq$ `head`

The `address` element

- The `address` element represents the contact information for its nearest `article` or `body` element ancestor. If that is the `body` element, then the contact information applies to the document as a whole.
- The `address` element must not be used to represent arbitrary addresses (e.g. postal addresses), unless those addresses are in fact the relevant contact information. (The `p` element is the appropriate element for marking up postal addresses in general.)
- The `address` element must not contain information other than contact information.
- Typically, the `address` element would be included along with other information in a `footer` element.

Beispiel 4.23. `<footer>`

```
  <address>
    For more details, contact
    <a href="mailto:js@example.com">John Smith</a>.
  </address>
  <p><small>© copyright 2038 Example Corp.</small></p>
</footer>
```

HTML5 — Sections

The `h1`, `h2`, `h3`, `h4`, `h5`, and `h6` elements

- These elements represent headings for their sections.
- These elements have a rank given by the number in their name. The `h1` element is said to have the highest rank, the `h6` element has the lowest rank, and two elements with the same name have equal rank.

HTML5 — Sections

Beispiel 4.24 (mit Sections). `<body>`

```
  <h1>Let's call it a draw(ing surface)</h1>
  <section>
    <h2>Diving in</h2>
  </section>
```

```
  <section>
    <h2>Simple shapes</h2>
  </section>
  <section>
    <h2>Canvas coordinates</h2>
    <section>
      <h3>Canvas coordinates diagram</h3>
    </section>
  </section>
  <section>
    <h2>Paths</h2>
  </section>
</body>
```

Beispiel 4.25 (ohne Sections). `<body>`

```
  <h1>Let's call it a draw(ing surface)</h1>
  <h2>Diving in</h2>
  <h2>Simple shapes</h2>
  <h2>Canvas coordinates</h2>
  <h3>Canvas coordinates diagram</h3>
  <h2>Paths</h2>
</body>
```

HTML5 — Grouping Content

Elemente (Auswahl)

`p` Absatz

`hr` Themenwechsel zwischen Absätzen, oder z. B. Szenenwechsel

`pre` Vorformatierter Text (Code, ASCII-Art, ...)

`ol` Sortierte Auflistung

`ul` Unsortierte Auflistung

`li` Listenelement

`dl`, `dt`, `dd` Liste von Name-Wert-Paaren

`figure` Container für Inhalt und „Caption“

`figcaption` Bildunterschrift

`main` Container für „dominante“ Inhalte eines anderen Elements

`div` Keine spezielle Bedeutung, repräsentiert/gruppiert nur seine Kindelemente.

HTML5 — Grouping Content

The `ul` element

- The `ul` element represents a list of items, where the order of the items is not important — that is, where changing the order would not materially change the meaning of the document.
- The items of the list are the `li` element child nodes of the `ul` element

The `ol` element

- The ol element represents a list of items, where the items have been intentionally ordered, such that changing the order would change the meaning of the document.

HTML5 — Grouping Content

Beispiel 4.26 (unordered). `<p>I have lived in the following countries:</p>`

```
<ul>
  <li>Norway
  <li>Switzerland
  <li>United Kingdom
  <li>United States
</ul>
```

Beispiel 4.27 (ordered). `<p>I have lived in the following countries (given in the order of when I first lived there):`

```
</p>
<ol>
 <li>Switzerland
 <li>United Kingdom
 <li>United States
 <li>Norway
</ol>
```

HTML5 — Text-level Semantics

The `time` element

The `time` element represents its contents, along with a machine-readable form of those contents in the datetime attribute. The kind of content is limited to various kinds of dates, times, time-zone offsets, and durations, ...

Beispiel 4.28. `<div class="vevent">`

```
  <span class="summary">Web 2.0 Conference</span>:
  <time class="dtstart" datetime="2005-10-05">October 5</time>
  - <time class="dtend" datetime="2005-10-07">7</time>,
  at the <span class="location">Argent Hotel, ...</span>
</div>
```

```
<p>We stopped talking at
<time datetime="2006-09-24T05:00-07:00">5am the next
morning</time>.
</p>
```

HTML5 — Text-level Semantics

The `strong` element

- The `strong` element represents strong importance for its contents.
- The relative level of importance of a piece of content is given by its number of ancestor `strong` elements; each `strong` element increases the importance of its contents.

Beispiel 4.29.
```
<p><strong>Warning.</strong> This dungeon is dangerous.
<strong>Avoid the ducks.</strong> Take any gold you find.
<strong><strong>Do not take any of the diamonds</strong>,
they are explosive and <strong>will destroy anything within
ten meters.</strong></strong> You have been warned.</p>
```

HTML5 — Embedded Content

Eingebettete Medien

- Bilder, Video, Audio
- `map`, `area`: Image Maps
- MathML, SVG
- Canvas
- ...

Beispiel 4.30 (Bild).
```
<p>You are standing in an open field west of a house.
  <img src="house.jpeg"
       alt="The house is white, with a boarded front door.">
  There is a small mailbox here.
</p>
```

... the `alt` attribute must be specified and its value must not be empty; the value must be an appropriate replacement for the image. ...

Scalable Vector Graphics (SVG) 1.1 (Second Edition)[37]

[37] http://www.w3.org/TR/SVG11/

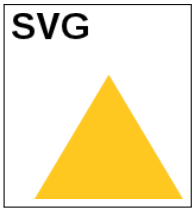
HTML5 — SVG

Beispiel 4.31 (HTML mit eingebettetem SVG).

```
<!DOCTYPE html>
<html lang="de">
  <head>
    <meta charset="utf-8">
    <title>SVG Beispiel</title>
  </head>
  <body>
    <h1>SVG</h1>
    <svg width="150px" height="130px">
      <polygon points="20,105 82.5,0 145,105"
               fill="#ffc821"/>
    </svg>
  </body>
</html>
```

HTML5 — SVG

Beispiel 4.32 (Ausgabe).



HTML5 — SVG

Beispiel 4.33 (HTML mit externem SVG).

```
<!DOCTYPE html>
<html lang="de">
  <head>
    <meta charset="utf-8">
    <title>SVG Beispiel</title>
  </head>
  <body>
    <h1>SVG</h1>
    <img src="triangle-svg.svg" alt="Ein Dreieck" />
  </body>
</html>
```

Beispiel 4.34 (SVG verwendet einen Namespace).

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
          "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg xmlns="http://www.w3.org/2000/svg"
     width="150px" height="130px">
  <polygon points="20,105 82.5,0 145,105" fill="#ffc821"/>
</svg>
```

Mathematical Markup Language (MathML) Version 3.0 2nd Edition[38]

HTML5 — MathML

The quadratic formula

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

HTML5 — MathML

```
<!DOCTYPE html>
<html>
  <head><title>The quadratic formula</title></head>
  <body>
    <h1>The quadratic formula</h1>
    <p>
      <math>
        <mi>x</mi>
        <mo>=</mo>
        <mfrac>
          <mrow>
            <mo form="prefix">-</mo> <mi>b</mi>
            <mo>&PlusMinus;</mo>
            <msqrt>
              <msup> <mi>b</mi> <mn>2</mn> </msup>
              <mo>-</mo>
              <mn>4</mn> <mo> </mo> <mi>a</mi> <mo> </mo> <mi>c</mi>
            </msqrt>
          </mrow>
          <mrow>
            <mn>2</mn> <mo> </mo> <mi>a</mi>
          </mrow>
        </mfrac>
      </math>
    </p>
  </body>
</html>
```

HTML5 — Canvas

Canvas

The `canvas` element provides scripts with a resolution-dependent bitmap canvas, which can be used for rendering graphs, game graphics, or other visual images on the fly.

In 2D or 3D (WebGL)

Beispiel 4.35 (HTML).

[38] http://www.w3.org/TR/MathML3/

```
<!DOCTYPE html>
<html lang="de">
  <head>
    <meta charset="utf-8">
    <title>HTML5 Canvas Beispiele</title>
    <script type="text/javascript" src="canvas.js"></script>
  </head>
  <body onload="draw();">
    <h1>HTML5 Canvas</h1>
    <canvas id="canvasId" width="150" height="130"></canvas>
  </body>
</html>
```

- HTML Canvas 2D Context[39]
- Canvas tutorial[40]
- WebGL[41]
- three.js[42]

HTML5 — Canvas

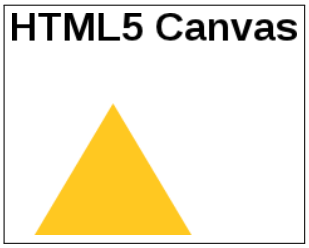
Beispiel 4.36 (JavaScript `canvas.js`).

```
function draw()
{
 const context = document.getElementById('canvasId')
                          .getContext("2d");

 const width = 125;  // Triangle Width
 const height = 105; // Triangle Height
 const padding = 20;

 // Draw a path
 context.beginPath();
 context.moveTo(padding + width/2, padding);         // Top Corner
 context.lineTo(padding + width, height + padding); // Bottom Right
 context.lineTo(padding, height + padding);          // Bottom Left
 context.closePath();

 // Fill the path
 context.fillStyle = "#ffc821";
 context.fill();
}
```

HTML5 — Canvas

[39] http://www.w3.org/TR/2dcontext/
[40] https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API/Tutorial
[41] https://developer.mozilla.org/en-US/docs/Web/WebGL
[42] http://threejs.org/examples/

Beispiel 4.37 (Ausgabe).



HTML 5 — Tabular Data

Bundesliga Tabelle

Platzierung	Mannschaft	Spiele	Siege	Unentschieden	Niederlagen	Tore	Gegentore	Tordifferenz	Punkte
1	Bayern München	29	23	4	2	76	13	63	73
2	Vfl Wolfsburg	28	18	6	4	62	30	32	60
3	Bayer 04 Leverkusen	29	15	9	5	56	31	25	54
4	Borussia Mönchengladbach	29	15	9	5	44	22	22	54
5	FC Schalke 04	28	11	8	9	37	31	6	41
6	FC Augsburg	28	12	3	13	34	36	-2	39
7	1899 Hoffenheim	29	10	7	12	43	47	-4	37
8	Borussia Dortmund	29	10	6	13	38	37	1	36
9	Eintracht Frankfurt	29	9	9	11	51	57	-6	36
10	Werder Bremen	28	9	8	11	43	57	-14	35
11	1. FSV Mainz 05	29	7	13	9	40	41	-1	34
12	1. FC Köln	29	8	10	11	29	35	-6	34
13	Hertha BSC	29	9	7	13	34	45	-11	34
14	SC Freiburg	29	6	11	12	29	39	-10	29
15	Hannover 96	29	7	8	14	32	49	-17	29
16	SC Paderborn 07	29	6	9	14	25	56	-31	27
17	VfB Stuttgart	28	6	8	14	31	51	-20	26
18	Hamburger SV	28	6	7	15	16	43	-27	25

HTML

```
<table>
  <caption>Bundesliga Tabelle</caption>
  <thead>
    <tr>
      <th>Platzierung</th>
      <th>Mannschaft</th>
      <th>Spiele</th>
      <th>Siege</th>
      <th>Unentschieden</th>
      <th>Niederlagen</th>
      <th>Tore</th>
      <th>Gegentore</th>
      <th>Tordifferenz</th>
      <th>Punkte</th>
    </tr>
  </thead>
  <tbody>
    <tr class="cl"> <td>1</td> <td>Bayern München</td>
      <td>29</td> <td>23</td> <td>4</td> <td>2</td> <td>76</td>
      <td>13</td> <td>63</td> <td>73</td>
    </tr>...
  </tbody>
</table>
```

CSS

```
body {
    font-family: Arial, sans-serif;
}
table {
    border: 1px solid black;
    border-collapse: collapse;
    text-align: left;
}
caption {
    font-size: 24px;
    margin-bottom: 24px;
}
th, td { padding: 5px 10px; }
td { border-top: 1px solid grey; }
thead {
    background-color: hsl(240, 50%, 20%);
    color: white;
    font-weight: bold;
    border-bottom: 1px solid white;
}
.cl { background-color: hsl(120, 60%, 55%); }
.clq { background-color: hsl(120, 60%, 70%); }
.el { background-color: hsl(120, 60%, 80%); }
.elq { background-color: hsl(120, 60%, 90%); }
.ab-rel { background-color: hsl(0, 60%, 80%); }
.ab { background-color: hsl(0, 60%, 70%); }
```

HTML5 — Tabular Data

Tables

- The `table` element represents data with more than one dimension, in the form of a table.

```
<h1>Today's Sudoku</h1>
<table>
 <colgroup><col><col><col>
 <colgroup><col><col><col>
 <colgroup><col><col><col>
 <tbody>
  <tr> <td> 1 <td>   <td> 3 <td> 6 <td>   <td> 4 <td> 7 <td>   <td> 9
  <tr> <td>   <td> 2 <td>   <td>   <td> 9 <td>   <td>   <td> 1 <td>
  <tr> <td> 7 <td>   <td>   <td>   <td>   <td>   <td>   <td>   <td> 6
 <tbody>
  <tr> <td> 2 <td>   <td> 4 <td>   <td> 3 <td>   <td> 9 <td>   <td> 8
  <tr> <td>   <td>   <td>   <td>   <td>   <td>   <td>   <td>   <td>
  <tr> <td> 5 <td>   <td>   <td> 9 <td>   <td> 7 <td>   <td>   <td> 1
 <tbody>
  <tr> <td> 6 <td>   <td>   <td>   <td> 5 <td>   <td>   <td>   <td> 2
  <tr> <td>   <td>   <td>   <td>   <td> 7 <td>   <td>   <td>   <td>
  <tr> <td> 9 <td>   <td>   <td> 8 <td>   <td> 2 <td>   <td>   <td> 5
</table>
```

HTML5 — Tabular Data

Today's Sudoku

1		3	6		4	7		9
	2			9			1	
7								6
2		4		3		9		8
5			9		7			1
6				5				2
				7				
9			8		2			5

HTML5 — Forms

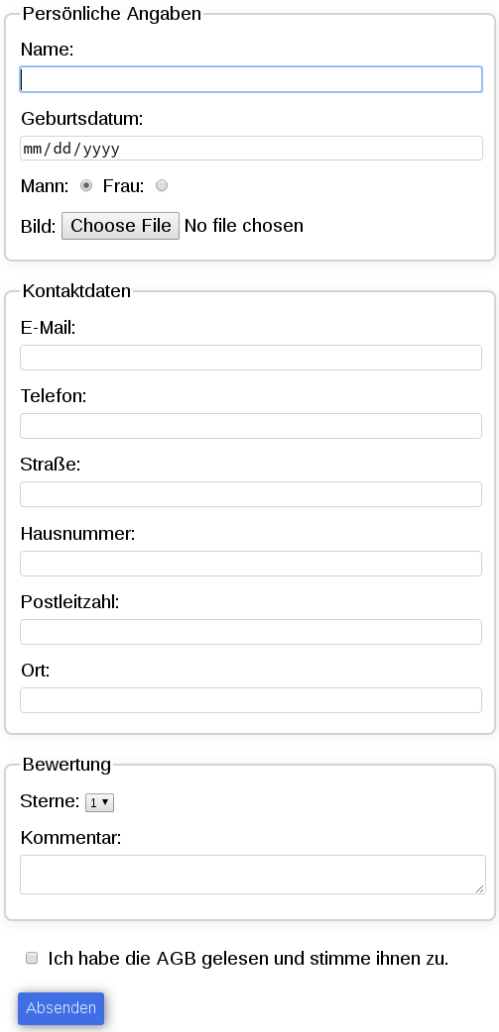
Forms

A form is a component of a Web page that has form controls, such as text fields, buttons, checkboxes, range controls, or color pickers. A user can interact with such a form, providing data that can then be sent to the server for further processing (...). No client-side scripting is needed in many cases, though an API is available so that scripts can augment the user experience or use forms for purposes other than submitting data to a server.

Viele neue Möglichkeiten

- Felder gruppieren mit `fieldset` und `legend`
- `label` und `input`
- Viele Feldtypen mit client-seitiger Überprüfung
- Attribute `required`, `autocomplete`, `autofocus`
- Verbesserte Nutzung auf mobilen Geräten

HTML 5 — Forms



HTML

```
<form accept-charset="utf-8">
  <fieldset>
    <legend>Persönliche Angaben</legend>
    <div class="field">
      <label for="name">Name:</label>
      <input id="name" type="text" autofocus>
    </div>
    <div class="field">
      <label for="geburtsdatum">Geburtsdatum:</label>
      <input id="geburtsdatum" type="date">
    </div>
    <div class="field inline">
      <label for="mann">Mann:</label>
      <input id="mann" name="geschlecht" value="mann" type="radio"
             checked="checked">
      <label for="frau">Frau:</label>
      <input id="frau" name="geschlecht" value="frau" type="radio">
    </div>
    <div class="field inline">
      <label for="bild">Bild:</label>
      <input id="bild" type="file" accept="image/*">
```

```
    </div>
  </fieldset>
```

HTML (Forts.)

```
  <fieldset><legend>Kontaktdaten</legend>
    <div class="field">
      <label for="mail">E-Mail:</label>
      <input id="mail" type="email">
    </div>...
    <div class="field">
      <label for="postleitzahl">Postleitzahl:</label>
      <input id="postleitzahl" type="number" min="10000" max="99999">
    </div>...
  </fieldset>
  <fieldset><legend>Bewertung</legend>
    <div class="field inline">
      <label for="note">Sterne:</label>
      <select id="note">
        <option value="1">1</option> <option value="2">2</option>
        <option value="3">3</option> <option value="4">4</option>
        <option value="5">5</option>
      </select>
    </div>...
  </fieldset>
  <div class="field inline">
    <input id="agb" type="checkbox">
    <label for="agb">Ich habe die AGB gelesen und stimme ihnen zu.</label>
  </div>
  <button type="submit">Absenden</button>
</form>
```

HTML5-Formulardaten und HTTP

Beispiel 4.38 (HTML5-Formular). `<!DOCTYPE html>`

```
<html>
  <head>
    <title>Form Demonstration</title>
  </head>
  <body>
    <form method="GET" action="/test1" enctype="application/x-www-form-urlencoded">
      <p><label>Name: <input name="name" type="text"></label>
      <p><label>Age: <input name="age" type="number"></label>
      <p><button>Submit</button>
    </form>
    <form method="POST" action="/test2" enctype="application/x-www-form-urlencoded">
      <p><label>Name: <input name="name" type="text"></label>
      <p><label>Age: <input name="age" type="number"></label>
      <p><button>Submit</button>
    </form>
    <form method="POST" action="/test3" enctype="multipart/form-data">
      <p><label>Name: <input name="name" type="text"></label>
      <p><label>Age: <input name="age" type="number"></label>
      <p><label>Profile picture: <input name="upload" type="file"></label>
      <p><button>Submit</button>
    </form>
```

```
  </body>
</html>
```

Beispiel 4.39 (HTTP-Request für `method="GET"`).

```
GET /test1?name=Einstein&age=139 HTTP/1.1
Host: localhost:3333
Referer: http://localhost:3333/
```

Beispiel 4.40 (HTTP-Request für `method="POST"` und `enctype="application/x-www-form-urlencoded"`).

```
POST /test2 HTTP
Host: localhost:3333
Content-Length: 21
Cache-Control: max-age=0
Origin: http://localhost:3333
Content-Type: application/x-www-form-urlencoded
Referer: http://localhost:3333/

name=Einstein&age=139
```

Beispiel 4.41 (HTTP-Request für `method="POST"` und `enctype="multipart/form-data"`).

```
POST /test3 HTTP/1.1
Host: localhost:3333
Content-Length: 44895
Origin: http://localhost:3333
Content-Type: multipart/form-data; boundary=----WebKitFormBoundaryf4A7P8NDKJg31AGK
Referer: http://localhost:3333/

------WebKitFormBoundaryf4A7P8NDKJg31AGK
Content-Disposition: form-data; name="name"

Einstein
------WebKitFormBoundaryf4A7P8NDKJg31AGK
Content-Disposition: form-data; name="age"

139
------WebKitFormBoundaryf4A7P8NDKJg31AGK
Content-Disposition: form-data; name="upload"; filename="test.gif"
Content-Type: image/gif

GIF87a...
...
...
------WebKitFormBoundaryf4A7P8NDKJg31AGK--
```

HTML5 — Fazit

- Komplett semantischer Ansatz
- Erweiterte Formulare
- Medien ohne Flash o. ä.
- Canvas-Grafik
- *Work in Progress*

4.8 Cascading Style Sheets

Cascading Style Sheets

W3C-Spezifikationen (Auswahl)

- Cascading Style Sheets home page[43]
- Cascading Style Sheets, level 2 revision 1[44] (2011)
- Selectors Level 3
- CSS Color Level 3
- Media Queries
- CSS Backgrounds and Borders Level 3
- CSS Conditional Rules Level 3
- CSS Image Values and Replaced Content Level 3
- CSS Multi-column Layout
- CSS Print Profile, CSS Speech
- CSS Text Decoration Module Level 3
- CSS Cascading and Inheritance Level 3
- CSS Fonts Level 3

Mozilla Developer Network

CSS reference[45]

Ziele

- Für HTML, XHTML, SVG, ...
- Trennung von Inhalt und Formatierung!
- Ersetzen veralteter/proprietärer/komplizierter Techniken

Liste aller W3C-Spezifikationen zu CSS[46]

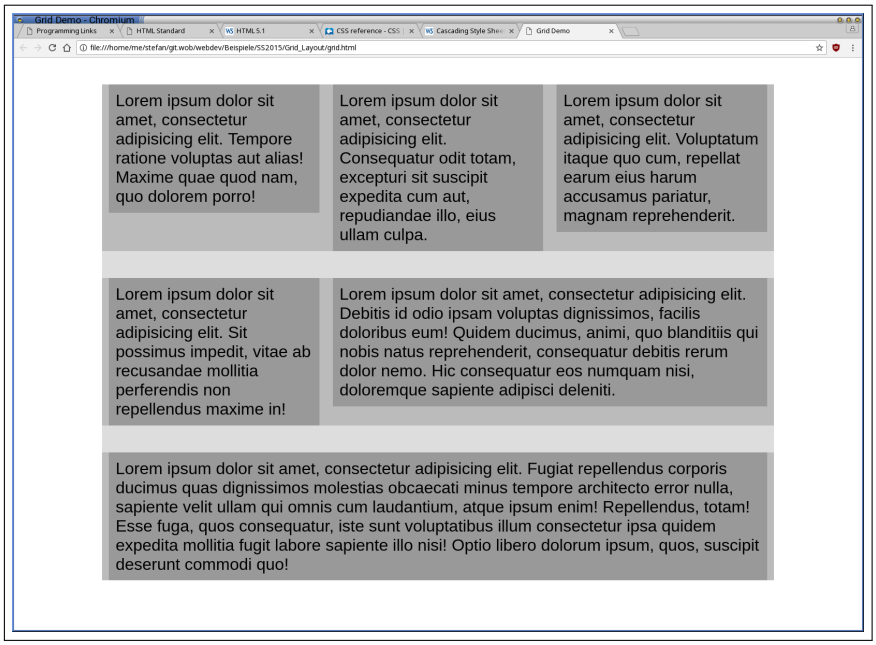
Buch (nicht perfekt, aber hilfreich, praktische Tipps):

Peter Müller. Flexible Boxes — Eine Einführung in moderne Websites. Galileo Computing 2013.

Beispiel: 2-dimensionale Anordnung mit Floats

Etwas veraltete Methode!

[43] https://www.w3.org/Style/CSS/
[44] http://www.w3.org/TR/CSS2/
[45] https://developer.mozilla.org/en-US/docs/Web/CSS/Reference
[46] http://www.w3.org/Style/CSS/current-work



HTML

```
<body>
  <div class="container">
    <div class="row">
      <div class="grid_1">Lorem ipsum...</div>
      <div class="grid_1">Lorem ipsum...</div>
      <div class="grid_1">Lorem ipsum...</div>
    </div>

    <div class="row">
      <div class="grid_1">Lorem ipsum...</div>
      <div class="grid_2">Lorem ipsum...</div>
    </div>

    <div class="row">
      <div class="grid_3">Lorem ipsum...</div>
    </div>
  </div>
</body>
```

CSS

```
* { box-sizing: border-box; }

.container { width: 80%; margin: auto; }

.row { margin: 4% 0; }

.grid_1, .grid_2, .grid_3 {
    display: block;
    float: left;
    margin-left: 1%;
    margin-right: 1%;
    padding: 1%;
```

```
}

.container .grid_1 { width: 31.333%; }
.container .grid_2 { width: 64.667%; }
.container .grid_3 { width: 98.0%; }

.row:after {
  content: "";
  display: table;
  clear: both;
}
```

Details siehe http://nicolasgallagher.com/micro-clearfix-hack/

Cascading Style Sheets

Styles können an verschiedenen Stellen definiert werden:

Beispiel 4.42 (Externes Style Sheet). `<head>`

```
  <link rel="stylesheet" href="main.css"/>
</head>
```

Beispiel 4.43 (Internes Style Sheet). `<style>`

```
  body { color: black; background: white; }
  em { font-style: normal; color: red; }
</style>
```

Beispiel 4.44 (`style`-Attribute). `<p>My sweat suit is`

```
  <span style="color: green; background: transparent">green</span>
  and my eyes are
  <span style="color: blue; background: transparent">blue</span>.</p>
```

Cascading Style Sheets

Eine *Style Rule* besteht aus zwei Teilen:

Selector

Auf welche Elemente ist die Regel anzuwenden?

Kriterien: u. a. Elementnamen, Attribute, Kontext der Elemente, benutzerdefinierte *Klassen*:

```
h1 { ... }
input[type="text"] { ... }
h1 em { ... }
h2:first-child { ... }
h1.appendix { ... }
```

Declarations

Properties (Name-Value-Paare) definieren Formatierungseigenschaften.

Selectors

Beispiel 4.45. `h1 { font-family: sans-serif; }`
`h2 { font-family: sans-serif; }`
`h3 { font-family: sans-serif; }`

Beispiel 4.46 (Group of Selectors). `h1, h2, h3 { font-family: sans-serif; }`

Type Selectors / Universal Selectors

Type selectors match elements by node name.

Beispiel 4.47.
```
span {
  color: green;
}
```

Universal selectors match elements of any type.

Beispiel 4.48.
```
*.warning {
  color: red;
}
```

Beispiel 4.49 (HTML). `The <span>usual span</span> is`
`<span class="warning">not</span> a`
`<strong class="warning">spam</strong>.`

Class Selectors

CSS class selectors match an element based on the contents of the element's class attribute.

Beispiel 4.50 (CSS).
```
span.classy {
  background-color: DodgerBlue;
}
```

Beispiel 4.51 (HTML).
```
<span class="classy">Here's a span.</span>
<span>Here's another.</span>
<span class="foo classy">And a third.</span>
<div class="classy">And a div.</div>
```

ID Selectors

CSS ID selectors match an element based on the contents of that element's ID attribute.

Beispiel 4.52 (CSS).
```
span#funny {
  background-color: DodgerBlue;
}
```

Beispiel 4.53 (HTML).
```
<span id="funny">Here's a span.</span>
<span>Here's another.</span>
```

Attribute Selectors

```
/* All spans with a "lang" attribute are bold */
span[lang] { font-weight:bold; }

/* All spans in Portuguese are green */
span[lang="pt"] { color:green; }

/* All spans in US English are blue  */
span[lang~="en-us"] { color: blue; }

/* All internal links (starting with #) have a gold background */
a[href^="#"] { background-color:gold; }

/* All links to urls ending in ".cn" are red */
a[href$=".cn"] { color: red; }

/* All links with "example" in the url have a grey background */
a[href*="example"] { background-color: #CCCCCC; }
```

Child Selectors

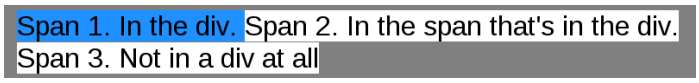
. . . direct children of elements . . .

Beispiel 4.54 (CSS). `span { background-color: white; }`

```
div > span {
  background-color: DodgerBlue;
}
```

Beispiel 4.55 (HTML). `<div>`

```
  <span>Span 1. In the div.
    <span>Span 2. In the span that's in the div.</span>
  </span>
</div>
<span>Span 3. Not in a div at all</span>
```



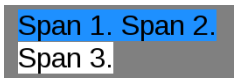
Descendant Selectors

Beispiel 4.56 (CSS). `span { background-color: white; }`

```
div span { background-color: DodgerBlue; }
```

Beispiel 4.57 (HTML). `<div>`

```
  <span>Span 1.
    <span>Span 2.</span>
  </span>
</div>
<span>Span 3.</span>
```



Adjacent and General Sibling Selectors

Beispiel 4.58 (CSS). `h4 + p { text-decoration: underline; }`
`h4 ~ p { color: blue; }`

Beispiel 4.59 (HTML).
```
<h3>Families in Trees</h3>
  <p>One</p>
  <h4>Siblings and Such</h4>
  <p>Two</p>
  <p>Three</p>
  <p>Four</p>
```

Families in Trees One **Siblings and Such** Two Three Four

`+` und `~` „schauen nach vorne“ im Dokument. Für den „davor“-Fall gibt es keinen äquivalenten Selektor.

Pseudo-Elements

Pseudo-Elemente ermöglichen Styling, das sonst nur mit zusätzlichem Markup möglich wäre.

`::first-letter`, `::first-line`, `::after`, `::before`

Beispiel 4.60. `/* make the first letter of every paragraph red and big */`

```
p::first-letter {
  color: red;
  font-size: 130%;
}

/* Change the letters of the first line of
   every paragraph to uppercase. */

p::first-line { text-transform: uppercase; }
```

LOREM IPSUM DOLOR SIT AMET, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore.

DUIS AUTE IRURE DOLOR IN reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur.

`::before` creates a pseudo-element that is the first child of the element matched. Often used to add cosmetic content to an element, by using the `content` property.

Beispiel 4.61 (CSS). `q::before {`
```
  content: "«";
  color: blue;
}
```

```
q::after  {
  content: "»";
  color: red;
}
```

Beispiel 4.62 (HTML). `<q>Some quotes</q>, he said,`
`<q>are better than none</q>.`

«Some quotes», he said, «are better than none».

Pseudo-Classes

A CSS pseudo-class is a keyword added to selectors that specifies a *special state* of the element to be selected.

Pseudo-Classes for Links and User Action

`:link` → `:hover` → `:active` → `:visited`

Beispiel 4.63. `a:link    { color:#0271fb; }`
`a:visited { color:#bd02fb; }`
`a:hover   { color:#000000; }`
`a:active  { color:#fb0e02; }`

Pseudo-Klassen ermöglichen Styling, als ob man manuell (und ggf. temporär) Klassen zu Elementen hinzugefügt hätte.

The `:target` pseudo-class represents the unique element, if any, with an `id` matching the fragment identifier of the URI of the document.

Beispiel 4.64. `:target { outline: solid black; }`
`...`
`<h1 id="section2">Basic Notions</h1>`

`http://example.com/folder/document.html`

Basic Notions

`http://example.com/folder/document.html#section2`

Basic Notions

The `:target` pseudo-class is useful to switch on/off some invisible elements. (Menus, . . .)

Structural Pseudo-Classes

. . . selection based on extra information that lies in the document tree . . .

Beispiel 4.65 (HTML). `<div><span>This span is limed!</span>`
`  <span>This span is not. :(</span>`
`</div>`

Beispiel 4.66 (CSS). `span:first-child { background-color: lime; }`

Beispiele 4.67. `span:nth-child(2n+1) { background-color: lime; }`
`p:nth-of-type(odd) { text-align: left; }`

`:nth-last-child()`, `:last-child`, `:first-of-type`, `:last-of-type`, `:only-child`, `:only-of-type`, `:empty`

Spezifität der Selektoren

Calculating a selector's specificity

A selector's specificity is calculated as follows:

- count 1 if the declaration is from is a 'style' attribute rather than a rule with a selector, 0 otherwise (= a)
- count the number of ID attributes in the selector (= b)
- count the number of other attributes, classes, and pseudo-classes in the selector (= c)
- count the number of element names and pseudo-elements in the selector (= d)

Concatenating the four numbers a-b-c-d gives the specificity.

Beispiel 4.68.
```
/* a=0 b=1 c=0 d=1 */
section#intro { color: #004; }

/* a=0 b=1 c=0 d=0 */
#intro { color: #00F; }

/* a=0 b=0 c=1 d=1 */
p.good { color: #040; }

/* a=0 b=0 c=1 d=0 */
.good { color: #0F0; }

/* a=0 b=0 c=0 d=2 */
body p { color: #444; }

/* a=0 b=0 c=0 d=1 */
p { color: #AAA; }

/* a=0 b=0 c=0 d=0 */
* { color: #000; }
```

Siehe: The cascade[47]

Siehe auch: http://specificity.keegan.st/

[47] http://www.w3.org/TR/CSS21/cascade.html#cascade

Cascading Style Sheets

Cascading — Zusammenspiel mehrerer Style Sheets

- Author Style Sheet
- User Style Sheet
- User Agent Style Sheet

Property-Werte

Werte aller Properties sind für jedes Element definiert:

1. explizit in der Cascade definiert,
2. geerbt vom Vaterelement, oder
3. initialer (Default) Wert

Relative / absolute / berechnete / tatsächliche Werte

Werte (Auswahl)

Absolute Längeneinheiten

- **in**: inches — 1in is equal to 2.54cm.
- **cm**: centimeters
- **mm**: millimeters
- **pt**: points — the points used by CSS are equal to 1/72nd of 1in.
- **pc**: picas — 1pc is equal to 12pt.
- **px**: pixel units — 1px is equal to 0.75pt.

Relative Längeneinheiten

- **em**: the 'font-size' of the relevant font
- **ex**: the 'x-height' of the relevant font
- Percentages: `120%`

URLs

```
body {
  background: url("http://www.example.com/pinkish.png");
}

li {
  list-style: url("http://www.example.com/redball.png") disc;
}
```

Colors

- RGB, RGBA

  ```
  em { color: #f00; }
  em { color: #ff0000; }
  em { color: rgb(255,0,0); }
  em { color: rgb(100%, 0%, 0%); }

  em { color: rgba(255,0,0,1); }
  em { color: rgba(100%, 50%, 0%, 0.1); }
  ```

- hue-saturation-lightness (HSL): `hsl(120, 100%, 25%)`
- HSLA ...
- Extended color keywords: `black`, `blue`, `blueviolet`, `burlywood`, ...

Properties: Fonts

Beispiel 4.69. `body { font-family: "Times New Roman", Times, serif; }`

```
h1, h2, h3 { font-style: italic; }
h1 em { font-style: normal; }

h3 { font-variant: small-caps; }

p { font-weight: normal; }
h1 { font-weight: bold; }

p { font-size: 16px; }
blockquote { font-size: larger; }
em { font-size: 150%; }
em { font-size: 1.5em; }

p { font: normal small-caps 120%/120% fantasy; }
```

Quelle für (Web-)Fonts

https://fonts.google.com/

Properties: Fonts — `font-family`

Beispiel 4.70. `<body>`
```
  <p style="font-family: serif">Serif: abcdeghijk</p>
  <p style="font-family: sans-serif">Sans-Serif: abcdeghijk</p>
  <p style="font-family: monospace">Monospace: abcdeghijk</p>
</body>
```

Serif: abcdeghijk

Sans-Serif: abcdeghijk

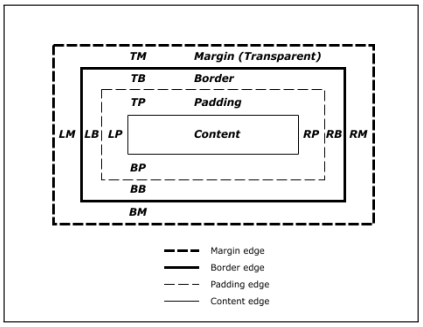
`Monospace: abcdeghijk`

Auch: `font-family: cursive` und `font-family: fantasy`

Properties: Text

Beispiel 4.71. `p { text-indent: 3em; }`
```
div.important { text-align: center; }
a:visited,a:link { text-decoration: underline; }

h1 { text-transform: uppercase; }

pre        { white-space: pre; }
p          { white-space: normal; }
td[nowrap] { white-space: nowrap; }
```

CSS Box Model



`box-sizing: content-box` ⇒ box size = content size (default)

`box-sizing: border-box` ⇒ box size = border+padding+content size

Properties: Boxes

Beispiel 4.72 (Width and Height). `table {`
```
  min-width: 75%;
```

```
  heigth: 50%;
}

body { max-width: 40em; }

img {
  max-width: 100%;
  height: auto;
}
```

Beispiel 4.73 (Margins). `body { margin: 2em; }`
`body { margin: 1em 2em; }`
`body { margin: 1em 2em 3em; }`

```
body {
  margin-top: 1em;
  margin-right: 2em;
  margin-bottom: 3em;
  margin-left: 2em;
}
```

Beispiel 4.74 (Padding). `h1 {`
```
  background: white;
  padding: 1em 2em;
}
```

Properties: Borders

Beispiel 4.75 (Borders). `h1 { border-width: thin; }`
`h1 { border-width: thin thick; }`

```
h2 {
    border-radius: 10px;
    border-style: solid;
    border-width: 5px 10px 15px 20px;
    border-color: red yellow green blue;
    box-shadow: 12px 6px 10px #ddd;
}
```

Visual Formatting Model

Elemente generieren null, eine oder mehrere *Boxen*.

Das Box-Layout hängt ab von

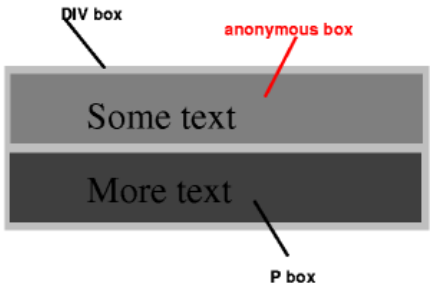
- Box-Dimensionen und -typen,
- Positionierungsschema (normal flow, float, absolute),
- der Struktur des HTML-Baums, Containing Blocks,
- externen Einflüssen (Viewport Size, Größe von Bildern, ...).

Formatierungsrichtungen

- *Block*-Formatierung: vertikal
- *Inline*-Formatierung: horizontal
- Auch: Floats und absolute Positionierung

Beispiel 4.76 (Display Properties).
```
div { display: block; }
p   { display: block; }
em  { display: inline; }
li  { display: list-item; }

img { display: none; }
```

Beispiel 4.77 (Anonymous block and inline boxes).
```
<DIV>
  Some text
  <P>More text
</DIV>
```



Sonstiges

- Box offsets: `top`, `right`, `bottom` `left`
- Layered presentation: the `z-index` property
- Visual effects: overflow and clipping
- JavaScript DOM-API
- ...

Weiterlesen: https://www.w3.org/TR/CSS2/visuren.html#q9.0

Generated Content, Automatic Numbering, Lists

Beispiel 4.78 (Content).
```
p.note:before { content: "Note: "; }
p.note        { border: solid green; }
```

Beispiel 4.79 (Content).
```
body:after {
  content: "The End";
  display: block;
  margin-top: 2em;
  text-align: center;
}
```

Beispiel 4.80 (Quotes).
```
/* Specify pairs of quotes for two levels in two languages */
q:lang(en) { quotes: '"' '"' "'" "'"; }
q:lang(no) { quotes: "«" "»" '"' '"'; }

/* Insert quotes before and after Q element content */
q:before { content: open-quote; }
q:after  { content: close-quote; }
```

Beispiel 4.81 (Automatic counters and numbering).
```
/* Create a chapter counter scope */
body {
  counter-reset: chapter;
}
h1:before {
  content: "Chapter " counter(chapter) ". ";
  counter-increment: chapter;  /* Add 1 to chapter */
}
h1 {
  counter-reset: section;      /* Set section to 0 */
}
h2:before {
  content: counter(chapter) "." counter(section) " ";
  counter-increment: section;
}
```

Beispiel 4.82 (Lists).
```
ul { list-style: upper-roman inside; }
ul > li > ul { list-style: circle outside; }

ol.alpha > li  { list-style: lower-alpha; }
ul li   { list-style: disc; }
```

Properties: Tables

Beispiel 4.83.
```
table   { border-collapse: collapse; }
tr#row1 { border: 3px solid blue; }
tr#row2 { border: 1px solid black; }
tr#row3 { border: 1px solid black; }

caption { caption-side: bottom;
          width: auto;
```

```
          text-align: left }

foo { display : table; }
bar { display : table-caption; }
```

Beispiel 4.84 (default style sheet). `table    { display: table; }`

```
tr       { display: table-row; }
thead    { display: table-header-group; }
tbody    { display: table-row-group; }
tfoot    { display: table-footer-group; }
col      { display: table-column; }
colgroup { display: table-column-group; }
td, th   { display: table-cell; }
caption  { display: table-caption; }
```

Media Queries

Die *Präsentation* des Dokuments (HTML5) wird an verschiedene Geräte angepasst, ohne das Dokument selbst zu modifizieren.

Beispiel 4.85 (CSS). `@media screen and (min-width: 1024px) {`

```
  html {
    background-color: #c0c0c0;
    background-image: url("big-bg.jpg");
    background-repeat: no-repeat;
  }
} /* end @media */
```

Beispiel 4.86 (CSS). `@media (min-width: 700px)`

```
       and (orientation: landscape) { ... }
@media all and (color) { ... }
```

Korrektur für mobile Browser

Auf Smartphones mit hoher Pixeldichte (ab 200 dpi) werden vom mobilen Browser oft zusätzliche Skalierungen vorgenommen.

Damit die Media Queries wie gewünscht funktionieren, ist dieses `meta`-Element in das `head`-Element im HTML einzufügen:

```
<meta name="viewport"
      content="width=device-width, initial-scale=1">
```

Siehe hierzu: Using the viewport meta tag to control layout on mobile browsers[48]

Layout — neuere CSS-Module

CSS Flexible Box Layout

[48] https://developer.mozilla.org/en-US/docs/Mozilla/Mobile/Viewport_meta_tag

- W3C Candidate Recommendation, 19 October 2017[49]
- Platz in einem Container dynamisch verteilen (1-dimensional, Block oder Inline)

CSS Grid Layout

- W3C Candidate Recommendation, 14 December 2017[50]
- Grid-basiertes Layout „ohne Tricks“ (2-dimensional)

Weitere Infos zu CSS Flexible Box Layout, Grid Layout und Multi-Column Layout

- A Complete Guide to Flexbox[51]
- MDN — Using CSS Flexible Boxes[52]
- A Complete Guide to Grid[53]
- MDN — CSS Grid Layout[54]
- MDN —Using CSS multi-column layouts[55]

CSS Flexible Box Layout

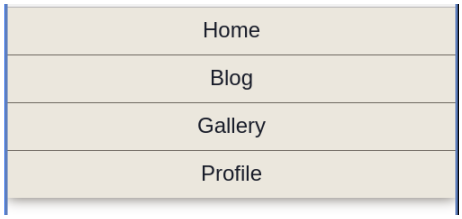
Beispiel 4.87 (HTML). `<!DOCTYPE html>`

```
<html>
<head>
  <meta charset="utf-8">
  <title>Flexbox</title>
  <link rel="stylesheet" href="layout.css">
  <link rel="stylesheet" href="style.css">
</head>
<body>
<nav>
  <a href="#">Home</a>
  <a href="#">Blog</a>
  <a href="#">Gallery</a>
  <a href="#">Profile</a>
</nav>
</body>
```

[49] http://www.w3.org/TR/css-flexbox-1/
[50] https://www.w3.org/TR/css-grid-1/
[51] https://css-tricks.com/snippets/css/a-guide-to-flexbox/
[52] https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Flexible_Box_Layout/Using_CSS_flexible_boxes
[53] https://css-tricks.com/snippets/css/complete-guide-grid/
[54] https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Grid_Layout
[55] https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Columns/Using_multi-column_layouts

Beispiel 4.88 (`layout.css`). `nav {`

```
  display: flex;
}

nav a:nth-child(4) {
  margin-left: auto;
}

@media all and (max-width: 500px) {
  nav {
    flex-flow: column;
  }
}
```

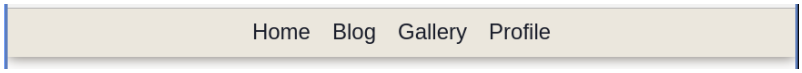


Beispiel 4.89 (`layout1.css`). `nav {`

```
  display: flex;
  justify-content: space-around;
}
```

Beispiel 4.90 (`layout2.css`). `nav {`

```
  display: flex;
  justify-content: center;
}
```



CSS Grid Layout

Beispiel 4.91 (HTML). `<!DOCTYPE html>`

```
<html>
<head>
  <meta charset="utf-8">
  <title>Grid Layout</title>
  <link rel="stylesheet" href="layout.css">
  <link rel="stylesheet" href="style.css">
</head>
<body>
  <header>Header</header>
  <aside>Sidebar</aside>
  <main>Main content</main>
  <footer>Footer</footer>
```

```
</body>
</html>
```

Beispiel 4.92 (`layout.css`).
```
body {
  display: grid;
  height: 100vh;
  grid-template-rows: 1fr 5fr 2fr;
  grid-template-columns: 1fr 3fr;
  grid-template-areas:
    "header header"
    "aside main"
    "footer footer";
}

header {
  grid-row: 1; grid-column: 1 / 3;
}
aside { grid-area: aside; }
main { grid-area: main; }
footer {  grid-area: footer; }
```

Header	
Sidebar	Main content
Footer	

Weiteres Beispiel[56]

CSS Multi-column Layout

Beispiel 4.93 (HTML).
```
<!doctype html>
<html>
  <head>
    <title>Multi-Column Layout</title>
    <link rel="stylesheet" href="layout.css">
    <link rel="stylesheet" href="style.css">
  </head>
  <body>
    <h2>Lorem Ipsum</h2>
    <p>Lorem ipsum dolor sit amet, consectetur...</p>
    <p>Ex laudantium vel, minus, voluptate...</p>
    <q>Lorem ipsum dolor sit amet.</q>
    <p>Assumenda iure labore, non deleniti alias...</p>
    ...
```

[56] https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Grid_Layout

```
    <h2>Ullam odio</h2>
    ...
  </body>
</html>
```

Beispiel 4.94 (`layout.css`).

```
body {
  max-width: 1200px;
  margin: 0 auto 5em;
  columns: 3 200px;
  column-gap: 3em;
  column-rule: 1px solid #aaa;
}

h2 {
  column-span: all;
  break-inside: avoid;
  break-after: avoid-page;
}
```

Lorem Ipsum

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Ullam odio pariatur, neque aspernatur eveniet numquam ipsum ut mollitia perspiciatis quas autem cupiditate molestias doloribus, sint quisquam, dignissimos id doloremque nesciunt placeat eius sunt molestiae tempora. Tenetur quo autem voluptatem sint, expedita distinctio quas dicta aliquam? Praesentium eius consectetur ut tempora laborum, sunt natus voluptate nisi laudantium commodi atque velit officia minus asperiores maxime cumque iusto delectus, nostrum reiciendis quia voluptas dolores quas, impedit, odio error. Nihil, laboriosam fugiat. Eius fugit, soluta amet aliquid nisi suscipit. Reiciendis ducimus totam ex ipsa placeat at dignissimos, voluptatum quos quod ratione culpa, magnam, expedita eligendi, porro! Consequuntur inventore dicta ratione, deleniti labore nobis! Consequatur corrupti totam ex expedita mollitia, odio autem aspernatur harum labore, atque accusamus, perferendis, magni dolores officiis laborum vel!

Ex laudantium vel, minus, voluptate necessitatibus obcaecati commodi repudiandae nobis est, eaque assumenda! Omnis voluptatem eius, ad, velit minima impedit mollitia sint expedita possimus veritatis soluta. Ad odit deleniti expedita molestiae alias voluptatibus doloremque temporibus accusantium minus enim voluptate ratione et, dicta laborum magni itaque sequi exercitationem qui velit porro corrupti tempora cupiditate blanditiis. Porro quos ipsum fuga magni est doloremque odio, cumque aut suscipit velit placeat, ab ullam dolorum, vel sequi odit culpa fugit neque molestiae doloribus! Nesciunt iusto, aut odio quo temporibus, ut consequatur facere quidem repellendus incidunt earum. Ducimus eaque sed, debitis nam ab libero dolor labore necessitatibus laudantium.

"Lorem ipsum dolor sit amet."

Assumenda iure labore, non deleniti alias officiis inventore, harum praesentium cumque atque vel reprehenderit minima commodi, dolorum similique. Cumque explicabo amet rerum quasi voluptatem nulla quo ducimus aspernatur laboriosam. Est at, asperiores voluptas dolores praesentium vero pariatur deleniti minima quam harum ipsum voluptatem laudantium beatae iure, accusamus nam officiis, nostrum! Dolor in expedita fugiat rerum minus, voluptatibus labore esse nulla dignissimos repellendus pariatur libero error possimus a consequatur est, veniam, accusantium id, recusandae enim alias facilis. Quidem sit laborum provident illum earum corporis quisquam suscipit vel modi eos quas blanditiis ea, esse commodi itaque repudiandae nisi debitis aliquam voluptatibus deleniti consequuntur neque, quod.

Neque molestiae vel sapiente praesentium dolor eveniet quod, quasi cumque nulla, unde laudantium optio earum ducimus, dicta nemo similique non. Rem quia facere exercitationem. Repellat optio inventore sint distinctio quae nesciunt hic ea voluptates consectetur, eaque voluptatum ipsum libero molestiae quaerat vitae reprehenderit nulla eveniet ratione qui doloremque. Nam odit minima sint sunt corporis consequatur necessitatibus quod illo iure quasi! Modi distinctio, rerum accusantium ex dolorum blanditiis facilis vitae eaque, ducimus fuga similique facere cupiditate repudiandae reprehenderit excepturi explicabo iste nihil. Deserunt, quis, modi.

Explicabo aliquam atque voluptates, architecto quas doloribus, temporibus consequatur sit quo quos reprehenderit quasi minima dolores, id. Esse molestiae sint quis necessitatibus quas, pariatur nobis cumque officia unde magnam fugit quos, iste, vero suscipit tenetur. Eaque est suscipit provident iste voluptatum doloribus quos ex dicta in, nam ab aperiam nisi nesciunt unde fugiat saepe assumenda! Maiores quisquam cupiditate ipsam consequuntur voluptatum, molestias sit, vitae dolores hic itaque facilis cumque.

Ullam odio

Ullam odio pariatur, neque aspernatur eveniet numquam ipsum ut mollitia perspiciatis quas autem

porro corrupti tempora cupiditate blanditiis. Porro quos ipsum fuga magni est doloremque odio,

Neque molestiae vel sapiente praesentium dolor eveniet quod, quasi cumque nulla, unde laudantium

CSS Fazit

- Präsentation von Dokumenten (HTML, SVG, . . .) beschreiben;
- außerhalb des Dokuments,
- auch geräteabhängig (Media Queries);
- viele Möglichkeiten, viele (neue) Module;

- ständig in Entwicklung

Zusatzeffekt

HTML-Dokumente sind viel einfacher und übersichtlicher als früher!

4.9 JavaScript

JavaScript

(Wikipedia)

- Skriptsprache
- Schwache, dynamische Typisierung
- Multiparadigmatisch, insbesondere
  - objektorientiert (mittels Prototypen)
  - imperativ
  - funktional (First-Class-Functions)
- Clientseitige Skripten können
  - mit dem Benutzer interagieren,
  - den Browser kontrollieren,
  - asynchron kommunizieren,
  - den Inhalt des angezeigten Dokuments verändern.
- Wird auch serverseitig eingesetzt (Node.js).
- JavaScript hat eine wesentlich andere Semantik als Java!

Quelle: https://de.wikipedia.org/wiki/JavaScript

JavaScript richtig lernen

Bücher

- Eloquent JavaScript, 3. edition[57]. Von Marijn Haverbeke. Licensed under a Creative Commons attribution-noncommercial license[58].
- Speaking JS[59]. Von Axel Rauschmayer.

[57] http://eloquentjavascript.net/
[58] http://creativecommons.org/licenses/by-nc/3.0/
[59] http://speakingjs.com/es5/index.html

- Exploring ES6[60]. Von Axel Rauschmayer.
- Übersicht der neuen Features in ES2015: https://babeljs.io/learn-es2015/

Links (MDN)

- JavaScript Guide[61]
- A re-introduction to JavaScript (JS Tutorial)[62]
- Zum Nachschlagen: JavaScript reference[63]

JavaScript-Versionen

- ECMAScript 5 und früher
- ECMAScript 6 = ES2015
- *ES2015+ hier in Vorlesung und Übung!*

JavaScript am Beispiel

JavaScript-Konsolen

- Developer Tools im Browser
- CLI von Node.js[64]

```
$ node
> 'I' + ' am'
'I am'
> 21 * 2
42
```

[60] http://exploringjs.com/es6/
[61] https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide
[62] https://developer.mozilla.org/en-US/docs/Web/JavaScript/A_re-introduction_to_JavaScript
[63] https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference
[64] http://nodejs.org/

Variablen und primitive Typen

`const`

- Nur lesender Zugriff.
- Zuweisung muss direkt bei der Deklaration erfolgen.
- Innere Werte, etwa bei Arrays oder Objekten, können verändert werden.

Beispiel 4.95.
```
const a = 42;
a = 43;     // Error; Assignment to constant variable.
const r;    // Error; Missing initializer.
```

Beispiel 4.96.
```
const p = {x: 1};
p.x = 3;
p.y = 2;
console.log(p);     // { x: 3, y: 2 }
```

`let`

- Wert kann nachträglich verändert werden.
- Empfehlung: Verwende überall `const`, außer wenn der Wert wirklich variabel sein soll.

Beispiel 4.97.
```
let isFinished = false;
// do stuff
isFinished = true;
```

Number

- Keine Unterscheidung zwischen ganzen Zahlen und Kommazahlen.
- Interne Repräsentation: *double* (IEEE 754).
- Ganze Zahlen können nur im Bereich $-(2^{53}-1)$ bis $2^{53}-1$ sicher dargestellt werden.
- Spezieller Wert `NaN`, „keine Zahl“.
- `-Infinity` und `Infinity`, etwa bei Division durch 0.
- Mehr dazu: Speaking JS[65].

Beispiel 4.98.
```
const n0 = 13.64;
const n1 = Number.MAX_SAFE_INTEGER + 1;
const n2 = Number.MAX_SAFE_INTEGER + 2;
console.log(n1 === n2);     // true!
console.log(NaN === NaN);   // false!
```

[65] http://speakingjs.com/es5/ch11.html

String

- Es gibt keinen gesonderten Typ für einzelne Zeichen.
- Umschlossen von `'...'` (single quotes) oder `"..."` (double quotes).
- Spezielle Zeichen, wie newline, müssen mit einem vorangestellten Backslash escaped werden.
- Mehr dazu: Speaking JS[66].

Beispiel 4.99.
```
const name = 'Clara Fall';
console.log(name);          // Clara Fall

const multiline = 'First\nSecond';
console.log(multiline);     // First
                            // Second
```

Template Strings

- Umschlossen mit `` `...` `` (Backticks).
- Text mit Platzhaltern der Form `${}`.
- Darin kann ein beliebiger JavaScript Ausdruck stehen.
- Zeilenumbrüche werden berücksichtigt.
- Mehr dazu: Exploring ES6[67].

Beispiel 4.100.
```
const name     = 'Clara Fall';
const greeting = `Hallo, ${name}! 2 * 21 = ${2 * 21}`;
console.log(greeting); // Hallo, Clara Fall! 2 * 21 = 42
```

`undefined`

- Standardwert für
  - Variablen, denen noch nicht explizit ein Wert zugewiesen wurde
  - Fehlende Argumente bei Funktionen
  - Fehlende Properties von Objekten
- Bedeutung: Kein Wert.

[66] http://speakingjs.com/es5/ch12.html
[67] http://exploringjs.com/es6/ch_template-literals.html

Beispiel 4.101.
```
let solution;
let p = { x: 1, y: 2 };
console.log(solution);        // undefined
console.log(p.z);             // undefined
console.log(solutions);       // ReferenceError

console.log(typeof solution); // undefined
```

`null`

- Wird anstelle von `undefined` eingesetzt, wenn ein Objekt erwartet wurde.
- Bedeutung: Kein Objekt.
- Leider liefert aber `typeof null` den Wert *object*. Dies ist ein Bug, der nicht mehr korrigiert werden kann, da existierende Programme unbrauchbar gemacht werden könnten.

Beispiel 4.102.
```
const futile = null;
console.log(futile);        // null

console.log(typeof futile); // object
```

Symbol

- Liefert einen einzigartigen Wert.
- Optionale textuelle Beschreibung.
- Kann als Schlüssel in Objekten verwendet werden.
- Mehr dazu: Exploring ES6[68].

Beispiel 4.103.
```
const unique = Symbol('really unique');
console.log(unique); // Symbol(really unique)

const uniq2 = Symbol('really unique');
console.log(uniq2);  // Symbol(really unique)
unique === uniq2;    // false

const COLOR_RED    = Symbol('Red');
const COLOR_ORANGE = Symbol('Orange');
// ...
```

Boolean

- Wert ist `true`/`false`.

[68] http://exploringjs.com/es6/ch_symbols.html

- Auch andere Typen können als Wahrheitswert interpretiert werden:
  - `null`, `undefined`, 0, `NaN` und `''` sind *falsy* Werte,
  - alle anderen sind *truthy*.
- Damit sind insbesondere alle Objekte wahr, also auch ein leeres Objekt `{}` und ein leeres Array [].
- Der String `'false'` wird ebenso als wahr interpretiert.
- Mehr dazu: Speaking JS[69].

Ausdrücke

Boolesche Operatoren

- Kombinieren zwei boolesche Werte und liefern einen booleschen Wert.
- Können auch für andere Typen genutzt werden. Die Zahl 0 etwa wird als `false` interpretiert.
- `&&` und || sind short-circuited: Sobald das Ergebnis feststeht, werden die nachfolgenden Operanden nicht mehr ausgewertet.
- Der Wert, der das Gesamtergebnis festlegt, wird zurückgegeben.

Beispiel 4.104.
```
const and = true && false; // false
const or  = false || 42;   // 42
const not = !true;         // false
```

Mathematische Operatoren

- Kombinieren zwei Zahlen und liefern eine Zahl.
- Weitere Funktionen unter dem `Math`-Objekt.
- Mehr dazu: Speaking JS[70] und Exploring ES6[71].

Beispiel 4.105.
```
const addition       = 1 + 2;        // 3
const subtraction    = 3 - 1;        // 2
const multiplication = 1 * 6;        // 6
const division       = 9 / 2;        // 4.5
const mod            = 4 % 2;        // 0
const exponentiation = 2 ** 10;      // 1024
const absoluteValue  = Math.abs(-1); // 1
const pi             = Math.PI;      // 3.141592653589793
console.log(Math.sin(pi / 2));       // 1
```

[69] http://speakingjs.com/es5/ch10.html
[70] http://speakingjs.com/es5/ch21.html
[71] http://exploringjs.com/es6/ch_numbers.html#_math

Vergleichsoperatoren

- Kombinieren zwei Werte zu einem booleschen Wert.

Beispiel 4.106. `const lessThan           = 1 < 2;  // true`

```
const lessThanOrEqual    = 2 <= 4; // true
const greaterThanOrEqual = 2 >= 6; // false
const greater            = 1 > 0;  // true
```

Gleichheitsoperator

- Besteht aus **drei** Gleichheitszeichen.
- Die Version mit zwei Gleichheitszeichen führt erst Typumwandlungen durch und sollte daher nicht verwendet werden. `'1'` und `true` werden etwa als gleich angesehen, nicht aber 2 und `true`.
- `===` und `!==` können für beliebige Werte benutzt werden.
- Bei Booleans, Numbers und Strings wird der Wert verglichen.
- Objekte und Symbole haben alle eine eigene Identität.

Beispiel 4.107. `const n1 = 2, n2 = 2;`

```
const equalNumber    = n1 === n2; // true
const notEqualNumber = n1 !== n2; // false
const equalObject    = {} === {}; // false
```

String-Operationen

- Werden auf Strings angewendet und liefern neue Strings.
- + Operator zum Konkatenieren.
- `[<index>]` zum Zugriff auf einzelne Zeichen.
- `length` Property zum Abfragen der Länge.
- Strings besitzen außerdem viele nützliche Methoden.
- Mehr dazu: Speaking JS[72] und Exploring ES6[73].

Beispiel 4.108. `const concatenate = 'Con' + 'cat';          // Concat`

```
const firstChar   = 'A long string...'[0];  // A
const length      = 'Count me!'.length;     // 9
const upper       = 'lower'.toUpperCase();  // LOWER
'a' + 'bc' === 'ab' + 'c';                  // true
```

[72] http://speakingjs.com/es5/ch12.html
[73] http://exploringjs.com/es6/ch_strings.html

Primitive Stringwerte vs. String-Objekte

- Primitive Stringwerte lassen sich mit den normalen Vergleichsoperatoren vergleichen.
- String-Objekte sind wie alle Objekte paarweise ungleich.
- Vermeide String-Objekte!

Beispiel 4.109.
```
'a' + 'bc' === 'ab' + 'c'; // true
typeof ('a' + 'bc');       // 'string'
'Bernd' <= 'Brot';         // true
String(5);                 // '5'
typeof String(5);          // 'string'
String(5) === '5';         // true
const s1 = new String('qwert');
const s2 = new String('qwert');
typeof s1;                 // 'object'
s1 === s2;                 // false
```

Zuweisungen

- Zuweisungen liefern den zugewiesenen Wert zurück.
- Für die meisten binären Operatoren `#` gibt es eine abgekürzte Version der Zuweisung `a = a # b`, nämlich `a #= b`.
- Zum Inkrementieren und Dekrementieren gibt es des Weiteren die bekannten Operatoren `++` und `--`.

Beispiel 4.110.
```
let assignment = 1331;
console.log(assignment);   // 1331
assignment += 5;
console.log(assignment);   // 1336
console.log(assignment++); // 1336
console.log(assignment);   // 1337
```

Bedingungsoperator

- „if-Anweisung als Ausdruck“
- Liefert einen Wert basierend auf einem Wahrheitswert.
- Struktur: `<condition> ? <valueIfTrue> : <valueIfFalse>`

Beispiel 4.111.
```
const isCorrect = false;
const state = isCorrect ? 'success' : 'fail'; // fail
```

Kontrollstrukturen

`if`

- Struktur (der `else`-Teil ist optional):

```
if (<condition>) {
  // stuff to do if <condition> is true
else {
  // stuff to do if <condition> is false
}
```

Beispiel 4.112.
```
if (false) {
  console.log('Logic error');
} else if (3 > 4) {
  console.log('Math error');
} else {
  console.log('You got it right!'); // <--
}
```

`switch`

- Struktur (default ist optional):

```
switch (<expression1>) {
case <expression2>:
  // stuff to do if equal to <expression2>
  // falls through
case <expression3>:
  // stuff to do if equal to <expression3>
  break;
default:
  // stuff to do if no case matched
  break; // optional, but recommended
}
```

- Ein beliebter Fehler ist, das `break` zu vergessen.

Beispiel 4.113.
```
const favouriteColor = 'red';
switch (favouriteColor) {
case 'red':
case 'blue':
case 'green':
  console.log('Good choice'); // <--
  break;
default:
  console.log(`What, you like ${favouriteColor}?!`);
  break;
}
```

`while`, `break`, `continue`

- `break` wird benutzt, um die Schleife abzubrechen (springt hinter die Schleife).
- `continue` wird benutzt, um den aktuellen Durchlauf abzubrechen (testet als nächstes die Bedingung).
- Struktur:

```
while (<condition>) {
  // do stuff
}
```

Beispiel 4.114.
```
while (true) {
  console.log('Still going'); // gets executed once
  break;
}
console.log('The end.');
```

`for`

- Struktur (alle drei Teile des Kopfes sind optional):

```
for (<init>; <condition>; <increment>) {
  // do stuff
}
```

Beispiel 4.115.
```
for (let i = 0; i < 3; i++) {
  console.log(i); // gets executed for 0, 1, 2
}
```

`do-while`

- Schleifenkörper wird immer mindestens einmal ausgeführt.
- Ein möglicher Anwendungsfall sind Benutzereingaben: Wiederholen, bis der Benutzer einen korrekten Wert eingegeben hat.
- Struktur:

```
do {
  // do stuff
} while (<condition>);
```

Beispiel 4.116.
```
let integer;
do {
  integer = Number(prompt('Enter an integer'));
} while (!Number.isInteger(integer));
console.log(integer);
```

`try`, `throw`

- Zum Abfangen von Fehlern.
- `finally` Zweig wird immer aufgerufen.
- Exceptions können beliebige Werte sein, sollten aber Instanzen der Klasse Error sein, damit Stacktraces erzeugt werden. Sie werden mit `throw` ausgelöst.
- Struktur (mindestens einer der `catch`/`finally`-Zweige muss vorhanden sein):

```
try {
  // do stuff that might fail
} catch (e) {
  // handle the error
} finally {
  // clean up
}
```

Beispiel 4.117. `try {`

```
  throw new Error('I\'ll kill your program!');
} catch (e) {
  console.log('Nah, I got you'); // <--
} finally {
  console.log('Cleaning up');    // <--
}
```

Funktionen

Funktionsdeklaration

- Struktur:

```
function <name>(<param1>, <param2>) {
  // stuff to do
}
```

- Funktion kann schon aufgerufen werden, bevor die Deklaration im Code erfolgt (*Hoisting*).

Beispiel 4.118. `myFunc();`

```
function myFunc () {
  console.log('Can be used above');
}
```

Funktionsausdruck

- Struktur (Zuweisung an Variable optional):

```
const <name> = function (<par1>, <par2>) {
  // stuff to do
}
```

- Funktionsausdrücke können erst nach der Initialisierung verwendet werden.

Beispiel 4.119.
```
expression(); // Error; not initialized
const expression = function () {
  console.log('Can only be used below');
};
```

Fat Arrow Functions

- Im Wesentlichen eine kurze Notation für Funktionsausdrücke.
- Klammern () um die Parameter sind optional, wenn es genau ein Parameter ist.
- Der Funktionskörper kann auch nur aus einem Ausdruck bestehen. Die geschweiften Klammern `{}` und das `return` werden entsprechend automatisch eingefügt.
- Struktur (mehrere Möglichkeiten):

```
const <name1> = (<param1>, <param2>) => {
  // stuff to do
};

const <name2> = <param> => <expression>;
```

Beispiel 4.120.
```
const fatArrow = () => {
  console.log('Shorthand notation');
};
fatArrow(); // Shorthand notation

const inc = x => x + 1;
console.log(inc(2)); // 3
```

Closures

- Ein Closure ist eine Funktion, die noch Zugriff auf die Umgebung bei ihrer Erstellung hat.

Beispiel 4.121.
```
const parent = function (parentParam) {
   const parentLocal = 'parentLocal';
   return (childParam) => {
      const childLocal = 'childLocal';
      console.log(parentParam, parentLocal, childParam, childLocal);
   };
};
const f = parent('parentParam');
f('childParam');  // parentParam parentLocal childParam childLocal
```

Functions as First-Class Citizens

In JavaScript, functions are *first-class objects*, i. e. they are objects and can be manipulated and passed around just like any other object. Specifically, they are `Function` objects.

Siehe https://developer.mozilla.org/de/docs/Web/JavaScript/Reference/Functions

Beispiel 4.122 (Funktion als Argument). `console.log([1, 2, 3].map(x => 2 * x));`
`// [2, 4, 6]`

Beispiel 4.123 (Funktion als Rückgabewert). `function lessThan(n) { return x => x < n; }`

```
lt5 = lessThan(5);
console.log(lt5);  // Function
lt5(4);   // true
lt5(42);  // false

console.log([1, 2, 3, 4].filter(lessThan(3))); // [1, 2]
```

Rest Parameters

- Erlaubt es Funktionen, eine variable Anzahl an Parametern zu verarbeiten.
- Innerhalb der Funktion wird der Rest-Parameter an ein Array der Werte gebunden.
- Muss am Ende der Parameterliste stehen.
- Struktur: `...<name>`

Beispiel 4.124. `const logAll = function (first, ...otherStrings) {`
```
  console.log(first, otherStrings);
};

logAll('Hello', 'new', 'world');
// Hello [ 'new', 'world' ]

logAll(); // undefined []
```

Spread Operator

- Wird benutzt, um eine Funktion mit einem Array der eigentlichen Parameter aufzurufen.
- Struktur: `...<array>`

Beispiel 4.125. `const max = (x, y) => (x > y ? x : y);`
```
console.log(max(99, 42)); // 99

const values = [1, 2];
console.log(max(...values)); // 2
```

Default Parameters

- Wenn beim Funktionsaufruf ein Parameter nicht angegeben wird, ist dieser standardmäßig mit undefined belegt.
- Default Parameter erlauben es, dieses Verhalten zu ändern.
- Struktur: `<param> = <default>`

Beispiel 4.126.
```
const test = function (lie = true) {
  return lie;
};
console.log(test()); // true
```

Arrays

Array-Literal

- Eckige Klammern um die gewünschten Werte.

Beispiel 4.127.
```
const empty = [];
console.log(empty) // []

const fibonacci = [1, 1, 2, 3, 5, 8, 13];
console.log(fibonacci); // [1, 1, 2, 3, 5, 8, 13]
```

Zugriff auf Elemente

- Erstes Element hat Index 0.
- Letztes Element hat Index `<array>.length - 1`.

Beispiel 4.128.
```
const fibonacci = [1, 1, 1, 3, 5, 8, 13];
fibonacci[2]    = 2;
console.log(fibonacci); // [1, 1, 2, 3, 5, 8, 13]
```

`for-of`

- Läuft über alle Werte im Array (oder Iterable) und führt für jeden den Schleifenkörper aus.
- Struktur:

  ```
  for (let v of <array>) {
    // do stuff
  }
  ```

- Oder, falls `v` im Schleifenkörper nicht geändert wird:

```
for (const v of <array>) { // do stuff }
```

Beispiel 4.129.
```
const fibonacci = [1, 1, 2, 3, 5, 8, 13];
for (const v of fibonacci) {
  console.log(v); // 1, 1, 2, 3, ...
}
```

Destructuring

- Erinnert an Pattern matching.
- Es wird von links nach rechts gematcht.
- Vergleichbar mit Rest Parametern bei Funktionen lassen sich überschüssige Werte auf der rechten Seite in einem Array auffangen.
- Kann auch zum Vertauschen von Variablen genutzt werden, ohne eine Hilfsvariable verwenden zu müssen.

Beispiel 4.130.
```
const [x, y, z] = [11, 42, 3];
console.log(x, y, z); // 11 42 3

const [head, ...tail] = [1, 2, 3, 4, 5];
console.log(head, tail); // 1 [2, 3, 4, 5]

let i = 2;
let j = 4;
console.log(i, j); // 2 4
[j, i] = [i, j];
console.log(i, j); // 4 2
```

Set

- In manchen Anwendungen Alternative zu Arrays.
- Enthält keine Duplikate.
- Schneller Test, ob ein Wert enthalten ist.
- Werte lassen sich wieder löschen.
- Unterstützt for-of Schleifen.
- Mehr dazu: Exploring ES6[74] und MDN[75].

[74] http://exploringjs.com/es6/ch_maps-sets.html#_set
[75] https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/

Beispiel 4.131.
```
const primes = new Set([1, 2, 3, 5, 7, 11, 13, 17]);
primes.add(19);
primes.delete(1);
console.log(primes.has(1)); // false
for (const p of primes) {
  console.log(p); // 2, 3, 5, 7, 11, 13, 17, 19
}
```

Objekte

Objekt-Literal

- Objekte verknüpfen Schlüssel (String oder Symbol) mit Werten.
- Schlüssel-Wert-Paare heißen *Properties*.
- Grundlage von JSON (JavaScript Object Notation).
- Struktur: `{ <key1>: <value1>, <key2>: <value2> }`

Beispiel 4.132.
```
const empty = {};
const andreas = {
  name: 'Andreas Kreuz',
  age:  26
};
```

Zugriff auf Elemente

- Zugriff über . nur möglich, wenn der Name ein gültiger Identifier ist, also als Variablenname verwendet werden könnte.
- Sonst muss man über [] zugreifen.
- Innerhalb [] kann ein beliebiger Ausdruck stehen, also zum Beispiel auch Variablen.

Beispiel 4.133.
```
const andreas = {
  name: 'Andreas Kreuz',
  age:  26
};
andreas.age = 27;
andreas['postal address'] = 'Olympus Mons, Mars';
console.log(andreas.name);    // Andreas Kreuz
console.log(andreas['name']); // Andreas Kreuz
```

`in`

- Testet, ob eine Property existiert.
- Gibt auch für geerbte Properties `true` zurück.

- Vorsicht! Ggf. `Map` verwenden.

Beispiel 4.134. `const tricky = { exists: undefined };`

```
tricky.exists // undefined
tricky.nope   // undefined

console.log('exists' in tricky); // true
console.log('nope' in tricky);   // false

const proto = { z: 42 };
const vec = Object.create(proto);
vec.x = 1; vec.y = 2;
console.log(vec, vec.z, 'z' in vec); // { x: 1, y: 2 } 42 true
```

`for-in`

- Läuft über die Schlüssel eines Objekts, inklusive der geerbten. Vorsicht!
- Struktur:

```
for (let key in <object>) {
  // do stuff
}
```

Beispiel 4.135.
```
const andreas = {
  name: 'Andreas',
  age:  26
};
for (let key in andreas) {
  console.log(key, andreas[key]); // name Andreas
                                  // age 26
}
```

Destructuring

- Erinnert an Pattern Matching.
- Rest Properties
- Properties können umbenannt werden: `<actualName>:<desiredName>`.

Beispiel 4.136.
```
const {name, age:years} = andreas;
console.log(name, years); // Andreas Kreuz 27
console.log(age);         // Reference Error

const {x, y, z} = vec;
console.log(x, y, z);  // 1 2 42

const {i, ...other} = {i: 1, j: 2, k: 3};
console.log(i, other); // 1 {j: 2, k: 3}
```

Beispiel 4.137 (Destructuring in Parametern).
```
function printFullName({firstName, lastName}) {
  console.log(`${firstName} ${lastName}`);
}

const person = {age: 31, firstName: 'Albert', lastName: 'Tross'};

printFullName(person);  // Albert Tross
```

Konstruktor

- Schablone für neue Objekte.
- In JavaScript sind Konstruktoren ganz normale Funktionen.
- Konvention: Funktionsname beginnt mit Großbuchstaben.
- Innerhalb des Konstruktors ist `this` das neue Objekt.
- Konstruktoren werden mit `new` aufgerufen.
- Der Prototyp des Konstruktors fasst in der Regel die Methoden der Objekte zusammen.
- Der Prototyp wird zwischen allen Instanzen geteilt.

Beispiel 4.138.
```
const Vec2Constructor = function (x, y) {
  this.x = x;
  this.y = y;
};

Vec2Constructor.prototype.add = function (v) {
  return new Vec2Constructor(
    this.x + v.x,
    this.y + v.y
  );
};

const vCon1  = new Vec2Constructor(0, 1);
const vCon2  = new Vec2Constructor(1, 0);
const result = vCon1.add(vCon2);
console.log(result); // Vec2Constructor {x: 1, y: 1}
console.log(vCon1.add === vCon2.add); // true
```

Klasse

- Nur syntaktischer Zucker für Konstruktorfunktionen.
- Die spezielle Methode `constructor` wird aufgerufen, wenn neue Instanzen mit `new` erzeugt werden.

- Methoden werden weiterhin über den Prototyp zwischen Instanzen geteilt.

Beispiel 4.139.
```
class Vec2Class {
  constructor(x, y) {
    this.x = x;
    this.y = y;
  }
  add(v) {
    return new Vec2Class(
      this.x + v.x,
      this.y + v.y
    );
  }
}
const vCla1  = new Vec2Class(0, 1);
const vCla2  = new Vec2Class(1, 0);
const result = vCla1.add(vCla2);
console.log(result); // Vec2Class {x: 1, y: 1}
console.log(vCla1.add === vCla2.add); // true
```

Vererbung

- Nur Einfachvererbung möglich.
- Schlüsselwort `extends`.
- Im Konstruktur der Unterklasse *muss* zuerst mittels `super` explizit der Oberklassenkonstruktor aufgerufen werden.
- `this` vor `super` zu verwenden ist ein Fehler.
- Unterklassen können Properties der Oberklasse überschreiben, man kann aber trotzdem über `super` auf diese zugreifen.

Beispiel 4.140.
```
class Person {
  constructor(name, age) {
    this.name = name;
    this.age = age;
  }
}

class Student extends Person {
  constructor (name, age, semester) {
    super(name, age);
    this.semester = semester;
  }
}

const karl = new Student('Karl Auer', 23, 8);
console.log(karl.name); // Karl Auer
```

Map

- Vergleichbar mit Objekten.
- Schlüssel können aber beliebige Werte sein, nicht nur Strings und Symbole.
- Schneller Test, ob ein Wert enthalten ist.
- Werte lassen sich wieder löschen.
- Unterstützt auch die for-of Notation.
- for-of muss `[key, value]` Einträge verarbeiten.
- Mittels Destructuring ergibt sich eine elegante Struktur.
- Mehr dazu: Exploring JS[76] und MDN[77].

Beispiel 4.141.
```
const scoreTable = new Map([
    ['Peter', 24], ['Sophie', 90], ['Klaus', 69]
]);
scoreTable.set('Laura', 27);
scoreTable.delete('Peter');
console.log(scoreTable.has('Peter')); // false
console.log(scoreTable.get('Laura')); // 27
for (const [name, score] of scoreTable) {
    console.log(name, score); // Sophie 90,
                              // Klaus 69,
                              // Laura 27
}
```

Methoden und Funktionen

- Normale Funktionsausdrücke besitzen alle ihr eigenes `this`.
- Wenn eine Funktion nicht als Methode aufgerufen wird, ist `this` `undefined`.

Beispiel 4.142.
```
const nums = [1, 2, 3];
nums.map(x => 2 * x);    // [ 2, 4, 6 ]

const map = nums.map;
map;  // [Function: map]
map(x => 2 * x);  // TypeError: Array.prototype.map called
                  // on null or undefined
```

Methoden und Funktionen

- Wenn eine normale Funktion nicht als Methode aufgerufen wird, ist `this` `undefined`.

[76] http://exploringjs.com/es6/ch_maps-sets.html#_map
[77] https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Map

Beispiel 4.143.
```
class NumberList1 {
  constructor(values) {
    this.values = values;
    this.sum    = 0;
    this.computeSum();
  }

  computeSum() {
    this.values.forEach(function (x) {
      this.sum += x; // Error; this is undefined!
    });
  }
}
```

Methoden und Funktionen

- Fat Arrow Funktionen erhalten ihr `this` von der umliegenden Funktion.

Beispiel 4.144.
```
class NumberList2 {
  constructor(values) {
    this.values = values;
    this.sum    = 0;
    this.computeSum();
  }

  computeSum() {
    // That works
    this.values.forEach(x => this.sum += x);
  }
}
```

Iterables, Iteratoren und Generatoren

Iterables

- Iterable: ein Objekt, das eine Methode `Symbol.iterator` besitzen.
- Diese Methode erzeugt Iteratoren.
- Solche Objekte können unter anderem in einer for-of Schleife benutzt werden oder mit dem Spread-Operator in ein Array umgewandelt werden.

Beispiel 4.145.
```
const iterable = [1, 2];
const iterator = iterable[Symbol.iterator]();

console.log(iterator.next());
// {value: 1, done: false}
```

```
console.log(iterator.next());
// {value: 2, done: false}

console.log(iterator.next());
// {value: undefined, done: true}
```

Iterator

- Ein „Zeiger“, um die Element in einer Datenstruktur zu durchlaufen.
- In JavaScript muss ein Iterator die Methode `next` implementieren.
- Diese Methode liefert bei jedem Aufruf Objekte der Form `{value: ..., done: ...}` zurück, wobei value der eigentliche Wert ist und done anzeigt, ob der Iterator fertig ist.
- Der Wert von done ist erst dann `true`, wenn der Iterator bereits über das Ende der Datenstruktur hinaus gelaufen ist.
- Mehr zu Iterables und Iteratoren: Exploring ES6[78].

Generatoren

- Funktionen, deren Ausführung pausiert und später fortgesetzt werden kann.
- Generatorfunktionen dienen als Erzeuger für Generatorobjekte.
- Mehrere aus eine Generatorfunktion erzeugte Generatorobjekte können sich in verschiedenen Stadien befinden.
- Einfacher Weg, um ein Objekt iterable zu machen.

Mehr zu Generatoren: Exploring ES6[79]

`function*`, `yield`

- Generatorfunktionen werden mit `function*` markiert.
- Mit `yield` wird ein Wert zurückgegeben und die Ausführung unterbrochen, bis der nächste Wert angefragt wird.
- Erst mit `return` (explizit oder implizit am Ende der Funktion) ist der Generator fertig.
- Es lassen sich somit endlose Generatoren realisieren.

[78] http://exploringjs.com/es6/ch_iteration.html
[79] http://exploringjs.com/es6/ch_generators.html

Beispiel 4.146.
```
const naturalNumbers = function* () {
  for (let i = 1; ; i++) {
    yield i;
  }
};

const it = naturalNumbers()[Symbol.iterator]();
console.log(it.next());
// { value: 1, done: false }
console.log(it.next());
// { value: 2, done: false }
console.log(it.next());
// { value: 3, done: false }
console.log(it.next());
// { value: 4, done: false }
console.log(it.next());
// { value: 5, done: false }
```

Beispiel 4.147.
```
const naturalNumbers = function* () {
  for (let i = 1; ; i++) {
    yield i;
  }
};
const take = function* (n, generator) {
  for (const v of generator) {
    if (n > 0) {
      yield v;
      n--;
    } else {
      return;
    }
  }
};
for (const n of take(10, naturalNumbers())) {
  console.log(n); // 1, 2, 3, ..., 10
}
```

Beispiel 4.148.
```
function* fibonacciNumbers () {
  let i = 0, j = 1;
  while (true) {
    yield j;
    [i, j] = [j, i + j];
  }
};

function* take (n, generator) {
  // same as above
};

console.log(...take(15, fibonacciNumbers()));
```

```
// 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610

console.log([...take(12, fibonacciNumbers())]);
// [ 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144 ]
```

`yield*`

- `yield*` liefert nacheinander alle Werte eines Generators.
- Vergleichbar mit normalem `yield` in einer for-of Schleife.

Beispiel 4.149 (Binärbaum, Inorder-Durchlauf).
```
class Tree {
  constructor(left, value, right) {
    this.left  = left;
    this.value = value
    this.right = right;
  }

  * [Symbol.iterator]() {
    if (this.left) {
      yield* this.left;
    }
    yield this.value;
    if (this.right) {
      yield* this.right;
    }
  }
}
```

Beispiel 4.150.
```
const tree = new Tree(
  new Tree(null, 1, null),
  5,
  new Tree(
    new Tree(null, 6, null),
    19,
    null
  )
);

console.log([...tree]); // [1, 5, 6, 19]
```

Promises und asynchrone Funktionen

Asynchrone Programmierung

- Beispiel: Programm soll nicht blockieren, während eine Datei geladen wird.
- Realisierung z. B. mit Callbacks

- *Callback Hell*: Viele verschachtelte Funktionsausdrücke.

Beispiel 4.151 (Callback im Node.fs File System API).
```
fs.readFile('/etc/passwd', (err, data) => {
  if (err) throw err;
  console.log(data);
});
```

Promises

- „Versprechen“ eines Wertes
- Viele neuere API-Funktionen liefern Promises zurück.
- Mithilfe des Promise-Konstruktors eigene Promises erzeugen.

Promises — neue APIs

- Die neue fetch-API basiert auf Promises. Sie löst XMLHttpRequests als Standardweg ab, asynchrone Anfragen an einen Server zu schicken.
- Mehr dazu: MDN[80]

Beispiel 4.152.
```
fetch('http://example.org')
  .then( res    => res.text())
  .then( text   => console.log(text))
  .catch(reason => console.log('Failed: ', error));
```

Beispiel 4.153 (`fetch` mit weiteren Parametern; JSON senden und empfangen).
```
var url = 'https://exa
var data = {username: 'example'};

fetch(url, {
  method: 'POST',
  body: JSON.stringify(data),
  headers:{
    'Content-Type': 'application/json'
  }
}).then(res => res.json())
  .catch(error => console.error('Error:', error))
  .then(response => console.log('Success:', response));
```

Promises

- „Versprechen“ eines Wertes
- Der Promise-Konstruktor erwartet eine Funktion, genannt *Executor*, mit zwei Parametern.

[80] https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API

- Die Parameter des Executors werden meist `resolve` und `reject` genannt und sind ebenfalls Funktionen.

Beispiel 4.154.
```
function slowAdd(a, b) {
  return new Promise(function (resolve, reject) {
    let result = a;
    for (let i = 0; i < b; i++) {
      if (result > 10000) {
        reject('I\'m bored');
        break;
      }
      result++;
    }
    resolve(result);
  });
}

slowAdd(12, 17)
  .then( result => console.log(result)) // 29
  .catch(reason => console.log('Failed: ', reason));
```

Promises (Forts.)

- Der Executor ruft die `resolve` Funktion auf, um das Ergebnis zurückzugeben und die `reject` Funktion, um einen Fehler anzuzeigen.
- Der Status eine Promises ist entweder *pending* oder *settled*. Bei settled wird zwischen *fulfilled* (`resolve` Funktion wurde aufgerufen) und *rejected* (`reject` Funktion wurde aufgerufen) unterschieden.
- Promises können zu einer Kette vereint werden (mittels `then`)
- Wenn irgendwo in der Kette etwas schief läuft, wird die gesamte Kette *rejected*.
- Rejection kann mit der `catch` Funktion behandelt werden.
- Mehr dazu: Exploring ES6[81].

Übersichtlicher als Callbacks.

Der resultierende Programmierstil fühlt sich aber immer noch nicht „natürlich“ an.

Asynchrone Funktionen

http://exploringjs.com/es2016-es2017/ch_async-functions.html

- Asynchrone Funktionen lassen asynchronen Code ähnlich wie synchronen aussehen, indem Promises versteckt werden.

[81] http://exploringjs.com/es6/ch_promises.html

- `async function` definiert eine asynchrone Funktion.
- Eine asynchrone Funktion gibt immer ein Promise zurück.
- `await` packt das Ergebnis aus, sobald es verfügbar ist.
- Nicht erfüllte Promises (rejected) werfen einen Fehler, der mit `try...catch` abgefangen werden kann.

Beispiel 4.155.
```
const getExample = async function () {
  const res  = await fetch('http://example.org');
  const text = await res.text();
  console.log(text);
};

getExample();  // returns Promise
```

Asynchrone Iteratoren

Bringen Iteratoren und Promises zusammen

Anwendungsszenario

- Anwendungsszenario: Wir wissen nicht (synchron), ob noch weitere Werte geliefert werden.
- Erinnerung: `[Symbol.iterator]` gibt einen (synchronen) Iterator zurück und macht ein Objekt zum (synchronen) Iterable.
- Erinnerung: (Synchrone) Iteratoren geben in ihrer next-Methode Objekte der Form `{value, done}` zurück.

Asynchrone Iteratoren

- Neu: `[Symbol.asyncIterator]` gibt einen asynchronen Iterator zurück und macht ein Objekt zum asynchronen Iterable.
- Neu: Asynchrone Iteratoren geben in ihrer next-Methode Promises zurück, die irgendwann zu Objekten der Form `{value, done}` resolven.

Schleifen

- Erinnerung: Mit for-of kann man die Werte eines (synchronous) Iterables verarbeiten.
- Neu: Mit for-await-of kann man die Werte eines asynchronous Iterables verarbeiten.
- Erinnerung: (Synchrone) Generatoren sind ein nützliches Werkzeug, um `[Symbol.iterator]` zu implementieren.

- Neu: Asynchrone Generatoren sind hilfreich für `[Symbol.asyncIterator]`.

Beispiel 4.156. `const file = {`

```
    content: ['Those', 'are', 'actually', 'lines', 'in', 'a', 'file'],
    async *[Symbol.asyncIterator]() {
        // Imagine we would receive the content from elsewhere
        // in several chunks.
        for (const line of this.content) {
            yield line;
            await sleep(500);
        }
    }
};

async function sleep(millis) {
    return new Promise((resolve, reject) => {
        setTimeout(resolve, millis);
    });
}
```

Beispiel 4.157 (Forts.). `async function printFile() {`

```
    for await (const line of file) {
        console.log(line);
    }
}

printFile();
```

Module

Module

- Echte Modularisierung, `import` und `export`.
- Alternative zu separaten script-Elementen im HTML-Dokument.
- Mehr zu Exports: MDN[82], mehr zu Imports: MDN[83].
- Mehr zu Modulen insgesamt: Exploring ES6[84]

Standardexport

- Maximal einer pro Modul.
- Wird mit einem Standardimport importiert.
- Häufigste Syntax:

[82] https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/export
[83] https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/import
[84] http://exploringjs.com/es6/ch_modules.html

- `export default <expression>;`
- `export default function (...) { ... }`
- `export default class (...) { ... }`

Beispiel 4.158 (`file.js`). `export default class {`
```
  constructor() {
    console.log('Called');
  }
}
```

Standardimport

- Importiert den Standardexport eines Moduls unter einem gegebenen Namen.
- Syntax: `import <alias> from <path>;`

Beispiel 4.159. `import MyClass from 'path/to/file.js';`

```
const def = new MyClass(); // Called
```

Benannter Export

- Wird mit einem benannten Import importiert.
- Häufigste Syntax: `export <declaration>`

Beispiel 4.160 (`file.js`). `export const hello = 'Hello';`

Benannter Import

- Importiert Objekte/Funktionen/Werte mit den angegeben Namen aus dem entsprechenden Modul. Syntax: `import {<name1>, <name2>} from <path>;`
- Man kann diese innerhalb des importierenden Moduls umbenennen, etwa um Kollisionen zu verhindern. Syntax: `import {<name> as <alias>} from <path>;`
- Man kann alle benannten Exports eines Moduls zu einem Objekt bündeln und so importieren (*qualified import*). Syntax: `import * as <alias> from <path>;`

Beispiel 4.161. `import {hello as greeting} from 'path/to/file.js';`

```
console.log(greeting); // Hello
console.log(hello);    // Reference Error
```
Beispiel 4.162. `import * as obj from 'path/to/file.js';`

```
console.log(obj.hello); // Hello
```

Leerer Import

- Manchmal möchte man ein Skript nur laden, interessiert sich aber nicht für die Dinge, die es exportiert.
- Der erste leere Import in einem Programm lädt es nicht nur, sondern führt es auch aus.
- Syntax: `import <path>;`

Beispiel 4.163 (`file.js`). `console.log('The end!');`

Beispiel 4.164. `import 'path/to/file.js'; // The end!`

Browser und DOM

- Introduction to the DOM — MDN[85]
- Document Object Model (DOM) — MDN[86]

JavaScript einbinden

- JavaScript kann im Browser ausgeführt werden. Dazu müssen Skripte in ein HTML-Dokument eingebunden werden.
- Ähnlich zu CSS kann man entweder auf ein externes Skript verweisen, das Skript direkt in das HTML-Dokument schreiben oder kurze Programmstücke als Attribute an Elemente anhängen.
- Für ES6 Module fügt man zum `script`-Element noch das Attribut `type='module'` hinzu.

Beispiel 4.165. `<h1>Testing alert</h1>`

```
<!-- Extern -->
<script src="js/hello.js"></script>
<script src="js/module.js" type="module"></script>

<!-- Intern -->
<script>alert('hello!');</script>

<!-- Inline -->
<button onclick="alert('Boom!')">DO NOT PRESS</button>
```

DOM Operationen

- JavaScript kann auch dazu verwendet werden, den Elementbaum eines HTML-Dokuments (DOM) zu manipulieren.

[85] https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Introduction
[86] https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model

- So kann man neue Elemente hinzufügen oder Textknoten austauschen.

Beispiel 4.166 („Volltextsuche“).
```
function talksAbout(node, string) {
  if (node.nodeType === document.ELEMENT_NODE) {
    for (const child of node.childNodes) {
      if (talksAbout(child, string))
        return true;
    }
    return false;
  } else if (node.nodeType === document.TEXT_NODE) {
    return node.nodeValue.includes(string);
  }
}

talksAbout(document.body, 'Bonn');
```

DOM — Datentypen und Vererbungshierachie

```
EventTarget
  |
  +- Node
       |
       +- Document (implements ParentNode)
       |
       +- DocumentFragment (implements ParentNode)
       |
       +- Element (implements ParentNode, ChildNode)
```

Und ca. 30 weitere...

`NodeList`

- Methoden, die mehrere Element zurückgeben können, geben immer eine `NodeList` zurück.
- Eine `NodeList` hat wie ein `Array` eine `length`-Property und man kann auf Elemente über `[<index>]` zugreifen.
- Andere `Array`-Methoden wie `map` oder `forEach` fehlen.
- Sie sind aber Iterables, so dass man sie in for-of-Schleifen benutzen kann und mit `[...<nodeList>]` in ein echtes Array umwandeln kann.
- Es gibt einerseits live `NodeLists`, die automatisch angepasst werden, wenn passende Elemente zum Dokument hinzugefügt werden (alle `getElementBy...` Methoden).
- `querySelector` und `querySelectorAll` geben static NodeLists zurück, die nicht mehr verändert werden.

DOM — Selection

- Wir müssen zunächst die Elemente auswählen, die wir bearbeiten möchten.
- Bevor man auf das DOM zugreift, muss das HTML komplett geparst sein. Man kann entweder
  - das `script`-Element ganz an das Ende des `body`-Elements setzen,
  - einen Listener auf das `DOMContentLoaded`-Event setzen, oder
  - dem `script`-Element das Attribut `defer` hinzufügen.

Beispiel 4.167 (`defer`). `<script defer src="..."></script>`

Elemente nach Id/Klassen/Tagnamen/CSS-Selektoren auswählen

```
Document.getElementById()
<Element/Document>.getElementsByClassName()
<Element/Document>.getElementsByTagName()
<Element/Document>.querySelector()
ParentNode.querySelectorAll()
```

`querySelector()` gibt nur den ersten Treffer zurück.

`querySelectorAll` liefert eine `NodeList`.

Beispiel 4.168.
```
const $addComment = document.getElementById('add-comment');
const $h3 = document.getElementsByTagName('h3');
const $fields = $addComment.getElementsByClassName('field');
const $blamebotComments =
document.querySelectorAll('article[data-author='blamebot']');
```

Attribute

- Die Standardattribute von HTML-Elementen sind auch als Properties der DOM-Objekte verfügbar.
- Der Typ eines `input`-Elements kann beispielsweise über `.getAttribute('type')` oder über `.type` abgefragt werden.

Klassen hinzufügen/umschalten/entfernen/testen

```
Element.classList.add()
Element.classList.toggle()
Element.classList.remove()
Element.classList.contains()
```

Sonstige Attribute abfragen/setzen/entfernen/testen

```
Element.getAttribute()
Element.setAttribute()
Element.removeAttribute()
Element.hasAttribute()
```

Beispiel 4.169. `for (const comment of $blamebotComments) {`
```
  comment.classList.add('toxic');
  comment.setAttribute('data-modaction', 'remove');
}
```

Manipulation

- Erzeugen und Hinzufügen von Elementen
- Inhalt bestehender Elemente verändern

```
Element.insertAdjacentElement()
Element.insertAdjacentHTML()
Element.insertAdjacentText()

ParentNode.prepend() [Experimental]
ParentNode.append()  [Experimental]
ChildNode.after()    [Experimental]
ChildNode.before()   [Experimental]
Node.appendChild()
Node.insertBefore()

ChildNode.remove() [Experimental]

ChildNode.replaceWith() [Experimental]
```

Manipulation (Forts.)

```
Node.textContent
Element.innerHTML

Node.cloneNode()

document.createElement()
document.createTextNode()
document.createDocumentFragment()
```

Beispiel 4.170. `function addComment(heading, $content, author = 'anonymous') {`
```
  const $newComment = document.createElement('article');
  $newComment.classList.add('comment');
  $newComment.setAttribute('data-author', author)
```

```
  $newComment.innerHTML = `<h3>${heading}</h3>`
  $newComment.appendChild($content);
  $addComment.insertAdjacentElement('beforeBegin', $newComment);
}
```

Traversierung

- Entlanglaufen im DOM-Baum, ausgehend von einem Knoten

Kindknoten

```
Node.childNodes
ParentNode.children
Node.firstChild
ParentNode.firstElementChild
Node.lastChild
ParentNode.lastElementChild
```

Wählen auch Textknoten/Kommentare/andere Instanzen von `Node` aus.

Geschwisterknoten

```
Node.nextSibling
Element.nextElementSibling
Node.previousSibling
Element.previousElementSibling
```

Elternknoten

```
Node.parentNode
Node.parentElement
```

`parentNode` wählt ggf. auch das `document` aus.

Vorfahren

```
Element.closest() [Experimental]
```

Läuft den DOM-Baum nach oben, bis ein Element gefunden wurde, das auf den gegebenen CSS-Selektor passt.

Filter

```
Element.matches()
```

Testet, ob das Element auf den CSS-Selektor passt.

Beispiel 4.171. `let commentCount = 0;`

```
let removedCount = 0;
let formCount = 0;
for (const $child of document.body.children) {
  if ($child.matches('article.comment')) {
    commentCount++;
    if ($child.getAttribute('data-modaction') === 'remove') {
      const heading = $child.firstElementChild.textContent;
      console.log(`Removing comment "${heading}"`);
      $child.remove();
      removedCount++;
    }
  } else if ($child.matches('form')) {
    formCount++;
  }
}
console.log(`Found ${commentCount} comments (removed ${removedCount})
and ${formCount} forms.`);
```

Events

- Man kann mehrere Listener für den gleichen Eventtyp bei einem Element registrieren.
- Browser haben ein Standardverhalten für bestimmte Events, etwa das Öffnen einer Seite, wenn auf einen Link geklickt wird.
- Events wandern erst von außen nach innen (capturing) und anschließend wieder nach außen (bubbling).
- Bei EventListenern kann man angeben, ob sie während der Capturing- oder der Bubbling-Phase auf das Event reagieren sollen.

https://stackoverflow.com/questions/4616694/what-is-event-bubbling-and-capturing

`EventTarget`

```
EventTarget.addEventListener()
EventTarget.removeEventListener()
EventTarget.dispatchEvent()
```

`Event`

`Event.target` das Element, auf dem das Event registriert wurde

`Event.currentTarget` das Element, zu dem der EventListener gehört

`Event.preventDefault()` verhindert das Standardverhalten des Browser

`Event.stopPropagation()` verhindert weiteres Bubbling/Capturing des Events

Beispiel 4.172.
```
$addComment.addEventListener('submit', event => {
  event.preventDefault(); // don't refresh page
  const heading = $addComment.querySelector('#comment-heading').value;
  const content = $addComment.querySelector('#comment-content').value;
  const $content = document.createElement('p');
  $content.innerHTML = content.replace(/\n/g, '<br>');
  addComment(heading, $content);
});
```

Browser-Unterstützung

- Den aktuellen Browserstatus der Methoden und Properties kann man bei http://caniuse.com nachschlagen.

- Solange man keine alten Versionen von IE unterstützen muss, kann man auf die Kompatibilitätsschicht von Bibliotheken wie jQuery verzichten.
- Einige Funktionen von jQuery, wie Animationen, sollten außerdem wenn möglich gemieden werden. In diesem Fall sind CSS-Animationen besser, da die GPU hier eingesetzt werden kann.
- Die API von jQuery bietet dennoch einige Abkürzungen, die die Entwicklung beschleunigen können.
- Es gibt allerdings auch leichtgewichtigere Alternativen mit weitgehend gleicher API, beispielsweise http://zeptojs.com/.

4.10 Web-Applikationen

Web-Applikationen

Web Publishing

- *Websites* sind thematische Sammlungen von Texten, Bildern, Sounds usw.
- Websites können sich über mehrere Webserver erstrecken, oder sich einen Webserver mit anderen Websites teilen.

Techniken

- statische Seiten
- Common Gateway Interface (CGI)

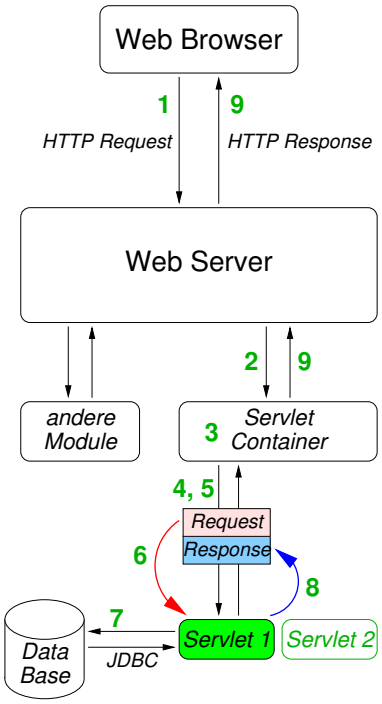
- Scripts, Libraries, APIs, (Programmier-)Sprachen, Frameworks, ...
- Application Server
- Content Management Systems

Java Servlets

- Standardisierte Java-Komponenten, die in *Servlet Containern* laufen.
- Ein Servlet Container kontrolliert die aktiven Servlets innerhalb des Web-Servers.
- Der Servlet Container läuft im Webserver-Prozess, oder als eigener Prozess (stabiler).
- Standardisierte (XML-basierte) Konfigurationsdateien ⇒ gute Portierbarkeit.
- Servlet-Programmierparadigma: Request/Response, basierend auf dem unterliegenden HTTP
- Flexible Session-Verwaltung
- ...

Java Servlets — Kontrollfluss am Beispiel

1. Webbrowser: HTTP-Request
2. Entscheidung für Servlet Container
3. Auswahl des Servlets
4. Ggf. Initialisierung des Servlets
5. Aufruf des Servlets, Argumente: (Request-Objekt, Response-Objekt)
6. Username, Session, HTML-Formularparameter, ...
7. Beliebige Berechnungen, insbes. JDBC-Zugriff auf RDBS
8. Ergebnisse im Response-Objekt ablegen
9. Flush, zurück zum Webserver



Java Servlets

Beispiel 4.173.

```
public class SimpleServlet extends HttpServlet {

    public void doGet(HttpServletRequest request, HttpServletResponse response)
      throws ServletException, IOException {
        // set header field first
        response.setContentType("text/html");

        // then get the writer and write the response data
        PrintWriter out = response.getWriter();
        out.println("<head><title> SimpleServlet Output</title>");
        out.println("</head><body>");
        out.println("<h1>SimpleServlet Output</h1>");
        out.println("<p>This is output from SimpleServlet.</p>");
        out.println("</body>");
        out.close();
    }

    public String getServletInfo() {
        return "A simple servlet";
    }
}
```

Node.js — JavaScript auf dem Server

Node.js

- basiert auf V8 (JavaScript-Engine in Google Chrome)
- event-based, non-blocking I/O; asynchron
- Modulsystem (CommonJS, ES6-Module)
- Nutzt nur einen Prozessorkern: Mehrere Prozesse notwendig statt mehreren Threads
- Paketverwaltung NPM

- https://nodejs.org/
- http://expressjs.com/

Express.js

Express.js — Web Application Framework for Node.js

- Routing
- Middleware
- Template Engines
- Debugging
- Database integration

Beispiel 4.174 (Routing).
```
var express = require('express');
var app = express();

// respond with "hello world" when a GET request
// is made to the homepage
app.get('/', function(req, res) {
  res.send('hello world');
});

app.get('/about', function (req, res) {
  res.send('about');
});

app.all('/secret', function (req, res, next) {
  console.log('Accessing the secret section ...');
  next(); // pass control to the next handler
});
```

- http://expressjs.com/

Content Management Systeme (CMS)

Eigenschaften

Trennung von Layout und Inhalt Autoren und Redakteure: Pflege der Website ohne HTML-, Skriptsprachen- oder Designkenntnisse

Admin bzw. Designer: Seitenlayout jederzeit änderbar

Templates Automatische Generierung der Seiten und der Navigation (Inhaltsverzeichnisse, Sitemaps, ...)

Workflow und Rechte ...

Personalisierung ...

Versionierung *Branching* und *Merging*: Historische Versionen erhalten und pflegen, Archivierung

Content Management Systeme (CMS)

Eigenschaften

Content Lifecycle Erstellen, Publizieren, Entfernen und/oder Archivieren, teilweise unter automatischer zeitabhängiger Steuerung

Dynamische / statische Seitenerzeugung Statische Seiten sind „effizienter“ bei hoher Last. Dynamische Seiten sind immer aktuell, personalisierbar, und belasten den Host stärker. ⇒ Caching

Schnittstellen für Daten und Funktionen, Webservices

5 XML Path Language (XPath)

5.1 Einleitung

XML Path Language

Navigation in XML-Bäumen

Finde

- das erste `person`-Element,
- das 7. Kind des dritten `person`-Elements,
- alle `documentation`-Elemente,
- den jeweils ersten Paragraphen jedes Kapitels.

XPath

- ist eine XML-Anfragesprache,
- ist eine funktionale Sprache,
- adressiert Teile von XML-Dokumenten,
- wird normalerweise in andere Sprachen eingebettet.

XPath-Einbettungen

Beispiel 5.1 (Verwendung in XSLT).

```
<xsl:template match="tr/td">
  ...
    <xsl:apply-templates select="../../tr/td[$i]"/>
  ...
</xsl:template>
```

```
<xsl:template match="@*|*|text()|processing-instruction()">
...
</xsl:template>
```

Weitere Einbettungen

- Teilmenge von XQuery
- Constraints in XML Schema
- XForms (Datenzugriff)
- ...

Spezifikationen

- **XML Path Language (XPath) Version 1.0** W3C Recommendation 16 November 1999
- **XML Path Language (XPath) 2.0**[87]
- **XQuery 1.0 and XPath 2.0 Functions and Operators**[88]
- W3C Recommendations 23. Januar 2007, bzw. Second Edition: 14. Dezember 2010
- Weitere Spezifikationen:
  - XQuery 1.0 and XPath 2.0 Data Model (XDM)[89]

[87] http://www.w3.org/TR/xpath20/
[88] http://www.w3.org/TR/xquery-operators/
[89] http://www.w3.org/TR/xpath-datamodel/

  - XQuery 1.0 and XPath 2.0 Formal Semantics[90]
  - XSLT 2.0 and XQuery 1.0 Serialization[91]
  - XQuery and XPath Full Text 1.0
  - ...
  - XML 1.0/1.1, Namespaces in XML, XML Schema, XML Information Set, ...
- Ausblick: **XML Path Language (XPath) 3.1** W3C Recommendation 21 March 2017[92]

XPath — Übersicht

- Gemeinsame Syntax und Semantik für zentrale Funktionalität von XSLT, XQuery, XML Schema, XPointer, ...,
- arbeitet auf der logischen Struktur (Bäume),
- benutzt eine Pfad-Notation (wie in Filesystemen oder URLs) zur Navigation innerhalb der Elementhierarchie,
- kann auch Strings, numerische Werte und andere XML-Schema-Datentypen verarbeiten,
- hat eine große Funktionsbibliothek,
- besitzt eine kompakte Syntax (nicht XML) zur Verwendung in URLs und Attributwerten,
- ist erweiterbar (Datentypen und Funktionen).

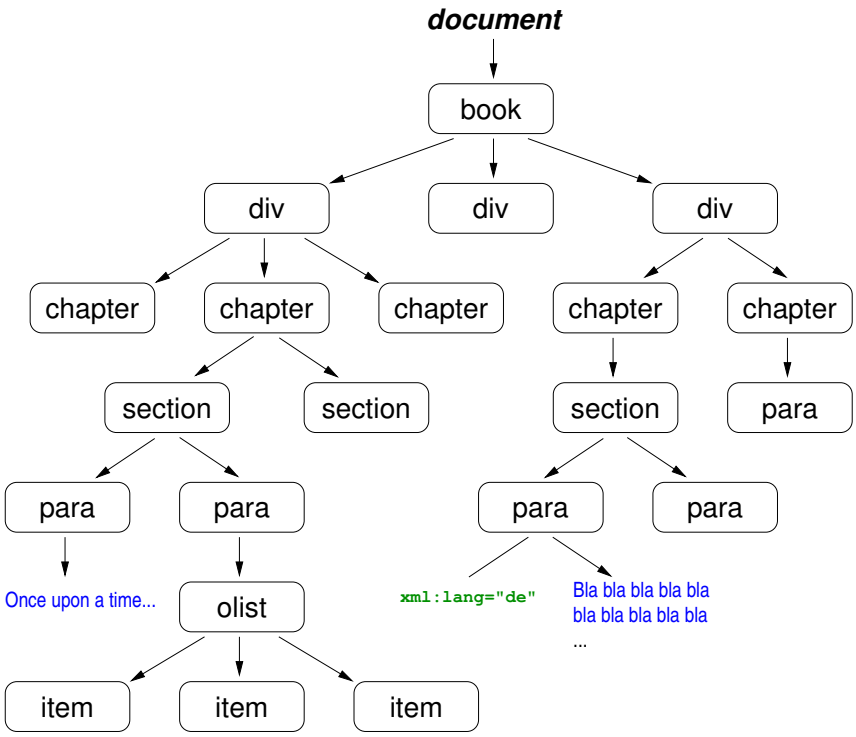
5.2 Datenmodell

Datenmodell

- XPath operiert auf Bäumen. Die Knotentypen sind:
  - Dokumentknoten (*Document Node*)
  - Elementknoten (*Element Nodes*)
  - Textknoten (*Text Nodes*)
  - Attributknoten (*Attribute Nodes*)
  - Namespace-Knoten (*Namespace Nodes*)
  - Kommentarknoten (*Comment Nodes*)
  - Verarbeitungsanweisungen (*Processing Instruction Nodes*)
- Ein XPath-Datenmodell kann aus einem *XML Information Set* konstruiert werden. (Der Algorithmus ist in der XPath Recommendation angegeben.)

[90] http://www.w3.org/TR/xquery-semantics/
[91] http://www.w3.org/TR/xslt-xquery-serialization/
[92] https://www.w3.org/TR/xpath-3/

Beispiel Datenmodell



5.3 Ausdrücke und Auswertung

5.3.1 Kontext

Auswertungskontext

XPath ist eine funktionale Sprache

- Zentrales Konzept sind *Ausdrücke*
- Auswertung liefert stets eine Liste (*Sequence*)

Konzepte und Begriffe

- Statischer Kontext / dynamischer Kontext
- Serialisierung (als XML, XHTML, HTML oder Text)
- *Document Order*
- Typen, *XML Schema*, *Sequence Type*
- ...

Statischer Kontext

Bestandteile des statischer Kontexts

- XPath 1.0 compatibility mode
- Statically known namespaces
- Default element/type namespace
- Default function namespace
- In-scope schema definitions
- In-scope variables
- Statically known documents
- ...

Vordefinierte Namespaces

Präfix	Namespace
`xs`	`http://www.w3.org/2001/XMLSchema`
`fn`	`http://www.w3.org/2005/xpath-functions`
`xdt`	`http://www.w3.org/2005/xpath-datatypes`
`err`	`http://www.w3.org/2005/xqt-errors`

Dynamischer Kontext

Bestandteile des dynamischen Kontexts

- ein *Context Item* (Knoten oder atomarer Wert), `"."`
- die *Kontextgröße* (integer; *Kontextgröße* ≥ 1), `"fn:last()"`
- die *Kontextposition* (integer; $1 \leq$ *Kontextposition* $\leq$ *Kontextgröße*), `"fn:position()"`
- eine Menge von *Variablenbindungen*, `"$QName"`
- eine *Funktionsbibliothek*
- *Current dateTime*, *timezone*, ...
- ...

5.3.2 Pfadausdrücke

Pfadausdrücke (Path Expressions)

Pfadausdrücke wählen eine Knotensequenz aus (relativ zu einem Kontextknoten).
Beispiel 5.2.

```
/descendant::chapter/child::para[fn:position()=1]
```

Pfadausdrücke bestehen aus einer Folge von *Schritten* (*Steps*).

Schritt (Step)

- besteht aus drei Teilen:
  1. einer *Achse*, die eine Sequenz von Knoten liefert,
  2. einem *Knotentest*, der Knotentyp und -namen selektiert, und
  3. optionalen *Prädikaten* für weitere Selektionen
- Die Auswertung erfolgt analog in drei Schritten.
- ausführliche Notation / abgekürzte Notation

Achsen (1)

`child` enthält alle Kinder des Kontextknotens.
(Attribute sind keine Kindknoten!)

`descendant` ist der transitive Abschluss der Achse `child`.

`parent` enthält den Vaterknoten des Kontextknotens.

`ancestor` ist der transitive Abschluss der Achse `parent`.

`following-sibling` bzw. `preceding-sibling` enthalten die Geschwister des Kontextknotens, die hinter bzw. vor dem Kontextknoten stehen.

`following` enthält alle `descendant`s des Dokumentknotens, die in Dokumentreihenfolge hinter dem Kontextknoten stehen, und nicht `descendant` des Kontextknotens sind.

`preceding` enthält alle `descendant`s des Dokumentknotens, die vor dem Kontextknoten stehen, und nicht `ancestor` des Kontextknotens sind.

Achsen (2)

`attribute` enthält alle Attribute des Kontextknotens, falls dieser ein Element ist. Sonst die leere Sequenz.

`self` enthält genau den Kontextknoten.

`descendant-or-self` enthält den Kontextknoten und dessen `descendant`s.

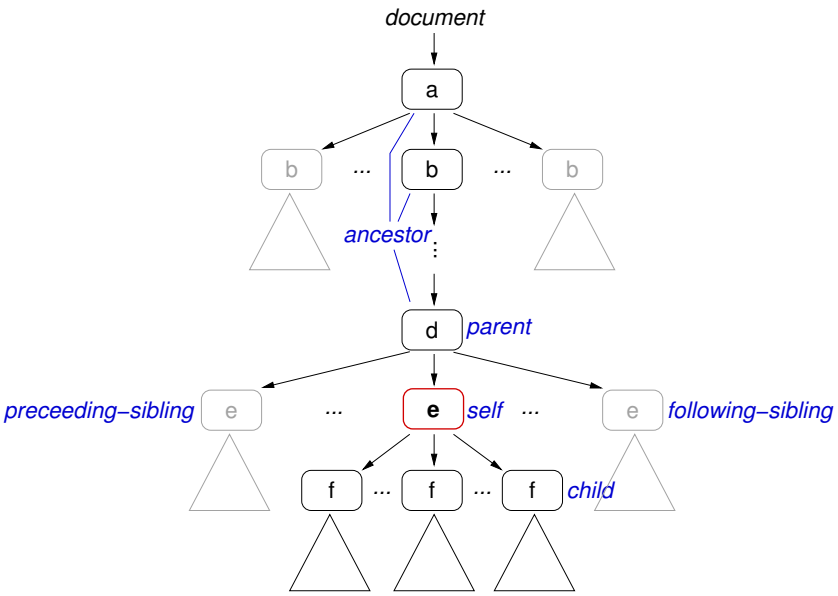
`ancestor-or-self` enthält den Kontextknoten und dessen `ancestor`s.

`namespace` enthält alle Namespaces des Kontextknotens.

Beobachtung (zum Nachdenken)

„The `ancestor`, `descendant`, `following`, `preceding` and `self` axes partition a document (ignoring attribute and namespace nodes): they do not overlap and together they contain all the nodes in a document.“

Achsen (3)



Achsen (4)

Bei der Auswertung von Steps unterscheiden wir

Rückwärtsachsen `parent`, `ancestor`, `ancestor-or-self`, `preceding`, `preceding-sibling`

Vorwärtsachsen (alle anderen Achsen)

Die resultierende Sequenz ist immer in *Dokumentreihenfolge.*

Kontextposition

- Vorwärtsachsen weisen ihren Knoten die *Kontextposition* in Dokumentreihenfolge zu.
- Rückwärtsachsen weisen ihren Knoten die *Kontextposition* in umgekehrter Dokumentreihenfolge zu.

Knotentests

Vorselektion des Knotentyps durch die Achse

Achse	**Haupt-Knotentyp**	**andere Knotentypen**
`attribute`	Attribut	
`namespace`	Namespace	
alle anderen	Element	Text, Processing Instruction, Kommentar, Dokument

QName oder ein Test der Form *prefix*:* ist erfüllt, wenn

1. der Typ der Haupt-Knotentyp der Achse ist, und
2. die *expandierten* qualifizierten Namen übereinstimmen.

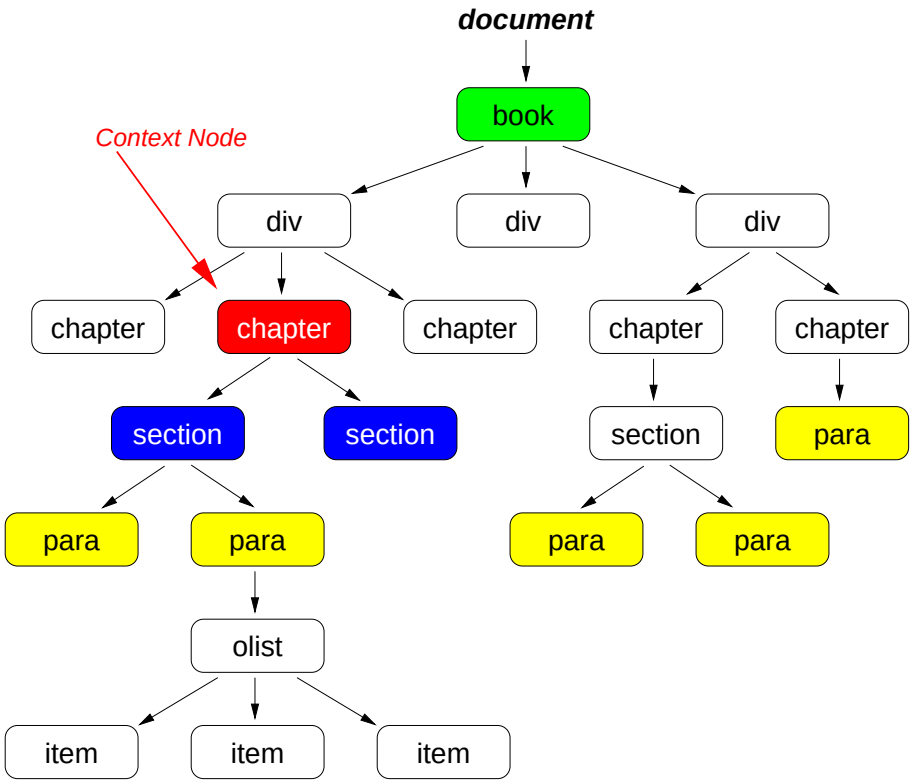
* selektiert alle Knoten des Haupt-Knotentyps.
Beispiele 5.3 („NameTest“-Knotentests).

```
child::para           attribute::href
child::*              attribute::html:*
child::text()
```

Auswertung eines Steps

1. die Achse liefert eine Sequenz,
2. der Knotentest filtert Knoten aus dieser Sequenz,
3. jedes Prädikat filtert weitere Knoten.

Beispiele




```
child::*
*
```

```
/descendant::para
```

```
ancestor::book
```

Beispiele: „KindTest“-Knotentests

- *node()* matches any node.
- *text()* matches any text node.
- *comment()* matches any comment node.
- *element()* matches any element node.
- *element(person)* matches any element node whose name is person, regardless of its type annotation.
- *element(person, surgeon)* matches any non-nilled element node whose name is person, and whose type annotation is surgeon or is derived from surgeon.
- *element(*, surgeon)* matches any non-nilled element node whose type annotation is surgeon (or is derived from surgeon), regardless of its name.
- *attribute()* matches any attribute node.

- *attribute(price)* matches any attribute whose name is price, regardless of its type annotation.
- *attribute(*, xs:decimal)* matches any attribute whose type annotation is xs:decimal (or is derived from xs:decimal), regardless of its name.

Prädikate

Auswertung der Prädikate

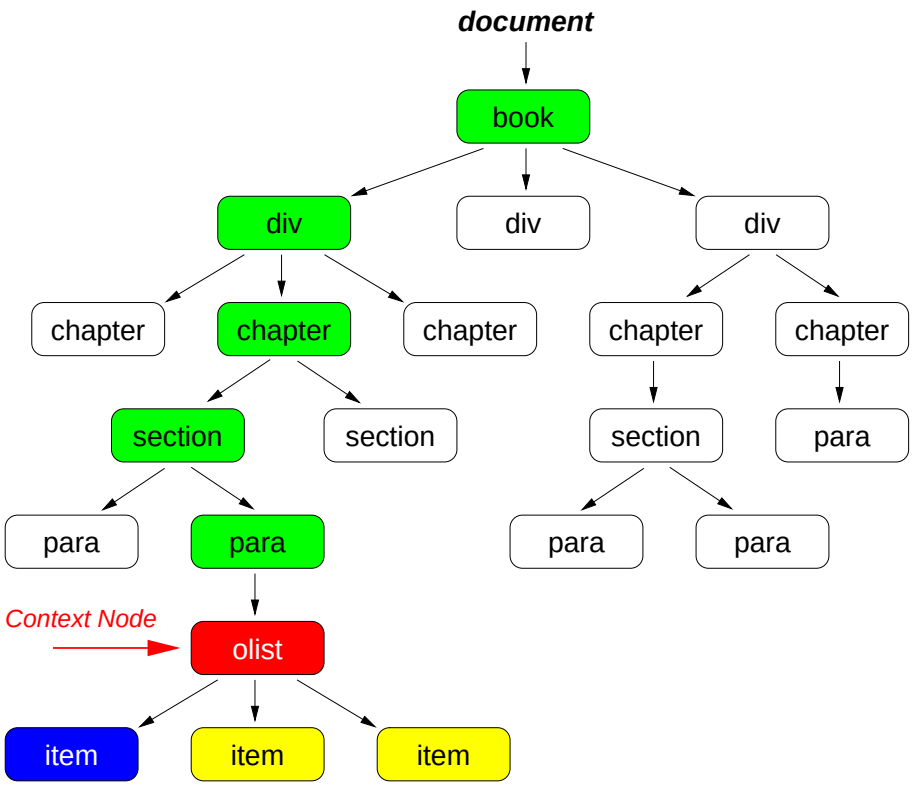
- Jedes Prädikat schränkt die aktuelle Sequenz weiter ein.
- Dazu wird das Prädikat für jeden Knoten der Sequenz ausgewertet.
- Der *Auswertungskontext* ist dabei wie folgt:

  Kontextknoten ist der zu testende Knoten,

  Kontextgröße ist die Größe der aktuellen Knotensequenz,

  Kontextposition ist die Position des Knotens in der Sequenz bzgl. der durch die Achse gegebenen Richtung.
- Die verbleibende Sequenz wird durch das nächste Prädikat gefiltert, usw.

Beispiele



```
child::item[fn:position()=1]
item[1]
```

```
child::item[fn:position()>1]
ancestor::*
```

Kombination von Steps

Sei E_1 ein *Step* und E_2 ein Ausdruck.

Welche Sequenz liefert E_1/E_2?

Zunächst wird $S = \text{eval}(E_1)$ ausgewertet.

S ist also eine Sequenz von Knoten.

Für jeden Knoten $N \in S$ wird E_2 ausgewertet, mit

dynamischer Kontext

- *Context Item* ist N,
- Kontextgröße ist die Länge von S,
- Kontextposition ist die Position von N in S (bzgl. der Achsenrichtung von E_1).

Die resultierenden Sequenzen werden zu einer neuen Sequenz S' zusammengefügt.

Falls S' nur Knoten enthält, werden alle Duplikate eliminiert.

Anschließend wird nach der Dokumentreihenfolge sortiert.

$\text{eval}(E_1/E_2)$ ist definiert als das Resultat dieser Operationen.

Falls S' nur atomare Werte enthält, ist $\text{eval}(E_1/E_2)$ definiert als S'.

Falls S' Knoten und atomare Werte enthält, ist das ein *Fehler*.

Beispiel 5.4. `/descendant::chapter/child::section`

Abgekürzte Notation

- `child::` ist die Default-Achse (kann also weggelassen werden)
- `attribute::` kann durch `@` abgekürzt werden
- `//` steht für `/descendant-or-self::node()/`
- `..` steht für `parent::node()`

- *numerisches Prädikat*: `[n]` (*n* von numerischem Typ) ist äquivalent zu `[fn:position() eq n]`.

Beispiel 5.5. `item[1]/@class`

```
child::item[fn:position() eq 1]/attribute::class
```

Beispiele zu Pfadausdrücken (1)

Beispiel 5.6. Alle `para`-*Elemente*:

```
/descendant::para
```

Beispiele 5.7 (mit Attributen). Alle `xml:lang`-*Attribute* aller `para`-Elemente:

```
/descendant::para/@xml:lang
```

Alle `para`-*Elemente*, die ein `xml:lang`-Attribut besitzen:

```
/descendant::para[@xml:lang]
```

Alle `para`-*Elemente*, die ein `xml:lang`-Attribut mit dem Wert `"de"` besitzen:

```
/descendant::para[@xml:lang="de"]
```

Beispiel 5.8 (Selektion nach Elementknoten). Alle `para`-*Elemente*, die ein Unterelement `olist` besitzen:

```
/descendant::para[olist]
```

Alle `para`-*Elemente*, die ein Unterelement `strong` mit dem Stringwert `"Stop"` besitzen:

```
/descendant::para[strong = "Stop"]
```

Beispiele zu Pfadausdrücken (2)

- *`child::chapter[2]`*
- *`child::employee[secretary][assistant]`* and *`employee[secretary][assistant]`* select all the `employee` children of the context node that have both a secretary child element and an *assistant* child element
- *`child::*`* and *`*`* select all element children of the context node
- *`child::text()`* and *`text()`* select all text node children of the context node
- *`attribute::*`* and *`@*`* select all the attributes of the context node
- *`child::*/child::para`* selects all `para` grandchildren of the context node
- *`child::para[fn:position()=1]`* selects the first `para` child of the context node
- *`child::para[fn:position()=fn:last()]`* selects the last `para` child of the context node

Beispiele zu Pfadausdrücken (3)

- *para[@type="warning"]* selects all para children of the context node that have a type attribute with value warning
- *para[@type="warning"][5]* selects the fifth para child of the context node that has a type attribute with value warning
- *para[5][@type="warning"]* selects the fifth para child of the context node if that child has a type attribute with value warning
- *employee[@secretary and @assistant]* selects all the employee children of the context node that have both a secretary attribute and an assistant attribute
- If *E* is any expression that returns a sequence of nodes, then the expression *E/.* returns the same nodes in document order, with duplicates eliminated based on node identity.

Abgekürzte Notation — Vorsicht!

Was ist der Unterschied zwischen

- `/descendant::para[1]` und
- `//para[1]` ⇒ `/descendant-or-self::node()/para[1]` ?

5.3.3 Stringwert

Stringwerte

Für jeden Knoten ist der *Stringwert* (*String-Value*) definiert:

Textknoten Text (Zeichendaten, immer min. ein Zeichen).

Attributknoten normalisierter Attributwert.

Elementknoten Konkatenation der Stringwerte aller Textknoten-*Descendants*, in *Dokumentreihenfolge*.

Dokumentknoten Konkatenation der Stringwerte aller Textknoten-*Descendants*, in *Dokumentreihenfolge*.

Namespace-Knoten der Namespace-URI.

Processing-Instruction-Knoten der Text nach dem *Target*.

Kommentarknoten Kommentartext.

Sind Typinformationen aus einem *XML Schema* verfügbar, werden diese bei der Normalisierung der Stringwerte berücksichtigt (*Schema Normalized Values*).

5.4 Ausdrücke

Ausdrücke und Funktionen

- Operatoren
- Literale
- Konstruktoren
- Sequenzausdrücke, Bereichsausdrücke, Filterausdrücke
- Arithmetische Ausdrücke, Vergleiche
- `for`-Ausdrücke, Bedingte Ausdrücke, Quantorenausdrücke
- ...

Operatoren

Präzedenz	Operator	Assoziativität
1	`,` (comma)	left-to-right
3	`for`, `some`, `every`, `if`	left-to-right
4	`or`	left-to-right
5	`and`	left-to-right
6	`eq`, `ne`, `lt`, `le`, `gt`, `ge`, `=`, `!=`, `<`, `<=`, `>`, `>=`, `is`, `<<`, `>>`	left-to-right
7	`to`	left-to-right
8	`+`, `-`	left-to-right
9	`*`, `div`, `idiv`, `mod`	left-to-right
10	`union`, |	left-to-right
11	`intersect`, `except`	left-to-right
12	`instance of`	left-to-right
13	`treat`	left-to-right
14	`castable`	left-to-right
15	`cast`	left-to-right
16	`-`(unary), `+`(unary)	right-to-left
17	`?`, `*`(OccurrenceIndicator), `+`(OccurrenceIndicator)	left-to-right
18	`/`, `//`	left-to-right
19	`[ ]`, `( )`, `{}`	left-to-right

Literale

- `"12.5"` ist ein String.
- `"He said, ""I don't like it."""` ist ein String, der u. a. zwei Anführungszeichen und ein Apostroph enthält.
- `12` ist ein `xs:integer`.

- `12.5` ist ein `xs:decimal`.
- `125E2` ist ein `xs:double`.
- Die Werte des Typs `xs:boolean` können durch die Funktionsaufrufe `fn:true()` und `fn:false()` repräsentiert werden.

Konstruktoren

Wie gibt man Werte atomarer Typen an, die keine eigenen Literale haben?
Beispiele 5.9 (Konstruktoren).

```
xs:integer("12")
```

```
xdt:dayTimeDuration("PT5H")
```

```
xs:date("2001-08-25")
```

```
xs:float("NaN")
```

Sequenzausdrücke (Sequences)

Beispiele 5.10.

```
(10, 1, 2, 3, 4)
```

```
(10, (1, 2), (), (3, 4))
```

Beide Ausdrücke liefern die Sequenz 10, 1, 2, 3, 4.

⇒ Sequenzen sind immer „flach“.

Beispiel 5.11. `(salary, bonus)` ⇒ `(child::salary, child::bonus)`

Sequenz aller `salary`-Kinder gefolgt von den `bonus`-Kindern.

Beispiel 5.12.

```
($price, $price)
```

Wenn `$price` z. B. den Wert `10.50` hat, ist das Resultat die Sequenz `(10.50, 10.50)`.

Bereichsausdrücke (Range Expressions)

Beispiele 5.13.

```
(10, 1 to 4)
```

liefert die Sequenz 10, 1, 2, 3, 4.

```
10 to 10
```

liefert die einelementige Sequenz 10.

```
15 to 10
```

liefert die leere Sequenz `()`.

```
fn:reverse(10 to 15)
```

⇒ 15, 14, 13, 12, 11, 10.

Filterausdrücke

Analog zu *Prädikaten* in Pfadausdrücken
Beispiel 5.14.

```
$products[price gt 100]
```

Beispiele 5.15. `(1 to 100)[. mod 5 eq 0]`

⇒ 5, 10, 15, . . . , 95, 100

```
(21 to 29)[5]
```

⇒ 25

Sequenzoperationen

Beispiel 5.16. `A`, `B` und `C` stehen symbolisch für drei Elementknoten. Folgende Variablenbindungen seien im Kontext vorhanden:

```
$seq1 = (A, B)
$seq2 = (A, B)
$seq3 = (B, C)
```

Dann:

`$seq1 union $seq2` ⇒ `(A, B)`

`$seq2 union $seq3` ⇒ `(A, B, C)`

`$seq1 intersect $seq2` ⇒ `(A, B)`

`$seq2 intersect $seq3` ⇒ `B`

`$seq1 except $seq2` ⇒ `()`

`$seq2 except $seq3` ⇒ `A`

Arithmetische Ausdrücke

Beispiele 5.17.

```
2 + 1
```

```
$a - 1
```

```
-3 div 2
```

```
-3 idiv 2
```

Die arithmetischen Operatoren sind überladen.
Beispiel 5.18.

```
$emp/hiredate - $emp/birthdate
```

Die Subtraktion zweier `xs:date`-Werte liefert einen `xdt:dayTimeDuration`-Wert.

(Operatoren können überladen sein, Funktionen nicht.)

Vorsicht bei Leerzeichen und Minus

`foo- foo`	*Syntaxfehler*: `foo-` ist ein `QName`.
`foo -foo`	ist äquivalent zu `foo - foo`
`foo(: comment :)- foo`	ist äquivalent zu `foo - foo`
`foo-foo`	ist ein `QName`, keine Subtraktion.
`10div 3`	*Syntaxfehler*
`10 div3`	*Syntaxfehler*
`10div3`	*Syntaxfehler*

Wert-Vergleiche

Die (überladenen) Vergleichsoperatoren *für Werte* sind `eq`, `ne`, `lt`, `le`, `gt` und `ge`.

Diese vergleichen *einzelne Werte* (einelementige Sequenzen).
Beispiel 5.19.

```
$book1/author eq "Kennedy"
```

- Falls `$book1/author` genau einen *atomaren Wert* liefert, ist das Ergebnis
  - `true`, falls der Wert `"Kennedy"` ist,
  - `false`, sonst
- Falls `$book1/author` die leere Sequenz liefert, ist das Ergebnis die leere Sequenz `()`.
- Falls `$book1/author` mehr als einen Wert liefert, ist das ein *Fehler*.

Wert-Vergleiche

Beispiel 5.20.

```
/descendant::product[weight gt 100]
```

(`product`-Elemente, die kein `weight`-Unterelement haben, werden nicht selektiert.)

Allgemeine Vergleiche

Die Vergleichsoperatoren *für Sequenzen* sind `=`, `!=`, `<`, `<=`, `>` und `>=`.

Diese Vergleichsoperatoren sind *existenzquantifiziert.*
Beispiel 5.21.

```
$book1/author = "Kennedy"
```

- `true`, falls `$book1/author` eine Sequenz ist, die (als Stringwert) den Wert `"Kennedy"` enthält.
- `false`, sonst

Beispiel 5.22. `(1, 2) = (2, 3)` ⇒ true

`(2, 3) = (3, 4)` ⇒ true

`(1, 2) = (3, 4)` ⇒ false (`=` ist *nicht transitiv.*)

`(1, 2) != (2, 3)` ⇒ true (`=` und `!=` sind *nicht komplementär.*)

Knotenvergleiche

Die Vergleichsoperatoren *für Knoten* sind `is`, `<<` und `>>`.

Der Vergleich prüft die *Objektidentität* (`is`) bzw. die *Dokumentreihenfolge* (`<<` und `>>`).
Beispiele 5.23.

```
/books/book[isbn="1558604820"]
              is /books/book[call="QA76.9 C3845"]
```

```
/transactions/purchase[parcel="28-451"]
           << /transactions/sale[parcel="33-870"]
```

Logische Ausdrücke

Operatoren: `and`, `or`

Konjunktion und Disjunktion können in XPath 2.0 *strikt* oder *nicht strikt* ausgewertet werden (implementierungsabhängig).

(In XPath 1.0 immer strikte Auswertung)
Beispiele 5.24.

```
1 eq 1 and 2 eq 2
```

```
1 eq 1 or 2 eq 3 div 0
```

Die Negation gibt es als Funktion `fn:not()`.
Beispiel 5.25 (Negation).

```
fn:not((1, 2) = (2, 3))
```

`for`-Ausdrücke

Beispiel 5.26.

```
for $i in (0 to 3) return 2 * $i + 1
```

⇒ `1, 3, 5, 7`
Beispiel 5.27 (verschachtelte Schleifen).

```
for $i in (10, 20), $j in (1, 2) return ($i + $j)
```

⇒ `11, 12, 21, 22`

```
for $x in X, $y in Y return $x + $y
```

wird expandiert zu

```
for $x in X return for $y in Y return $x + $y
```

Beispiel 5.28 (mit Variablenabhängigkeit).

```
for $x in $z, $y in f($x) return g($x, $y)
```

`for`-Ausdrücke

Der `return`-Ausdruck hat denselben Fokus wie der gesamte `for`-Ausdruck.
Summe über Bestellung bilden:

Beispiel (falsch)

```
fn:sum(for $i in order-item return @price * @qty)
```

Beispiel 5.29 (richtig).

```
fn:sum(for $i in order-item return $i/@price * $i/@qty)
```

Bedingte Ausdrücke

Beispiele 5.30.

```
if ($widget1/unit-cost < $widget2/unit-cost)
        then $widget1
        else $widget2
```

```
if ($part/@discounted)
        then $part/wholesale
        else $part/retail
```

Quantorenausdrücke

Beispiele 5.31.

```
every $part in /parts/part satisfies $part/@discounted
```

```
some $emp in /emps/employee satisfies
     ($emp/bonus > 0.25 * $emp/salary)
```

```
some $x in (1, 2, 3), $y in (2, 3, 4)
     satisfies $x + $y = 4
```

⇒ true

```
every $x in (1, 2, 3), $y in (2, 3, 4)
     satisfies $x + $y = 4
```

⇒ false

Ausdrücke mit Sequenztypen

Beispiel 5.32.

```
5 instance of xs:integer
```

⇒ `true`

```
(5, 6) instance of xs:integer+
```

⇒ `true`

```
5 instance of xs:integer+
```

⇒ `true`

Funktionsaufrufe

```
my:three-argument-function(1, 2, 3)

my:two-argument-function((1, 2), 3)

my:two-argument-function(1, ())

my:one-argument-function((1, 2, 3))

my:one-argument-function(())

my:zero-argument-function( )
```

Typhierarchie

Grafische Darstellung der Xpath-/XQuery-Typhierarchie[93]

5.5 Funktionsbibliothek

Funktionsbibliothek

XQuery 1.0 and XPath 2.0 Functions and Operators

- Accessor-Funktionen
- Konstruktoren
- Numerische Funktionen
- String-Funktionen
- URI-Funktionen
- Boolesche Funktionen
- Datums- und Zeitfunktionen
- Funktionen für QNames
- Sequenzfunktionen
- Kontextfunktionen
- ...

[93] http://www.w3.org/TR/xquery-operators/#datatypes

„Accessor“-Funktionen

- `fn:node-name($arg as node()?) as xs:QName?`
- `fn:string() as xs:string` Stringwert des *Context Item*
- `fn:string($arg as item()?) as xs:string` Stringwert des Arguments
- `fn:base-uri($arg as node()?) as xs:anyURI?`
- `fn:document-uri($arg as node()?) as xs:anyURI?`
- ...

Konstruktorfunktionen

Beispiele 5.33.

```
xs:string($arg as xdt:anyAtomicType?) as xs:string?
xs:decimal($arg as xdt:anyAtomicType?) as xs:decimal?
xs:integer($arg as xdt:anyAtomicType?) as xs:integer?
xs:normalizedString($arg as xdt:anyAtomicType?)
                                as xs:normalizedString
xs:dateTime($arg as xs:anyAtomicType?) as xs:dateTime?

fn:dateTime($arg1 as xs:date?, $arg2 as xs:time?) as
                                          xs:dateTime
...
```

Anwendung auf leere Sequenz liefert die leere Sequenz.
Beispiel 5.34.

```
fn:dateTime(xs:date("1999-12-31"), xs:time("12:00:00"))
```

Numerische Funktionen

Operatorsemantik

```
op:numeric-add($arg1 as numeric, $arg2 as numeric)
                                        as numeric
```

Weitere Funktionen

`fn:abs`, `fn:ceiling`, `fn:floor`, `fn:round`, `fn:round-half-to-even`, ...

String-Funktionen: Unicode

```
fn:codepoints-to-string($arg as xs:integer*)
                                as xs:string

fn:string-to-codepoints($arg as xs:string?)
                              as xs:integer*
```

Beispiel 5.35.

```
fn:string-to-codepoints("Thérèse")
```

⇒ (84, 104, 233, 114, 232, 115, 101)

String-Funktionen: Vergleiche

Collations

Eine *Collation* spezifiziert, wie Strings verglichen und damit sortiert werden.

```
fn:compare($comparand1 as xs:string?,
           $comparand2 as xs:string?) as xs:integer?

fn:compare($comparand1 as xs:string?,
           $comparand2 as xs:string?,
           $collation  as xs:string) as xs:integer?
```

`fn:compare` liefert -1 (<), 0 (=), or 1 (>).

String-Funktionen: Vergleiche

Beispiele 5.36 (Collation).

```
fn:compare('Strasse', 'Straße', 'deutsch')
```

⇒ 0

```
fn:compare('Strassen', 'Straße')
```

⇒ 1 (wenn *Default Collation* deutsch)

`fn:compare` definiert auch die Semantik der Operatoren `eq`, `ne`, `lt`, `le`, `gt` und `ge`.

String-Funktionen

`fn:concat`, `fn:string-join`, `fn:substring`, `fn:string-length`, `fn:normalize-space`, `fn:normalize-unicode`, `fn:upper-case`, `fn:lower-case`, `fn:translate`, `fn:encode-for-uri`, `fn:iri-to-uri`, `fn:escape-html-uri`
Beispiele 5.37.

```
fn:concat('un', 'grateful')

⇒ "ungrateful"

fn:string-join(('a', 'b', 'cd'), '; ')

⇒ "a; b; cd"

fn:substring("motor car", 6)

⇒ " car"
```

String-Funktionen: Matching

`fn:contains`, `fn:starts-with`, `fn:ends-with`, `fn:substring-before`, `fn:substring-after`
Beispiele 5.38.

```
fn:contains ("tattoo", "ttt")

 ⇒ false

fn:starts-with("tattoo", "tat")

 ⇒ true

fn:substring-before ("tattoo", "attoo")

 ⇒ "t"
```

String-Funktionen: Pattern Matching

`fn:matches`, `fn:replace`, `fn:tokenize`

verwenden *reguläre Ausdrücke*
Beispiele 5.39.

```
fn:matches("abracadabra", "bra")

 ⇒ true
```

```
fn:matches("abracadabra", "^bra")

 ⇒ false

fn:replace("abracadabra", "bra", "*")

 ⇒ "a*cada*"

fn:tokenize("abracadabra", "(ab)|(a)")

⇒ ("", "r", "c", "d", "r", "")
```

URI-Funktionen

```
fn:resolve-uri($relative as xs:string?) as xs:anyURI?

fn:resolve-uri($relative as xs:string?,
                    $base as xs:string) as xs:anyURI?
```

Beispiel 5.40.

```
fn:resolve-uri("../impressum.html",
               "http://foo.com/Images/42.png")

⇒ http://foo.com/impressum.html
```

Beispiel 5.41.

```
fn:resolve-uri("foo.html")

⇒ file:/tmp/foo.html
```

Datums- und Zeitfunktionen

Beispiele 5.42.

```
op:time-greater-than(xs:time("08:00:00+09:00"),
                     xs:time("17:00:00-06:00"))

⇒ false

fn:years-from-duration(
                    xdt:yearMonthDuration("P20Y15M"))

⇒ 21

fn:timezone-from-date(xs:date("1999-05-31-05:00"))

⇒ -PT5H

fn:adjust-dateTime-to-timezone(
                  xs:dateTime("2002-03-07T10:00:00"))
```

⇒ `2002-03-07T10:00:00-05:00`, wenn die implizite Zeitzone `-05:00` (`-PT5H0M`) ist.

Funktionsbibliothek

Funktionen für QNames

`fn:resolve-QName`, `fn:QName`, `op:QName-equal`, `fn:prefix-from-QName`, `fn:local-name-from-QName`, `fn:namespace-uri-from-QName`, `fn:namespace-uri-for-prefix`, `fn:in-scope-prefixes`

Funktionen für Knoten

`fn:name`, `fn:local-name`, `fn:namespace-uri`, `fn:number`, `fn:lang`, `op:is-same-node`, `op:node-before` `op:node-after` `fn:root`

Funktionen für Sequenzen

`fn:boolean`, `op:concatenate`, `fn:index-of`, `fn:empty`, `fn:exists`, `fn:distinct-values`, `fn:insert-before`, `fn:remove`, `fn:reverse`, `fn:subsequence`, `fn:unordered`, `fn:deep-equal`
Beispiele 5.43.

```
fn:boolean((1,2,3))
```

⇒ `true`

```
fn:boolean(())
```

⇒ `false`

```
fn:index-of(("abc", "ab", "b", "ab", "c"), "ab")
```

⇒ `2, 4`

```
fn:remove((41, 42, 43), 2)
```

⇒ `41, 43`

Aggregationsfunktionen

`fn:count`, `fn:avg`, `fn:max`, `fn:min`, `fn:sum`
Beispiele 5.44.

```
fn:max((3,4,5)), fn:min((3,4,5))
```

⇒ `5, 3`

```
fn:sum((3,4,5))
```

⇒ `12`

Sequenzgenerierende Funktionen

`op:to` `fn:id`, `fn:idref`, `fn:doc`, `fn:doc-available`, `fn:collection`
Beispiel 5.45.

```
1 to 3
```

⇒ `(1, 2, 3)`

Kontext-Funktionen

`fn:position`, `fn:last`, `fn:current-dateTime`, `fn:current-date`, `fn:current-time`, `fn:implicit-timezo`
`fn:default-collation`, `fn:static-base-uri`
Beispiel 5.46.

```
fn:current-dateTime()
```

⇒ `2016-07-05T10:29:45.6+02:00`

Erweiterungen

Anwendungsdefinierte Funktionen

Definiert durch die Anwendung, in die XPath eingebettet ist, z. B. XSLT.

Benutzerdefinierte Funktionen

Z. B. in XQuery und XSLT

Zusammenfassung und Ausblick

XML Path Language (XPath) 1.0

- XPath ist eine einfache XML-Anfragesprache
- kompakte Syntax, einfaches Datenmodell
- Ausdrücke liefern Knotensequenzen

XML Path Language (XPath) 2.0

- Starke Typisierung basierend auf *XML Schema*
- Viele neue Datentypen und Funktionen
- Neue Ausdrücke: `for`, `if`, Quantoren, ...

- Pattern Matching mit regulären Ausdrücken
- Teilmenge von *XQuery* 1.0
- Kompatibilitätsmodus zu XPath 1.0

XML Path Language (XPath) 3.0

Funktionen als First-Order Datentyp
Beispiel 5.47 (Inline Function Expressions).

```
let $f := function($i as xs:integer, $j as xs:integer)
            as xs:integer { $i * $j }
```

Beispiel 5.48 (Dynamic Function Call).

```
$f(2, 3)
```

Beispiel 5.49 (Currying).

```
let $g := $f(23, ?)
return $g(42)
```

Higher-Order Functions: `fn:map`, `fn:filter`, `fn:fold-left`, `fn:fold-right`, `fn:map-pairs`

Beispiel 5.50 (`fn:map`).

```
fn:map(fn:string-length#1, ("XML", "XPath", "ok"))
```

ergibt Sequenz (3, 5, 2)
Beispiel 5.51 (`fn:fold-left`).

```
fn:fold-left(function($a, $b) { $a * $b }, 1, (2,3,5,7))
```

ergibt 210, denn $210 = (((1 \cdot 2) \cdot 3) \cdot 5) \cdot 7$
Beispiel 5.52 (`fn:map-pairs`).

```
fn:map-pairs(fn:concat#2, ("a","b","c"), ("x","y","z"))
```

ergibt `('ax', 'by', 'cz')`

Simple Map Operator `(E1 ! E2)` (`E1`, `E2`: sequences of items)
Beispiel 5.53.

```
child::div1 / child::para / string() ! concat("id-", .)
```

Beispiel 5.54.

```
$emp ! (@first, @middle, @last)
```

Returns the values of the attributes `first`, `middle`, and `last` for element `$emp`, in the order given. (The `/` operator here returns the attributes in an unpredictable order.)

Beispiel 5.55.

```
avg( //employee / salary ! translate(., '$', '')
     ! number(.))
```

Returns the average salary of the employees, having converted the salary to a number by removing any `$` sign and then converting to a number. (The second occurrence of `!` could not be written as `/` because the left-hand operand of `/` cannot be an atomic value.)

6 XSL Transformations (XSLT)

6.1 Einleitung

Vorüberlegungen

Grundidee: Inhalte und Formatierung trennen!

Separation of Concerns

Szenario

1. Eingabe: Klasse strukturierter XML Dokumente (z. B. durch DTD oder XML Schema definiert)
2. Umformung in ein Präsentationsformat, für Ausgabemedien wie Web-Browser (HTML) oder Papier (z. B. PDF).
3. Diese Umformung wird in einem *Stylesheet* formalisiert.

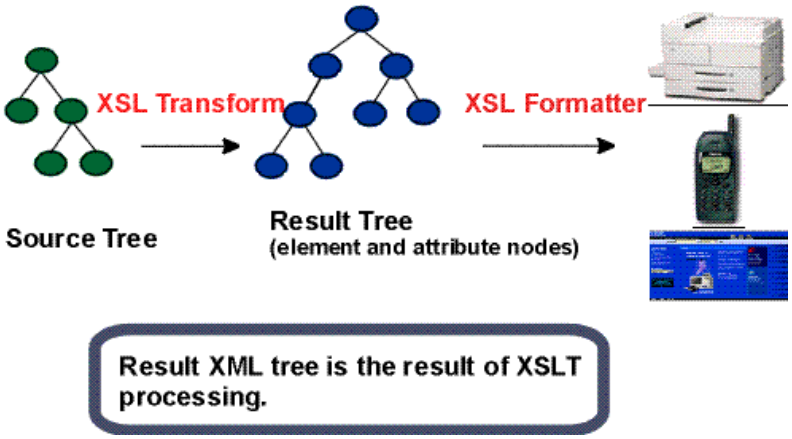
Stylesheets

Anforderungen an eine Stylesheet-Sprache für XML:

- leicht zu erlernen, einfache Syntax,
- beliebige Formatierungen für beliebige Elemente,
- komplette Baumstruktur des Dokuments zugreifbar,
- Turing-vollständig (berechnungsuniversell),
- international: Unicode, Schreibweisen links → rechts, rechts → links, unten → oben,
- mächtiges Rendering-Modell für professionelles Layout.

Stylesheet-Sprachen (z. B. für SGML, XML, HTML) unterscheiden sich stark bzgl. Mächtigkeit und Paradigmen.

Komponenten der XSL-Sprachfamilie



Komponenten der XSL-Sprachfamilie

XML Path Language (XPath) Sprache zur Adressierung von Dokumentteilen, verwendet u. a. in XSLT und XPointer

XSL Transformations (XSLT) Erste Phase (*Transformation*): Konstruktion eines *Ergebnisbaums* aus dem *Eingabebaum*

Extensible Stylesheet Language (XSL) Zweite Phase (*Formatierung*): Interpretation des Ergebnisbaums als Textsatz-Anweisungen (*Formatting Objects*) und entsprechende Aufbereitung für das Ausgabemedium.

XSLT und Formatting Objects lassen sich auch einzeln sinnvoll einsetzen.

Hier: *XSLT als* ***Programmiersprache*** *für XML.*

Spezifikationen

- **XSL Transformations (XSLT) Version 1.0** W3C Recommendation 16. November 1999
- ***XSL Transformations (XSLT) Version 2.0***[94] W3C Recommendation 23. Januar 2007
- **XSL Transformations (XSLT) Version 3.0** W3C Recommendation 8. Juni 2017
- **Extensible Stylesheet Language (XSL) Version 1.1** W3C Recommendation 5. Dezember 2006

[94] http://www.w3.org/TR/xslt20/

XSL Transformations (XSLT) — Übersicht

- Eine *XSLT-Transformation* wird durch ein XML-Dokument beschrieben, das *Stylesheet*.
- Die Transformation überführt normalerweise einen *Eingabebaum* in einen *Ausgabebaum*. (Allgemein: null oder mehr Eingabe- in einen oder mehrere Ausgabebäume)
- Der Ausgabebaum kann eine völlig andere Struktur und ein anderes Vokabular als der Eingabebaum haben.
- Die Transformation wird durch Regeln (*Templates*) beschrieben, die *Patterns* auf *Sequenzen* abbilden.
- Regelauswahl durch *Prioritätsalgorithmus*
- XSLT ist erweiterbar (namespace-basiert).

6.2 Stylesheets

Ein erstes Beispiel

Beispiel 6.1 (Eingabebaum).

```
<slideshow>
  <title>This is a title</title>
  <slides>

    <slide xml:lang="en-US">
      <title>This is slide 1</title>
      <point>Point 1</point>
      <point>Point 2</point>
      <point>Point 3</point>
      <point>Point 4</point>
    </slide>

    <slide>
      <title>This is slide 2</title>
      ...
    </slide>
  </slides>
</slideshow>
```

Beispiel 6.2 (Gewünschter XHTML-Ausgabebaum).

```
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>This is a title</title>
  </head>
  <body>
    <h1>This is a title</h1>
    <h2>This is slide 1</h2>
    <ul>
```

```
      <li>Point 1</li>
      <li>Point 2</li>
      <li>Point 3</li>
      <li>Point 4</li>
    </ul>
    <h2>This is slide 2</h2>
    <ul/>
  </body>
</html>
```

Beispiel 6.3 (XSLT-Stylesheet).

```
<xsl:stylesheet version="2.0"
     xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
     xmlns="http://www.w3.org/1999/xhtml">
  <xsl:output method="xhtml" indent="yes"/>

  <xsl:template match="slideshow">
    <html>
      <head>
        <title>
          <xsl:apply-templates select="title"/>
        </title>
      </head>
      <body>
        <h1><xsl:apply-templates select="title"/></h1>
        <xsl:apply-templates select="slides"/>
      </body>
    </html>
  </xsl:template>

  <xsl:template match="slide">
    <h2><xsl:apply-templates select="title"/></h2>
    <ul>
      <xsl:apply-templates select="point"/>
    </ul>
  </xsl:template>

  <xsl:template match="point">
    <li><xsl:apply-templates/></li>
  </xsl:template>

</xsl:stylesheet>
```

Top-Level Elemente

Beispiel 6.4 (`xsl:stylesheet` mit Top-Level Elementen).

```
<xsl:stylesheet version="2.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:my="http://example.com/functions">

  <xsl:import href="lib-tables.xsl"/>
```

```
  <xsl:variable name="c" select="42"/>

  <xsl:function name="my:f">
    <xsl:param name="x" as="xs:integer"/>
    <xsl:sequence select="$c * $x"/>
  </xsl:function>

  <xsl:template match="message">
    <p><xsl:apply-templates select="child::node()"/></p>
  </xsl:template>

  <xsl:output method="html"/>
</xsl:stylesheet>
```

Struktur der Stylesheets

Definition: Stylesheet Element

```
<xsl:stylesheet
  id? = id
  extension-element-prefixes? = tokens
  exclude-result-prefixes? = tokens
  version = number
  xpath-default-namespace? = uri
  default-validation? = "preserve" | "strip"
  default-collation? = uri-list
  input-type-annotations? =
                "preserve" | "strip" | "unspecified">
  <!-- Content: (xsl:import*, other-declarations) -->
</xsl:stylesheet>
```

(Spezielle Notation zur Elementdefinition)

`xsl:transform` kann alternativ und gleichbedeutend zu `xsl:stylesheet` verwendet werden.

Top-Level Elemente

Kinder des `xsl:stylesheet`-Elements

`xsl:import` (vor allen anderen Kindern)

`xsl:include`, `xsl:attribute-set`, `xsl:character-map`, `xsl:decimal-format`, `xsl:function`, `xsl:import-schema`, `xsl:key`, `xsl:namespace-alias`, `xsl:output`, `xsl:param`, `xsl:preserve-space`, `xsl:strip-space`, `xsl:template`, `xsl:variable`, user-defined data elements

Diese Elemente aus dem XSLT-Namespace heißen *Deklarationen*.

Struktur der Stylesheets

- Die Reihenfolge der Deklarations-Elemente hat auf die Verarbeitung keinen Einfluss, außer bei Importen mit `xsl:import` und beim Error-Recovery.

Erweiterbarkeit

- Weitere Nicht-XSLT-Elemente sind zulässig, wenn sie einen nicht-XSLT Namespace besitzen.

  Beispiele:

  - Daten, Tabellen
  - Information über die Weiterverarbeitung des Ausgabebaums
  - Informationen, wo der Eingabebaum zu finden ist
  - Metadaten zum Stylesheet
  - Dokumentation (z. B. des Stylesheets)

- Nicht-XSLT-Attribute in XSLT-Elementen sind zulässig, wenn sie einen nicht-XSLT Namespace besitzen.

Stylesheets kombinieren

- `xsl:include` kombiniert Stylesheets auf der *Baumebene*.

  Achtung: Mehrfaches Einfügen desselben Stylesheets führt zu doppelten Definitionen ⇒ Fehler.

- `xsl:import` erlaubt auch das *Überschreiben* von Stylesheets.

  Dazu später mehr ...

Datenmodell

Das XSLT-2.0-Datenmodell entspricht im wesentlichen dem XPath-2.0-Datenmodell.

Erweiterungen

- XML Versions
- Stripping Whitespace from the Stylesheet
- Stripping Type Annotations from a Source Tree
- Stripping Whitespace from a Source Tree
- Attribute Types and DTD Validation
- ...

Transformationsregeln (Template Rules)

Beispiel 6.5 (Template Rule).

```
<xsl:template match="/">
  <html>
    <head>
      <title>Expense Report Summary</title>
    </head>
    <body>
      <p>Total Amount:
        <xsl:value-of select="expense-report/total"/>
      </p>
    </body>
  </html>
</xsl:template>
```

Pattern → *Sequence Constructor* → *XSLT-Instruktionen*

Template Rules

Definition: Template Rule

```
<!-- Category: declaration -->
<xsl:template
  match? = pattern
  name? = qname
  priority? = number
  mode? = tokens
  as? = sequence-type>
  <!-- Content: (xsl:param*, sequence-constructor) -->
</xsl:template>
```

- Das `match`-Attribut oder das `name`-Attribut muss vorhanden sein.
- Der Elementinhalt heißt *Sequence Constructor*.
- Daraus generiert werden Knoten und/oder atomare Werte (also *Items*)

6.3 Sequenzkonstruktoren

Template Rules: Sequence Constructor

Der *Sequence Constructor* ist eine Knotensequenz (im Stylesheet) und wird ausgewertet zu einer Ergebnissequenz von Items (Knoten und/oder atomaren Werten).

Knotentypen im Sequence Constructor

Textknoten werden in die Ergebnissequenz kopiert.

Literal Result Elements werden zu Elementknoten gleichen Namens ausgewertet.

XSLT-Instruktionen liefern eine Sequenz.

Extension Instructions liefern ebenfalls eine Sequenz.

Die von diesen Knoten gelieferten Teilsequenzen werden zur Ergebnissequenz konkateniert.
Beispiel 6.6 (Sequence Constructor).

```
<xsl:template match="list">
  <p>List:</p>
  <ul class="{@class}"><xsl:apply-templates/></ul>
</xsl:template>
```

Textknoten im Sequence Constructor

Bestimmte Whitespace-Textknoten werden ignoriert.

Die verbleibenden Textknoten werden in die Ergebnissequenz kopiert.
Beispiel 6.7 (Textknoten im Sequence Constructor).

```
<xsl:template match="product/price">
  Der Preis beträgt <xsl:value-of select="."/> EUR.
</xsl:template>
```

Beispiel 6.8 (Ergebnis).

```
  Der Preis beträgt [...] EUR.
```

Literal Result Elements

Element, das weder zum XSLT-Namespace gehört noch Erweiterungselement ist
Beispiel 6.9.

```
<xsl:template match="product/price">
  <em class="sale">Nur <xsl:value-of select="."/> EUR.</em>
</xsl:template>
```

Auswertung

- Wird zu neuem Elementknoten ausgewertet.
- Der Elementinhalt wird wiederum als Sequence Constructor behandelt.
- Nicht-XSLT-Attribute werden übernommen, Auswertung als *Attributwert-Template*.
- Weitere Attribute können mit `xsl:attribute` und `xsl:use-attribute-sets` erzeugt werden.

XSLT-Instruktionen

- Erzeugen neuer Knoten:

  `xsl:document`, `xsl:element`, `xsl:attribute`, `xsl:processing-instruction`, `xsl:comment`, `xsl:value-of`, `xsl:text`, `xsl:namespace`

- Beliebige Sequenz berechnen: `xsl:sequence`
- Bedingte und mehrfache Auswertung: `xsl:if`, `xsl:choose`, `xsl:for-each`, `xsl:for-each-group`
- Aufruf von Templates: `xsl:apply-templates`, `xsl:apply-imports`, `xsl:call-template`, `xsl:next-match`
- Variablendeklarationen: `xsl:variable`, `xsl:param`
- Weitere Instruktionen: `xsl:number`, `xsl:analyze-string`, `xsl:message`, `xsl:result-document`

`apply-templates`

Definition: Applying Template Rules

```
<!-- Category: instruction -->
<xsl:apply-templates
  select? = expression
  mode? = token>
  <!-- Content: (xsl:sort | xsl:with-param)* -->
</xsl:apply-templates>
```

Knoten auswählen, Templates darauf anwenden
Beispiel 6.10.

```
<xsl:template match="message">
  <p><xsl:apply-templates select="child::node()"/></p>
</xsl:template>
```

- Der XPath-Ausdruck im `select`-Attribut (default: `child::node()`) liefert eine Sequenz, die ggf. noch sortiert wird (Unterelemente `xsl:sort`).
- Zu jedem Knoten wird dann die „am besten passende“ Template-Regel ermittelt und ausgewertet.
- Die Konkatenation der resultierenden Sequenzen ist das Ergebnis der `apply-templates`-Instruktion.

Dynamischer Kontext

Der dynamische Kontext bei der Auswertung einer Template-Regel für den i-ten Knoten K_i der sortierten Eingabesequenz ist:

- *Context Item*: K_i
- Kontextposition: i
- Kontextgröße: Länge der sortierten Eingabesequenz

apply-templates

Beispiel 6.11 (Stylesheet).

```
<xsl:template match="message">
  <p><xsl:apply-templates select="child::node()"/></p>
</xsl:template>

<xsl:template match="emph">
  <strong><xsl:apply-templates/></strong>
</xsl:template>
```

Beispiel 6.12 (Eingabe).

```
<message>
  Proceed <emph>at once</emph> to the exit!
</message>
```

Beispiel 6.13 (Ausgabe).

```
<p>
  Proceed <strong>at once</strong> to the exit!
</p>
```

Built-In Template-Regeln (vereinfachte Version)

Falls keine Regel eines Stylesheets zu einem Knoten passt, werden eingebaute Mechanismen aktiviert, hier als Pseudo-Regeln dargestellt.

Dokument- und Elementknoten

```
<xsl:template match="*|/">
  <xsl:apply-templates/>
</xsl:template>
```

Text- und Attributknoten

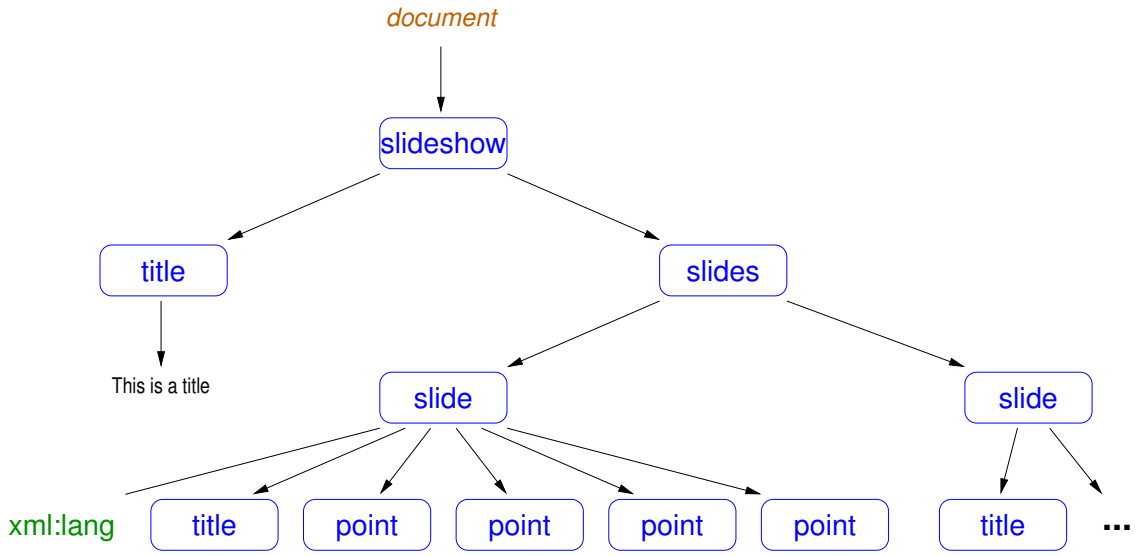
```
<xsl:template match="text()|@*">
  <xsl:value-of select="string(.)"/>
</xsl:template>
```

Processing Instructions und Kommentare ignorieren

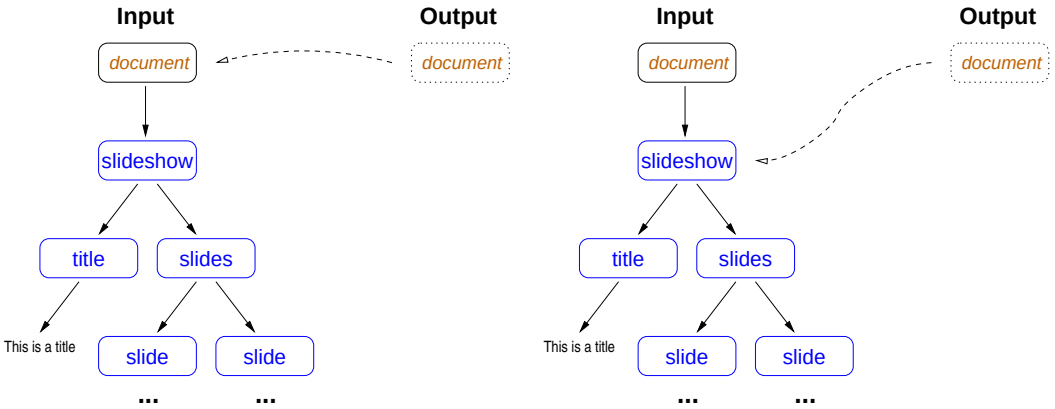
```
<xsl:template match="processing-instruction()|comment()"/>
```

Beispiel einer XSLT-Berechnung

Beispiel 6.14 (Eingabebaum).



Beispiel 6.15 (Erste Schritte).

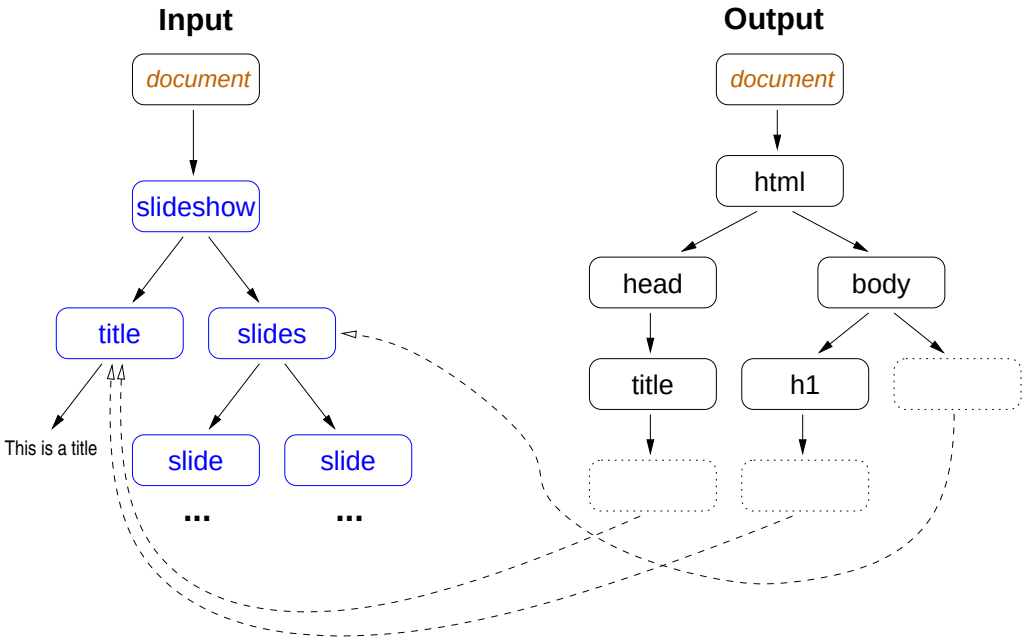


Beispiel 6.16 (Template-Regel).

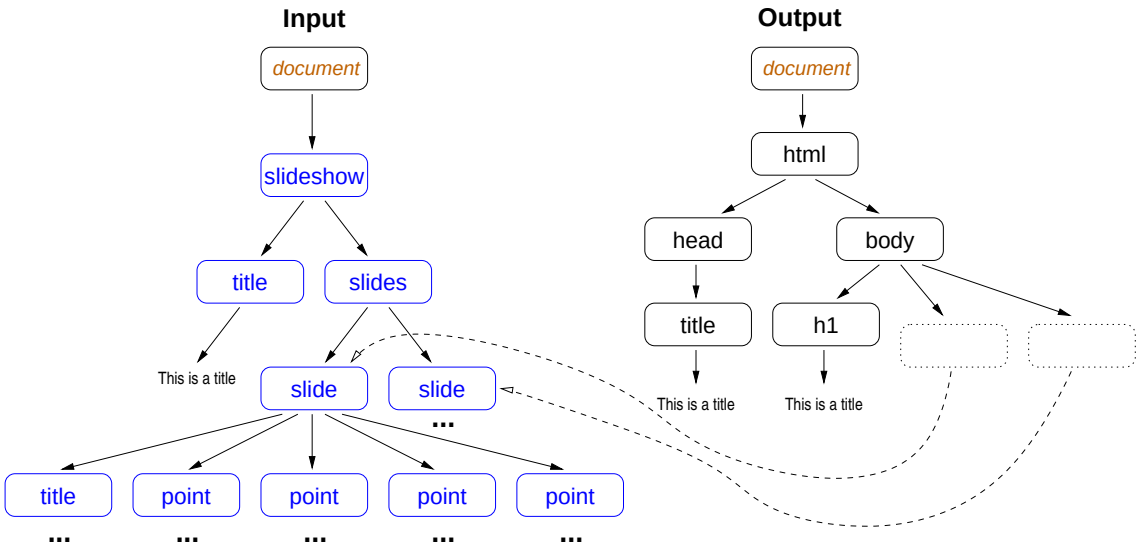
```
<xsl:template match="slideshow">
  <html>
    <head>
      <title>
```

```
        <xsl:apply-templates select="title"/>
      </title>
    </head>
    <body>
      <h1><xsl:apply-templates select="title"/></h1>
      <xsl:apply-templates select="slides"/>
    </body>
  </html>
</xsl:template>
```

Beispiel 6.17 (Template-Regel anwenden).



Beispiel 6.18 (Default-Regel: „Rekursive Verarbeitung“).



apply-templates

Beispiel 6.19 (Zugriff auf weiter entfernte Stellen im Baum).

```
<xsl:template match="employee">
  <fo:block>
    Employee <xsl:apply-templates select="name"/>
    belongs to group
    <xsl:apply-templates select="ancestor::department/group"/>
  </fo:block>
</xsl:template>
```

Beispiel 6.20 (einfache Gruppierung durch mehrere `apply-template`).

```
<xsl:template match="product">
  <table>
    <xsl:apply-templates select="sales/domestic"/>
  </table>
  <table>
    <xsl:apply-templates select="sales/foreign"/>
  </table>
</xsl:template>
```

`apply-templates` — Vorsicht!

Beispiel 6.21 (inneres `div`-Element wird doppelt verarbeitet).

```
<xsl:template match="doc">
  <xsl:apply-templates select="descendant::div"/>
</xsl:template>
<xsl:template match="div">
  <xsl:apply-templates/>
</xsl:template>


<doc>
  <div>
    <div>...</div>
  </div>
</doc>
```

Beispiel 6.22 (Endlosschleife!).

```
<xsl:template match="foo">
  <xsl:apply-templates select="."/>
</xsl:template>
```

Patterns

- Ein Pattern beschreibt *Bedingungen* an Knoten.
- Patterns sind eine Teilmenge der XPath-Ausdrücke.

Definition 6.23 (Pattern-Syntax, informell)**.**

- Menge von Pfadausdrücken, durch `|` verbunden
- Jeder Schritt verwendet nur die Achsen `child` oder `attribute`, oder den Operator `//`.
- Jeder Schritt kann beliebige Prädikate verwenden

Beispiele 6.24.

```
para
olist/item

section|subsection|subsubsection
@number | @text

subsection//para[1]
```

Patterns — Semantik

Definition 6.25 (Semantik von Patterns, Skizze)**.** Ein Pattern *passt* zu einen Knoten (*matches a node*) in einer gegebenen Umgebung, wenn der Knoten Element der durch einen *äquivalenten Ausdruck* beschriebenen Sequenz von Knoten ist (bzgl. eines „möglichen Kontexts“).

Beispiel 6.26. Pattern: `p`

Äquivalenter Ausdruck: `root(.)//(child-or-top::p)`

Patterns

Alternative Beschreibung des Matching (prädikativ)

Die Definition des Matching stützt sich auf die Auswertung von XPath-Ausdrücken.

In der Praxis ist es oft einfacher, die Patterns direkt als Bedingungen zu lesen — *von rechts nach links*.

Beispiel 6.27. `appendix//ulist/item[position()=1]`

Dieser Ausdruck passt zu einen Knoten gdw.

1. der Knoten ein `item`-Element ist,
2. der Ausdruck `position()=1` `true` ergibt, ausgewertet mit dem Knoten als Kontextknoten und den *Siblings*, die `item`-Elemente sind, als Kontextknotenliste, und
3. der *Parent* des Knotens zu `appendix//ulist` passt

Whitespace aus dem Stylesheet entfernen

1. Kommentare und Verarbeitungsanweisungen entfernen

2. Benachbarte Textknoten verschmelzen
3. Nur aus *Whitespace* bestehende Textknoten entfernen, wenn sie nicht Kind eines `xsl:text` Elements sind und nicht durch `xml:space="preserve"` geschützt sind.
4. Nur aus *Whitespace* bestehende Textknoten immer entfernen, wenn sie Kind eines der folgenden Elementknoten sind: `xsl:analyze-string`, `xsl:apply-imports`, `xsl:apply-templates`, `xsl:attribute-set`, `xsl:call-template`, `xsl:character-map`, `xsl:choose`, `xsl:next-match`, `xsl:stylesheet`, `xsl:transform`.
5. Nur aus *Whitespace* bestehende Textknoten entfernen, wenn der rechte Nachbar ein `xsl:param` oder `xsl:sort` Element ist.

Whitespace aus dem Eingabebaum entfernen

Auch in Eingabebäumen sind einige der nur aus *Whitespace* bestehenden Textknoten irrelevant.

Eine Teilmenge der Elementnamen wird dazu als *whitespace-preserving* klassifiziert, mittels `xml:space="preserve"` und der Deklarationen:

```
<xsl:strip-space
  elements = tokens />

<xsl:preserve-space
  elements = tokens />
```

Details zur Semantik: siehe XSLT-Spezifikation

Modes

- *mode*: Einfache Möglichkeit, je nach Situation unterschiedliche Regelmengen anzuwenden.
- (In Java: Objekt hat mehr als eine Methode.)

Definition 6.28 (Mode-Namen)**.**

- Es gibt einen *Default-Mode*, der keinen Namen hat.
- Jeder andere Mode hat einen *QName* als Namen.

Definition 6.29 (`mode`-Attribute bei `xsl:template`)**.**

- `mode`-Attribute fehlt: Die Regel ist im Default-Mode anwendbar.
- `mode`-Attribute enthält eine Liste aus *QName*s und/oder `#default`: Die Regel ist auf die Modes aus der Liste bzw. auf den Default-Mode anwendbar.
- `mode="#all"`: Die Regel ist auf alle Modes anwendbar.

Modes

Beispiel 6.30.

```
<xsl:template match="chapter">
  <h1><xsl:apply-templates select="title"/></h1>
  <xsl:apply-templates select="section|para"/>
</xsl:template>


<xsl:template match="chapter" mode="toc">
  <!-- generate table of contents entry -->
  ...
  <xsl:apply-templates select="section" mode="toc"/>
</xsl:template>


<xsl:template match="para">
  <p><xsl:apply-templates/></p>
</xsl:template>

<xsl:template match="para" mode="toc"/>
```

Modes

Definition 6.31 (`mode`-Attribute bei `xsl:apply-templates`).

- Während der Auswertung gibt es stets einen *Current-Mode*.
- `mode`-Attribute fehlt: Die Auswertung wird im Default-Mode durchgeführt. „Current-Mode `:=` Default-Mode“
- `mode`-Attribute enthält genau einen *QName*: Die Auswertung wird im angegebenen Mode durchgeführt. „Current-Mode `:=` QName“
- `mode="#default"`: Die Auswertung wird im Default-Mode durchgeführt. „Current-Mode `:=` Default-Mode“
- `mode="#current"`: Die Auswertung wird im Current-Mode durchgeführt. „Current-Mode bleibt unverändert.“

Built-In Template-Regeln (mit `mode`)

Falls keine Regel eines Stylesheets zu einem Knoten passt, werden eingebaute Mechanismen aktiviert, hier als Pseudo-Regeln dargestellt.

Dokument- und Elementknoten (approximiert)

```
<xsl:template match="*|/" mode="#all">
  <xsl:apply-templates mode="#current"/>
</xsl:template>
```

(Alle beim Aufruf mitgegebenen Parameter werden ebenfalls weitergereicht.)

Text- und Attributknoten

```
<xsl:template match="text()|@*" mode="#all">
  <xsl:value-of select="string(.)"/>
</xsl:template>
```

Processing Instructions und Kommentare ignorieren

```
<xsl:template match="processing-instruction()|comment()" mode="#all"/>
```

Stylesheets importieren

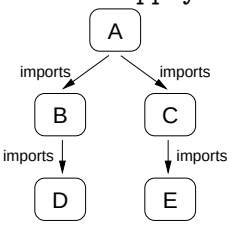
Beispiel 6.32.

```
<xsl:stylesheet version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:import href="article.xsl"/>
  <xsl:import href="bigfont.xsl"/>

  <xsl:template match="/">...</xsl:template>
</xsl:stylesheet>
```

Importierte Regeln sind nachrangig

Precedence („Vorrang“) ergibt sich durch Postorder-Durchlauf des Import-Baums, also (in aufsteigender Reihenfolge): **D**, **B**, **E**, **C**, **A**.

Mit `xsl:apply-imports` kann also das Verhalten der Regelauswahl beeinflusst werden.



Stylesheets importieren

Beispiel 6.33 (`import2.xsl`).

```
<xsl:stylesheet version="2.0"
          xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:template match="snip|snap">
    <p><xsl:apply-templates/></p>
  </xsl:template>
```

```
  <xsl:template match="*">
    <div><xsl:apply-templates/></div>
  </xsl:template>

</xsl:stylesheet>
```

Beispiel 6.34 (`import1.xsl`).

```
<xsl:stylesheet version="2.0"
          xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:import href="import2.xsl"/>

  <xsl:template match="/*">
    <body>
      <xsl:apply-templates/>
    </body>
  </xsl:template>

  <xsl:template match="snip">
    <h1><xsl:apply-templates/></h1>
  </xsl:template>

</xsl:stylesheet>
```

Beispiel 6.35 (Input).

```
<data>
  <snip>Hello world!</snip>
  <snap>Good morning</snap>
</data>
```

Beispiel 6.36 (Output).

```
<body>
  <h1>Hello world!</h1>
  <p>Good morning</p>
</body>
```

apply-imports

Definition 6.37 (Overriding Template Rules)**.**

```
<!-- Category: instruction -->
<xsl:apply-imports>
  <!-- Content: xsl:with-param* -->
</xsl:apply-imports>


<!-- Category: instruction -->
<xsl:next-match>
  <!-- Content: (xsl:with-param | xsl:fallback)* -->
</xsl:next-match>
```

- Eine Template-Regel, die andere außer Kraft setzt (siehe `xsl:import`), kann mit den Instruktionen `xsl:apply-imports` und `xsl:next-match` die außer Kraft gesetzten Template-Regeln aktivieren.
- `xsl:apply-imports` aktiviert nur Regeln in importierten Stylesheets,
- `xsl:next-match` aktiviert alle Regeln mit niedriger *Precedence* und/oder *Priorität*.

`apply-imports`

Beispiel 6.38 (`doc.xsl`).

```
<xsl:template match="example">
  <pre><xsl:apply-templates/></pre>
</xsl:template>
```

Beispiel 6.39 (Stylesheet importiert `doc.xsl`).

```
<xsl:import href="doc.xsl"/>

<xsl:template match="example">
  <div style="border: solid red">
     <xsl:apply-imports/>
  </div>
</xsl:template>
```

Beispiel 6.40 (Ergebnis der Transformation).

```
<div style="border: solid red"><pre>...</pre></div>
```

Konfliktauflösung

Ein Knoten kann auf mehr als eine Regel passen. Das Konfliktauflösungsverfahren ist exakt definiert.

Konfliktauflösung

1. Unter allen passenden Regeln werden nur die mit der höchsten *Import Precedence* berücksichtigt.
2. Innerhalb dieser Regeln werden nur die mit dem höchsten *Prioritätswert* berücksichtigt: Attribut `priority` oder aus den *Default-Regeln*.
3. Die Default-Regeln ergeben Werte von -0.5 bis 0.5. Spezifischere Patterns erhalten meist höhere Prioritätswerte als weniger spezifische.

Beispiele 6.41. `*` $\rightarrow -0.5$

`para` $\rightarrow 0$

`para[1]` $\rightarrow 0.5$

Wenn mehr als eine Regel übrig bleibt, ist das ein Fehler.

Der Prozessor kann diesen Fehler melden und abbrechen, oder er wählt die Regel aus, die im Stylesheet am weitesten *hinten* steht.

Schleifen: `for-each`

Definition 6.42 (Definition: Repetition)**.**

```
<!-- Category: instruction -->
<xsl:for-each
  select = expression>
  <!-- Content: (xsl:sort*, sequence-constructor) -->
</xsl:for-each>
```

Das Template wird für jedes Item der Ergebnissequenz des `select`-Ausdrucks instanziert. Der Kontext wird dabei analog zu `xsl:apply-templates` gesetzt.

Schleifen: `for-each`

Beispiel 6.43 (Daten).

```
<customers>
  <customer>
    <name>...</name>
    <order>...</order>
    <order>...</order>
  </customer>
  <customer>
    <name>...</name>
    <order>...</order>
    <order>...</order>
  </customer>
</customers>
```

Beispiel 6.44 (Stylesheet: Daten in Tabelle darstellen).

```
<xsl:template match="/">
  <html>
    <head>
      <title>Customers</title>
    </head>
    <body>
      <table>
        <tbody>
          <xsl:for-each select="customers/customer">
            <tr>
              <th>
                <xsl:apply-templates select="name"/>
              </th>
              <xsl:for-each select="order">
                <td>
                  <xsl:apply-templates/>
                </td>
              </xsl:for-each>
            </tr>
          </xsl:for-each>
        </tbody>
```

```
      </table>
    </body>
  </html>
</xsl:template>
```

Das ist schlechter Programmierstil!

- Unübersichtlich!
- Teile nicht wiederverwendbar!
- In der Praxis löst man solche Aufgaben mit mehreren Templates und `apply-templates`!

Beispiel 6.45 (Stylesheet: Daten in Tabelle darstellen, ohne `xsl:for-each`).

```
<xsl:template match="/">
  <html>
    <head><title>Customers</title></head>
    <body>
      <table>
        <tbody>
          <xsl:apply-templates select="customers/customer"/>
        </tbody>
      </table>
    </body>
  </html>
</xsl:template>

<xsl:template match="customer"">
  <tr>
    <th><xsl:apply-templates select="name"/></th>
    <xsl:apply-templates select="order"/>
  </tr>
</xsl:template>
```

Sinnvollere Beispiele für `xsl:for-each>`

An verschiedenen Stellen auf kommenden Folien...

Bedingte Verarbeitung

Definition 6.46 (Conditional Processing with `xsl:if`)**.**

```
<!-- Category: instruction -->
<xsl:if
  test = expression>
  <!-- Content: sequence-constructor -->
</xsl:if>
```

- Der Boolesche Wert des Ausdrucks im `test`-Attribut wird berechnet.
- Falls des Ergebnis `true` ist, wird der Sequence Constructor ausgewertet.
- Andernfalls wird die leere Sequenz zurückgegeben.

Bedingte Verarbeitung

Beispiel 6.47 (Einträge durch Komma trennen).

```
<xsl:template match="namelist/name">
  <xsl:apply-templates/>
  <xsl:if test="not(position() eq last())">, </xsl:if>
</xsl:template>
```

Beispiel 6.48 (Einträge durch Komma trennen).

```
<xsl:template match="namelist/name">
  <xsl:if test="position() gt 1">, </xsl:if>
  <xsl:apply-templates/>
</xsl:template>
```

Beispiel 6.49 (jede zweite Zeile einfärben).

```
<xsl:template match="item">
  <tr>
    <xsl:if test="position() mod 2 = 0">
       <xsl:attribute name="class">even</xsl:attribute>
    </xsl:if>
    <xsl:apply-templates/>
  </tr>
</xsl:template>
```

Bedingte Verarbeitung

Definition 6.50 (Conditional Processing with `xsl:choose`)**.**

```
<!-- Category: instruction -->
<xsl:choose>
  <!-- Content: (xsl:when+, xsl:otherwise?) -->
</xsl:choose>


<xsl:when
  test = expression>
  <!-- Content: sequence-constructor -->
</xsl:when>


<xsl:otherwise>
  <!-- Content: sequence-constructor -->
</xsl:otherwise>
```

- Der Sequence Constructor des ersten `xsl:when`-Elements wird ausgewertet, dessen `test`-Ausdruck erfüllt ist
- Sind alle Tests nicht erfüllt, wird der Sequence Constructor der `xsl:otherwise`-Instruktion instanziert, falls vorhanden. Andernfalls wird die leere Sequenz zurückgegeben.

Bedingte Verarbeitung

Beispiel 6.51 (Verschachtelte Listen nummerieren).

```
<xsl:template match="orderedlist/listitem">
  <fo:list-item indent-start='2pi'>
    <fo:list-item-label>
      <xsl:variable name="level" select="count(ancestor::orderedlist) mod 3"/>
      <xsl:choose>
        <xsl:when test='$level=1'>
          <xsl:number format="i"/>
        </xsl:when>
        <xsl:when test='$level=2'>
          <xsl:number format="a"/>
        </xsl:when>
        <xsl:otherwise>
          <xsl:number format="1"/>
        </xsl:otherwise>
      </xsl:choose>
      <xsl:text>. </xsl:text>
    </fo:list-item-label>
    <fo:list-item-body>
      <xsl:apply-templates/>
    </fo:list-item-body>
  </fo:list-item>
</xsl:template>
```

6.4 Variablen und Parameter

Variablen und Parameter

Definition 6.52 (Variables)**.**

```
<!-- Category: declaration -->
<!-- Category: instruction -->
<xsl:variable
  name = qname
  select? = expression
  as? = sequence-type>
  <!-- Content: sequence-constructor -->
</xsl:variable>
```

`xsl:variable` und `xsl:param` heißen *Bindungselemente* und definieren *Variablenbindungen* (Sequenzen an Namen).
Beispiel 6.53 (globale und lokale Variablen).

```
<xsl:stylesheet ...>
  <xsl:variable name="numItems" select="count(/data/item)"/>

  <xsl:template match="item">
    <xsl:variable name="frac" select="value div $numItems"/>
    ...
  </xsl:template>
</xsl:stylesheet>
```

Variablen und Parameter

Parameter sind Variablen mit der zusätzlichen Eigenschaft, dass der Wert bei gewissen Aufrufen gesetzt werden kann.

Definition 6.54 (Parameters)**.**

```
<!-- Category: declaration -->
<xsl:param
  name = qname
  select? = expression
  as? = sequence-type
  required? = "yes" | "no"
  tunnel? = "yes" | "no">
  <!-- Content: sequence-constructor -->
</xsl:param>
```

- *Globale Parameter* des Stylesheets,
- *lokale Parameter* von Templates und Funktionen
- Die `xsl:param`-Elemente können *Default-Bindungen* enthalten für den Fall, dass der Aufrufer keinen Wert liefert.
- Das Attribut `required` ist nur bei Stylesheet- und Templateparametern erlaubt.

  (Funktionsparameter müssen immer zwingend angegeben werden.)

Variablen und Parameter — Werte

Werte ohne as-Attribut

1. Hat das Bindungselement ein `select`-Attribut, muss dessen Wert ein Ausdruck sein. Der Bindungswert ist der Wert dieses Ausdrucks, eine Sequenz.

   (In diesem Fall muss das Bindungselement ein leeres Element sein!)

   ```
   <xsl:variable name="summe" select="1 + 2 + 3"/>
   ```

2. Hat das Bindungselement kein `select`-Attribut und ist leer, dann ist der Bindungswert der leere String.

   ```
   <xsl:variable name="nix"/>
   ```

3. Hat das Bindungselement kein `select`-Attribut und nicht-leeren Inhalt, dann ist der Bindungswert ein *temporärer Baum*. Die Wurzel ist ein Dokumentknoten, dessen Kinder die Knoten der Sequenz sind.

   ```
   <xsl:param name="twoPoints">
     <li>Truth</li>
     <li>Freedom</li>
   </xsl:param>
   ```

Variablen und Parameter

Beispiel 6.55.

```
<xsl:variable name="i" as="xs:integer*" select="1 to 3"/>
```

Beispiel 6.56 (⇒ Sequenz (2, 4, 6)).

```
<xsl:variable name="seq" as="xs:integer*">
  <xsl:for-each select="1 to 3">
    <xsl:sequence select=". * 2"/>
  </xsl:for-each>
</xsl:variable>
```

Beispiel 6.57.

```
<xsl:variable name="i" as="xs:integer" select="@size"/>
```

Beispiel 6.58 (⇒ Sequenz von Attributknoten).

```
<xsl:variable name="attset" as="attribute()+">
  <xsl:attribute name="x">2</xsl:attribute>
  <xsl:attribute name="y">3</xsl:attribute>
</xsl:variable>
```

Beispiel 6.59 (⇒ Knoten).

```
<xsl:variable name="n">2</xsl:variable>
```

Beispiel 6.60 (⇒ atomarer Wert).

```
<xsl:variable name="n" select="2"/>
```

Temporäre Bäume

Beispiel 6.61 (Two-Phase Transformation).

```
<xsl:stylesheet
  version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:import href="phase1.xsl"/>
  <xsl:import href="phase2.xsl"/>

  <xsl:variable name="intermediate">
    <xsl:apply-templates select="/"
                         mode="phase1"/>
  </xsl:variable>

  <xsl:template match="/">
    <xsl:apply-templates select="$intermediate"
                         mode="phase2"/>
  </xsl:template>

</xsl:stylesheet>
```

Globale Variablen und Stylesheetparameter

Globale Namen dürfen nur einmal gebunden werden.

Sichtbarkeit: ganzes Stylesheet (Überdeckung durch lokale Variablen/Parameter möglich)

Beispiel 6.62.

```
<xsl:param name="para-font-size"
           as="xs:string">12pt</xsl:param>

<xsl:template match="para">
 <fo:block font-size="{$para-font-size}">
   <xsl:apply-templates/>
 </fo:block>
</xsl:template>
```

Beispiel 6.63 (Reihenfolge beliebig bei globalen Variablen und Parametern).

```
<xsl:variable name="example-font-size"
              select="$para-font-size"/>
<xsl:param name="para-font-size"
           as="xs:string">12pt</xsl:param>
```

Lokale Variablen und Parameter

Sichtbarkeit: in den nachfolgenden *Siblings* und deren *Descendants* (Überdeckung durch lokale Variablen möglich)

Beispiel 6.64.

```
<xsl:template name="html-head">
  <xsl:param name="title"/>
  <head>
     <title><xsl:value-of select="$title"/></title>
     <xsl:variable name="media">print</<xsl:variable>
     ...
  </head>
</xsl:template>
```

Beispiel 6.65 (Local Variable Shadowing a Global Variable).

```
<xsl:param name="x" select="1"/>
<xsl:template name="foo">
  <xsl:variable name="x" select="2"/>
  <xsl:value-of select="$x"/>
</xsl:template>
```

Named Templates

Templates, die ein `name`-Attribut besitzen, können direkt aufgerufen werden.

Definition 6.66 (Named Templates)**.**

```
<!-- Category: instruction -->
<xsl:call-template
  name = qname>
  <!-- Content: xsl:with-param* -->
</xsl:call-template>
```

- Beim Aufruf mit `call-template` werden eventuelle `match`-, `priority`- und `mode`-Attribute ignoriert.
- Der Fokus ändert sich nicht beim Aufruf von `call-template`.

Parameterübergabe an Templates

Definition 6.67 (Passing Parameters to Templates)**.**

```
<xsl:with-param
  name = qname
  select? = expression
  as? = sequence-type
  tunnel? = "yes" | "no">
  <!-- Content: sequence-constructor -->
</xsl:with-param>
```

- Als *Child* erlaubt bei `xsl:apply-templates`, `xsl:call-template`, `xsl:apply-imports`, `xsl:next-match`.
- Bestimmung des Bindungswerts wie bei `xsl:variable` und `xsl:param`.
- Der Fokus wird dabei vom *Parent*-Element übernommen.

Parameterübergabe an Templates

Beispiel 6.68 (Parameterübergabe).

```
<xsl:template name="numbered-block">
  <xsl:param name="format">1. </xsl:param>
  <fo:block>
    <xsl:number format="{$format}"/>
    <xsl:apply-templates/>
  </fo:block>
</xsl:template>

<xsl:template match="ol//ol/li">
  <xsl:call-template name="numbered-block">
    <xsl:with-param name="format">a. </xsl:with-param>
  </xsl:call-template>
</xsl:template>
```

Tunnel-Parameter

Ein einmal gesetzter Tunnel-Parameter wird durch alle untergeordneten Template-Aufrufe gereicht, ohne dass diese den Parameter deklarieren müssen.

Beispiel 6.69 (Tunnel-Parameter verwenden).

```
<xsl:template match="equation">
  <xsl:param name="equation-format" select="'(1)'"
             tunnel="yes"/>
  <xsl:number level="any" format="{$equation-format}"/>
</xsl:template>
```

Beispiel 6.70 (weiter oben in der Berechnung ...).

```
<xsl:template match="appendix">
  ...
  <xsl:apply-templates>
    <xsl:with-param name="equation-format" select="'[i]'"
                    tunnel="yes"/>
  </xsl:apply-templates>
  ...
</xsl:template>
```

6.5 Weitere Instruktionen

Attributwert-Templates

- Attributwerte von (u. a.) *Literal Result Elements* sind *Attributwert-Templates*.
- Attributwert-Templates können Ausdrücke in geschweiften Klammern `{expr}` enthalten.
- Diese werden zum Auswertungszeitpunkt durch den Stringwert des Ausdrucks ersetzt.
- Als Attributwert-Templates werden u. a. interpretiert:
  - Attributwerte in *Literal Result Elements*
  - `name`- und `namespace`-Attributwerte (in `xsl:element`- und `xsl:attribute`-Elementen usw.)
- Als Attributwert-Templates werden z. B. *nicht* interpretiert:
  - Attributwerte, die (per Definition) Ausdrücke, Patterns oder Namensreferenzen auf XSLT-Objekte enthalten
  - Attributwerte von Deklarationen (Top-Level-Elemente)
  - `xmlns`-Attribute

Attributwert-Templates

Beispiel 6.71 (Attributwert-Templates).

```
<xsl:variable name="image-dir" select="'/images'"/>

<xsl:template match="photograph">
  <img src="{$image-dir}/{href}" width="{size/@width}"/>
</xsl:template>
```

Beispiel 6.72 (Eingabe).

```
<photograph>
  <href>headquarters.jpg</href>
  <size width="300"/>
</photograph>
```

Beispiel 6.73 (Ausgabe).

```
<img src="/images/headquarters.jpg" width="300"/>
```

Attributwert-Templates

Beispiel 6.74 (Producing a Space-Separated List).

```
<temperature readings="{10.32, 5.50, 8.31}"/>
```

Beispiel 6.75 (Output).

```
<temperature readings="10.32 5.5 8.31"/>
```

Überflüssige Namespaces entfernen

Beispiel 6.76 (Excluding Namespaces from the Result Tree).

```
<xsl:stylesheet xsl:version=1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:a="a.uri"
  xmlns:b="b.uri">
  exclude-result-prefixes="#all">

<xsl:template match="/">
  <foo xmlns:c="c.uri" xmlns:d="d.uri"
       xsl:exclude-result-prefixes="c"/>
</xsl:template>

</xsl:stylesheet>
```

Beispiel 6.77 (Output).

```
<foo xmlns:d="d.uri"/>
```

Namespace-Alias

Wenn das Ausgabedokument einen Namespace benötigt, der im Stylesheet zu Konflikten führt ...

Beispiel 6.78 (Generate a Stylesheet).

```
<xsl:stylesheet
  version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:fo="http://www.w3.org/1999/XSL/Format"
  xmlns:axsl="file://namespace.alias">

  <xsl:namespace-alias stylesheet-prefix="axsl"
                       result-prefix="xsl"/>
...
  <xsl:template match="block">
    <axsl:template match="{.}">
       <fo:block><axsl:apply-templates/></fo:block>
    </axsl:template>
  </xsl:template>

</xsl:stylesheet>
```

Elemente erzeugen mit `xsl:element`

Definition 6.79 (Creating Element Nodes Using `xsl:element`)**.**

```
<!-- Category: instruction -->
<xsl:element
  name = { qname }
  namespace? = { uri-reference }
  inherit-namespaces? = "yes" | "no"
  use-attribute-sets? = qnames
  type? = qname
  validation? = "strict" | "lax" | "preserve" | "strip">
  <!-- Content: sequence-constructor -->
</xsl:element>
```

- Mit `xsl:element` können Name und Namespace des erzeugten Elementknotens „berechnet“ werden.
- Die Attributwerte von `name` und `namespace` sind *Attributwert-Templates*.
- Der *Sequence-Constructor* in xsl:element liefert Attribute und Kindknoten des neuen Elementknotens.

Attribute erzeugen mit `xsl:attribute`

Definition 6.80 (Creating Attribute Nodes Using `xsl:attribute`)**.**

```
<!-- Category: instruction -->
<xsl:attribute
  name = { qname }
  namespace? = { uri-reference }
  select? = expression
  separator? = { string }
  type? = qname
  validation? = "strict" | "lax" | "preserve" | "strip">
  <!-- Content: sequence-constructor -->
</xsl:attribute>
```

Attribute erzeugen mit `xsl:attribute`

Beispiel 6.81 (Sequenzen ausgeben mit `separator`).

```
<xsl:template match="/">
  <hello>
    <this>
      <xsl:attribute name="a" select="1 to 5"/>
    </this>
    <that>
      <xsl:attribute name="{substring('abc', 2, 1)}"
                     separator=", " select="1 to 5"/>
    </that>
  </hello>
</xsl:template>
```

Beispiel 6.82 (Ausgabe).

```
<hello>
  <this a="1 2 3 4 5"/>
  <that b="1, 2, 3, 4, 5"/>
</hello>
```

Elemente und Attribute erzeugen

Beispiel 6.83 (`xsl:element` und `xsl:attribute`).

```
<xsl:template match="/">
  <hello>
    <test a="eins"/>
    <test>
      <xsl:attribute name="b">zwei</xsl:attribute>
    </test>
    <xsl:element name="{concat('te','s','t')}">
      <xsl:attribute name="c">drei</xsl:attribute>
    </xsl:element>
    <xsl:element name="p:b" namespace="urn:foo.bar">
      <c>hello</c>
    </xsl:element>
  </hello>
</xsl:template>
```

Beispiel 6.84 (Ausgabe).

```
<hello>
   <test a="eins"/>
   <test b="zwei"/>
   <test c="drei"/>
   <p:b xmlns:p="urn:foo.bar"><c>hello</c></p:b>
</hello>
```

Named Attribute Sets

Definition 6.85 (Named Attribute Sets)**.**

```
<!-- Category: declaration -->
<xsl:attribute-set
  name = qname
  use-attribute-sets? = qnames>
  <!-- Content: xsl:attribute* -->
</xsl:attribute-set>
```

Beispiel 6.86.

```
<xsl:template match="chapter/heading">
  <fo:block font-stretch="condensed"
            xsl:use-attribute-sets="title-style">
    <xsl:apply-templates/>
  </fo:block>
</xsl:template>

<xsl:attribute-set name="title-style">
  <xsl:attribute name="font-size">12pt</xsl:attribute>
  <xsl:attribute name="font-weight">bold</xsl:attribute>
</xsl:attribute-set>
```

Named Attribute Sets

Beispiel 6.87 (Overriding Attributes in an Attribute Set)*.*

```
<xsl:attribute-set name="base-style">
  <xsl:attribute name="font-family">Univers</xsl:attribute>
  <xsl:attribute name="font-size">10pt</xsl:attribute>
  <xsl:attribute name="font-style">normal</xsl:attribute>
  <xsl:attribute name="font-weight">normal</xsl:attribute>
</xsl:attribute-set>

<xsl:template match="o">
  <fo:block xsl:use-attribute-sets="base-style"
            font-size="12pt"
            font-style="italic">
    <xsl:attribute name="font-size">14pt</xsl:attribute>
    <xsl:attribute name="font-weight">bold</xsl:attribute>
    <xsl:apply-templates/>
  </fo:block>
</xsl:template>
```

Beispiel 6.88 (Output).

```
<fo:block font-family="Univers"
          font-size="14pt"
          font-style="italic"
          font-weight="bold">
...
</fo:block>
```

Erzeugen von Textknoten

Literal Text Nodes im Stylesheet, oder:

Definition 6.89 (Creating Text Nodes Using `xsl:text`)**.**

```
<!-- Category: instruction -->
<xsl:text
  [disable-output-escaping]? = "yes" | "no">
  <!-- Content: #PCDATA -->
</xsl:text>
>
```

The attribute `disable-output-escaping` *is deprecated.*

Beispiel 6.90.

```
<xsl:function name="f:wrap">
  <xsl:param name="s"/>
  <xsl:text>(</xsl:text>
  <xsl:value-of select="$s"/>
  <xsl:text>)</xsl:text>
</xsl:function>

... <xsl:value-of select="f:wrap('---')"/> ...
```

Beispiel 6.91 (Ausgabe).

```
(---)
```

Erzeugen von Textknoten

Definition 6.92 (Generating Text with `xsl:value-of`)**.**

```
<!-- Category: instruction -->
<xsl:value-of
  select? = expression
  separator? = { string }
  [disable-output-escaping]? = "yes" | "no">
  <!-- Content: sequence-constructor -->
</xsl:value-of>
```

Beispiel 6.93.

```
<x><xsl:value-of select="1 to 4"/></x>
<y><xsl:value-of select="1 to 4" separator="|"/></y>
```

Beispiel 6.94 (Ausgabe).

```
<x>1 2 3 4</x>
<y>1|2|3|4</y>
```

Erzeugen von Dokumentknoten

Definition 6.95 (Creating Document Nodes)**.**

```
<!-- Category: instruction -->
<xsl:document
  validation? = "strict" | "lax" | "preserve" | "strip"
  type? = qname>
  <!-- Content: sequence-constructor -->
</xsl:document>
```

Beispiel 6.96 (Checking Uniqueness Constraints in a Temporary Tree).

```
<xsl:variable name="tree" as="document-node()">
  <xsl:document validation="strict">
    <xsl:apply-templates/>
  </xsl:document>
</xsl:variable>
```

Erzeugen von Verarbeitungsanweisungen

Definition 6.97 (Creating Processing Instructions)**.**

```
<!-- Category: instruction -->
<xsl:processing-instruction
  name = { ncname }
  select? = expression>
  <!-- Content: sequence-constructor -->
</xsl:processing-instruction>
```

Beispiel 6.98.

```
<xsl:processing-instruction name="xml-stylesheet"
  select="('href=&quot;book.css&quot;',
           'type=&quot;text/css&quot;')"/>
```

Beispiel 6.99 (Ausgabe).

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

Erzeugen von Kommentaren

Definition 6.100 (Creating Comments)**.**

```
<!-- Category: instruction -->
<xsl:comment
  select? = expression>
  <!-- Content: sequence-constructor -->
</xsl:comment>
```

Beispiel 6.101.

```
<xsl:comment>This file is generated.</xsl:comment>
```

Beispiel 6.102 (Ausgabe).

```
<!--This file is generated.-->
```

Kopieren, flache Kopie (Shallow Copy)

Definition 6.103 (Shallow Copy)**.**

```
<!-- Category: instruction -->
<xsl:copy
  copy-namespaces? = "yes" | "no"
  inherit-namespaces? = "yes" | "no"
  use-attribute-sets? = qnames
  type? = qname
  validation? = "strict" | "lax" | "preserve" | "strip">
  <!-- Content: sequence-constructor -->
</xsl:copy>
```

Das *Context Item* wird kopiert, ohne Attribute, Kindknoten usw.

Kopieren, flache Kopie (Shallow Copy)

Beispiel 6.104.

```
<xsl:template match="ul">
  <xsl:copy/>
  <xsl:copy>
    <xsl:attribute name="a" select="1"/>
  </xsl:copy>
</xsl:template>
```

Beispiel 6.105 (Eingabe).

```
<ul class="tidy">
  <li>Eins</li>
  <li>Zwei</li>
</ul>
```

Beispiel 6.106 (Ausgabe).

```
<ul/>
<ul a="1"/>
```

Kopieren, flache Kopie (Shallow Copy)

Beispiel 6.107 (Identische Transformation).

```
<xsl:template match="@*|node()">
  <xsl:copy>
    <xsl:apply-templates select="@*|node()"/>
  </xsl:copy>
</xsl:template>
```

Beispiel 6.108 (`xml:lang`-Attribut durchkopieren).

```
<xsl:template name="copy-lang">
  <xsl:for-each select="@xml:lang">
    <xsl:copy/>
  </xsl:for-each>
</xsl:template>
```

Kopieren, tiefe Kopie (Deep Copy)

Definition 6.109 (Deep Copy)**.**

```
<!-- Category: instruction -->
<xsl:copy-of
  select = expression
  copy-namespaces? = "yes" | "no"
  type? = qname
  validation? = "strict" | "lax" | "preserve" | "strip" />
```

Rekursives Kopieren aller *Items* der berechneten Sequenz.

Auch Attribute werden kopiert.

Kopieren, tiefe Kopie (Deep Copy)

Beispiel 6.110.

```
<xsl:template match="ul">
  <xsl:copy-of select="."/>
</xsl:template>
```

Beispiel 6.111 (Eingabe).

```
<ul class="tidy">
  <li>Eins</li>
  <li>Zwei</li>
</ul>
```

Beispiel 6.112 (Ausgabe).

```
<ul class="tidy">
  <li>Eins</li>
  <li>Zwei</li>
</ul>
```

Sequenzen erzeugen

Definition 6.113 (Constructing Sequences)**.**

```
<!-- Category: instruction -->
<xsl:sequence
  select = expression>
  <!-- Content: xsl:fallback* -->
</xsl:sequence>
```

Beispiel 6.114*.*

```
<xsl:variable name="values" as="xs:integer*">
  <xsl:sequence select="(1,2,3,4)"/>
  <xsl:sequence select="(8,9,10)"/>
</xsl:variable>
<xsl:value-of select="sum($values)"/>
```

Beispiel 6.115 (Ausgabe)*.*

37

6.6 Sortieren / Gruppieren

Sortierung

Definition 6.116 (Rückblick: Applying Template Rules)**.**

```
<!-- Category: instruction -->
<xsl:apply-templates
  select? = expression
  mode? = token>
  <!-- Content: (xsl:sort | xsl:with-param)* -->
</xsl:apply-templates>
```

Sortierung

Definition 6.117 (Sorting)**.**

```
<xsl:sort
  select? = expression
  lang? = { nmtoken }
  order? = { "ascending" | "descending" }
  collation? = { uri }
  stable? = { "yes" | "no" }
  case-order? = { "upper-first" | "lower-first" }
  data-type? = { "text" | "number" | qname-but-not-ncname }>
  <!-- Content: sequence-constructor -->
</xsl:sort>
```

- `xsl:sort`-Elemente heißen *Sort Key Components.*
- Als Children von `xsl:apply-templates`, `xsl:for-each`, `xsl:for-each-group` und `xsl:perform-sort` bewirken sie eine Sortierung der berechneten Knotenmenge.
- Das erste `xsl:sort`-Element gibt den primären Sortierschlüssel an, das zweite den sekundären (benötigt, wenn Werte bzgl. des ersten gleich sind), usw.

Sortierung

- Der Sortierschlüssel berechnet sich aus dem `select`-Attribut des `xsl:sort`-Elements oder aus dessen Inhalt. Default ist `select="."`
- Das *Context Item* ist das *Item* der unsortierten Sequenz, dessen Sortierschlüssel berechnet wird.
- Der Vergleich bzgl. vorgegebener Datentypen kann durch *Cast*-Funktionen erreicht werden.

Beispiel 6.118 (Datumsvergleich).

```
<xsl:sort select="xs:date(@dob)"/>
```

Werte vom Typ `xs:string` können unter einer *Collation* verglichen werden, d. h. wie mit der XPath-Funktion `compare($a, $b, $collation)`.

Sortierung — Kontrollattribute

`order` `ascending` oder `descending`

`collation` (alternativ zu `lang` und `case-order`)

`lang` The `lang` attribute indicates that a collation suitable for a particular natural language should be used.

`case-order` `upper-first` (A a B b) oder `lower-first` (a A b B)

`stable` Stabiles Sortieren (`yes`) oder implementierungsabhängig (`no`)

(nur im ersten `xsl-sort`-Element)

`data-type` For backwards compatibility with XSLT 1.0 ...

Sequenzen sortiert abarbeiten

Mit `xsl:apply-templates` (oder auch `xsl:for-each`)

Beispiel 6.119 (Sortieren nach Name).

```
<xsl:template match="employees">
  <ul>
    <xsl:apply-templates select="employee">
      <xsl:sort select="name/family"/>
      <xsl:sort select="name/given"/>
    </xsl:apply-templates>
  </ul>
</xsl:template>

<xsl:template match="employee">
```

```
  <li>
    <xsl:value-of select="name/given"/>
    <xsl:text> </xsl:text>
    <xsl:value-of select="name/family"/>
  </li>
</xsl:template>
```

Sequenzen sortieren

Definition 6.120 (Creating a Sorted Sequence)**.**

```
<!-- Category: instruction -->
<xsl:perform-sort
  select? = expression>
  <!-- Content: (xsl:sort+, sequence-constructor) -->
</xsl:perform-sort>
```

Beispiel 6.121 (Sorting a Sequence of Atomic Values)*.*

```
<xsl:function name="local:sort" as="xdt:anyAtomicType*">
  <xsl:param name="in" as="xdt:anyAtomicType*"/>
  <xsl:perform-sort select="$in">
    <xsl:sort select="."/>
  </xsl:perform-sort>
</xsl:function>
```

Sequenzen sortieren

The following example defines a function that sorts books by price, and uses this function to output the five books that have the lowest prices:

Beispiel 6.122*.*

```
<xsl:function name="bib:books-by-price"
              as="schema-element(bib:book)*">
  <xsl:param name="in" as="schema-element(bib:book)*"/>
  <xsl:perform-sort select="$in">
    <xsl:sort select="xs:decimal(bib:price)"/>
  </xsl:perform-sort>
</xsl:function>

<xsl:template ...>
  ...
  <xsl:copy-of select="bib:books-by-price(//bib:book)
                       [position() = 1 to 5]"/>
  ...
</xsl:template>
```

Gruppierung

- Gruppieren nach gemeinsamen Werten

- Gruppieren nach Position in einer Sequenz
- Gruppen fester Länge
- Verschachtelung möglich
- Zwei neue Funktionen in XPath-Ausdrücken:

Definition 6.123 (The Current Group)**.**

```
current-group() as item()*
```

Liste der Items in der aktuellen Gruppe

Definition 6.124 (The Current Grouping Key)**.**

```
current-grouping-key() as xdt:anyAtomicType?
```

Der gemeinsame Wert einer Gruppe

Gruppierung

Definition 6.125 (The `xsl:for-each-group` Element)**.**

```
<!-- Category: instruction -->
<xsl:for-each-group
  select = expression
  group-by? = expression
  group-adjacent? = expression
  group-starting-with? = pattern
  group-ending-with? = pattern
  collation? = { uri }>
  <!-- Content: (xsl:sort*, sequence-constructor) -->
</xsl:for-each-group>
```

- Die durch das Attribut `select` gegebene Sequenz wird gruppiert.
- Die Gruppierung wird spezifiziert durch *genau eines* der Attribute `group-by`, `group-adjacent`, `group-starting-with` oder `group-ending-with`.

Gruppierung mit `group-by` (1. Beispiel)

Beispiel 6.126 (Eingabe).

```
<list>
  <person sex="male">
    <firstname>Adam</firstname>
    <lastname>Apple</lastname>
  </person>
  <person sex="female">
    <firstname>Susan</firstname>
    <lastname>Smith</lastname>
```

```
  </person>
  <person sex="female">
    <firstname>Anne</firstname>
    <lastname>Miller</lastname>
  </person>
  <person sex="male">
    <firstname>John</firstname>
    <lastname>Smith</lastname>
  </person>
</list>
```

Beispiel 6.127 (Stylesheet).

```
<xsl:stylesheet version="2.0"
        xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:template match="/">
    <doc>
      <xsl:for-each-group select="//person" group-by="@sex">
        <section>
          <title>
            <xsl:value-of select="current-grouping-key()"/>
          </title>
          <xsl:apply-templates select="current-group()"/>
        </section>
      </xsl:for-each-group>
    </doc>
  </xsl:template>

  <xsl:template match="person">
    ...
  </xsl:template>
</xsl:stylesheet>
```

Beispiel 6.128 (Ausgabe).

```
<doc>
   <section>
      <title>male</title>
      <para>Adam Apple</para>
      <para>John Smith</para>
   </section>
   <section>
      <title>female</title>
      <para>Susan Smith</para>
      <para>Anne Miller</para>
   </section>
</doc>
```

Gruppierung mit group-by (2. Beispiel)

Beispiel 6.129 (Eingabe).

```
<cities>
```

```
  <city name="Milano"  country="Italia"      pop="5"/>
  <city name="Paris"   country="France"      pop="7"/>
  <city name="München" country="Deutschland" pop="4"/>
  <city name="Lyon"    country="France"      pop="2"/>
  <city name="Venezia" country="Italia"      pop="1"/>
</cities>
```

Beispiel 6.130 (gewünschte Ausgabe).

```
<table>
  <tr>
    <th>Position</th><th>Country</th><th>Cities</th>
    <th>Population</th>
  </tr>
  <tr>
    <td>1</td><td>Italia</td><td>Milano, Venezia</td><td>6</td>
  </tr>
  <tr>
    <td>2</td><td>France</td><td>Lyon, Paris</td><td>9</td>
  </tr>
  <tr>
    <td>3</td><td>Deutschland</td><td>München</td><td>4</td>
  </tr>
</table>
```

Beispiel 6.131 (gewünschte Ausgabe).

Position	Country	Cities	Population
1	Italia	Milano, Venezia	6
2	France	Lyon, Paris	9
3	Deutschland	München	4

Beispiel 6.132 (Stylesheet).

```
<xsl:template match="/">
  <table>
    <tr>
      <th>Position</th><th>Country</th><th>Cities</th>
      <th>Population</th>
    </tr>
    <xsl:for-each-group select="cities/city"
                        group-by="@country">
      <tr>
        <td><xsl:value-of select="position()"/></td>
        <td><xsl:value-of select="@country"/></td>
        <td>
          <xsl:value-of select="current-group()/@name"
                        separator=", "/>
        </td>
        <td><xsl:value-of select="sum(current-group()/@pop)"/></td>
      </tr>
    </xsl:for-each-group>
  </table>
</xsl:template>
```

Gruppierung mit Sortierung

Beispiel 6.133 (Gruppen sortieren nach Key).

```
<xsl:sort select="current-grouping-key()"/>
```

Beispiel 6.134 (Gruppen sortieren nach Größe).

```
<xsl:sort select="count(current-group())"/>
```

Gruppierung mit `group-by` (3. Beispiel)

`group-by` liefert eine Sequenz ⇒ Elemente können zu mehreren Gruppen gehören.
Beispiel 6.135 (Eingabe).

```
<titles>
  <title>A Beginner's Guide to <ix>Java</ix></title>
  <title>Learning <ix>XML</ix></title>
  <title>Using <ix>XML</ix> with <ix>Java</ix></title>
</titles>
```

Beispiel 6.136 (Stylesheet).

```
<xsl:template match="titles">
  <xsl:for-each-group select="title" group-by="ix">
    <h2><xsl:value-of select="current-grouping-key()"/></h2>
    <xsl:for-each select="current-group()">
      <p><xsl:value-of select="."/></p>
    </xsl:for-each>
  </xsl:for-each-group>
</xsl:template>
```

Beispiel 6.137 (Ausgabe).

```
<h2>Java</h2>
    <p>A Beginner's Guide to Java</p>
    <p>Using XML with Java</p>
<h2>XML</h2>
    <p>Learning XML</p>
    <p>Using XML with Java</p>
```

Gruppierung mit `group-starting-with`

Beispiel 6.138 (Eingabe). Identifying a Group by its Initial Element

```
<body>
  <h2>Introduction</h2>
  <p>XSLT is used to write stylesheets.</p>
  <p>XQuery is used to query XML databases.</p>
  <h2>What is a stylesheet?</h2>
  <p>A stylesheet is an XML document used to define a
    transformation.</p>
  <p>Stylesheets may be written in XSLT.</p>
  <p>XSLT 2.0 introduces new grouping constructs.</p>
</body>
```

Beispiel 6.139 (Gewünschte Ausgabe).

```
<chapter>
  <section title="Introduction">
    <para>XSLT is used to write stylesheets.</para>
    <para>XQuery is used to query XML databases.</para>
  </section>
  <section title="What is a stylesheet?">
    <para>A stylesheet is an XML document used to define a
      transformation.</para>
    <para>Stylesheets may be written in XSLT.</para>
    <para>XSLT 2.0 introduces new grouping constructs.</para>
  </section>
</chapter>
```

Beispiel 6.140 (Stylesheet).

```
<xsl:template match="body">
  <chapter>
    <xsl:for-each-group select="*" group-starting-with="h2">
      <section title="{current-group()[self::h2]}">
        <xsl:for-each select="current-group()[self::p]">
          <para><xsl:value-of select="."/></para>
        </xsl:for-each>
      </section>
    </xsl:for-each-group>
  </chapter>
</xsl:template>
```

Gruppierung mit `group-adjacent`

Beispiel 6.141 (Eingabe). Grouping Alternating Sequences of Elements

```
<p>Do <em>not</em>:
  <ul><li>talk,</li>
      <li>eat, or</li>
      <li>use your mobile telephone</li>
  </ul>
  while you are in the cinema.
</p>
```

Beispiel 6.142 (Gewünschte Ausgabe).

```
<p>Do <em>not</em>:</p>
<ul>
  <li>talk,</li>
  <li>eat, or</li>
  <li>use your mobile telephone</li>
</ul>
<p>while you are in the cinema.</p>
```

Beispiel 6.143 (Stylesheet).

```
<xsl:template match="p">
  <xsl:for-each-group select="node()"
                      group-adjacent="self::ul or self::ol">
    <xsl:choose>
      <xsl:when test="current-grouping-key()">
        <xsl:copy-of select="current-group()"/>
      </xsl:when>
      <xsl:otherwise>
        <p>
          <xsl:copy-of select="current-group()"/>
        </p>
      </xsl:otherwise>
    </xsl:choose>
  </xsl:for-each-group>
</xsl:template>
```

In this case, the grouping key is a boolean condition, true or false, so the effect is that a grouping establishes a maximal sequence of nodes for which the condition is true, followed by a maximal sequence for which it is false, and so on.

6.7 Nummerierung

Nummerierung

Definition 6.144 (Numbering)**.**

```
<!-- Category: instruction -->
<xsl:number
  value? = expression
  select? = expression
  level? = "single" | "multiple" | "any"
  count? = pattern
  from? = pattern
  format? = { string }
  lang? = { nmtoken }
  letter-value? = { "alphabetic" | "traditional" }
  ordinal? = { string }
  grouping-separator? = { char }
  grouping-size? = { number } />
```

Eine Zahlensequenz wird unter Berücksichtigung der *Konvertierungsattribute* (wichtigstes: `format`) in einen Textknoten ausgegeben.

Nummerierung

- Vorgegebene Zahlensequenz (aus Attribut `value`) formatieren
- Position im Dokument nummerieren
- Konvertierungsattribute

Nummerierung

Vorgegebene Zahlensequenz formatieren

Das Attribut `value` wird zu einer Sequenz von ganzen Zahlen ausgewertet.

Beispiel 6.145 (Numbering a Sorted List).

```
<xsl:template match="items">
  <xsl:for-each select="item">
    <xsl:sort select="."/>
    <p>
      <xsl:number value="position()" format="1. "/>
      <xsl:value-of select="."/>
    </p>
  </xsl:for-each>
</xsl:template>
```

Nummerierung

Position im Dokument nummerieren

- Das Attribut `value` ist nicht vorhanden.
- Ein Knoten wird ausgewählt: Attribut `select`, falls vorhanden, sonst der Kontextknoten.
- Folgende Attribute kontrollieren die Nummerierung:
  - Das `level`-Attribut spezifiziert die zu betrachtenden Ebenen des Eingabebaums. Werte: `single`, `multiple` oder `any`. Default ist `single`.
  - Das `count`-Attribut spezifiziert über ein Pattern, welche Knoten auf den ausgewählten Ebenen gezählt werden sollen. Default sind die Knoten mit gleichem Typ und Namen wie der ausgewählte Knoten.
  - Das `from`-Attribut enthält ein Pattern, das angibt, ab welchem Knoten die Zählung startet.

Nummerierung über die Position

Beispiel 6.146 (Numbering the Items in an Ordered List).

```
<xsl:template match="ol/item">
  <fo:block>
    <xsl:number/>
    <xsl:text>. </xsl:text>
    <xsl:apply-templates/>
  </fo:block>
<xsl:template>
```

Beispiel 6.147 (Numbering Notes within a Chapter).

```
<xsl:template match="note">
  <fo:block>
     <xsl:number level="any" from="chapter" format="(1) "/>
     <xsl:apply-templates/>
  </fo:block>
</xsl:template>
```

Nummerierung über die Position

Beispiel 6.148 (Multi-Level Numbering).

```
<xsl:template match="title">
  <fo:block>
     <xsl:number level="multiple"
                 count="chapter|section|subsection"
                 format="1.1 "/>
     <xsl:apply-templates/>
  </fo:block>
</xsl:template>
<xsl:template match="appendix//title" priority="1">
  <fo:block>
     <xsl:number level="multiple"
                 count="appendix|section|subsection"
                 format="A.1 "/>
     <xsl:apply-templates/>
  </fo:block>
</xsl:template>
```

`chapter`: 1, 2, 3, ...; `section`: 1.1, 1.2, 1.3, ...; `subsection`: 1.1.1, 1.1.2, 1.1.3, ...;

`appendix`: A, B, C, ...; `section` in `appendix`: A.1, A.2, ...

Nummerierung — Konvertierungsattribute

- Sequenzen von Zahlen formatieren
- Attribut `format`, Default: `1`
- Alphanumerische Token: Zahlenformate

  `"1"` ⇒ 1 2 ... 10 11 12 ...

  `"01"` ⇒ 01 02 ... 09 10 11 12 ... 99 100 101 ...

  `"A"` ⇒ A B C ... Z AA AB AC ...

  `"a"` ⇒ a b c ... z aa ab ac ...

  `"i"` ⇒ i ii iii iv v vi vii viii ix x ...

  `"I"` ⇒ I II III IV V VI VII VIII IX X ...

  `"w"` ⇒ one two three four ...

  `"W"` ⇒ ONE TWO THREE FOUR ...

  `"Ww"` ⇒ One Two Three Four ...
- Restliche Zeichen: Trennzeichen

Nummerierung — Konvertierungsattribute

Beispiele 6.149 (Ordinalzahlen).

- `format="1" ordinal="-°" language="it"`

  `1° 2° 3° 4° ...`

- `format="Ww" ordinal="-o" language="it"`

  `Primo Secondo Terzo Quarto Quinto ...`

- `format="w" ordinal="-e" language="de"`

  `erste, zweite, dritte, vierte`

Beispiel 6.150 (Formatting a List of Numbers).

```
format="A-001(i)"
```

Output for (5, 13, 7):

```
E-013(vii)
```

Nummerierung — International

- `format="&#x30A2;"` specifies Katakana numbering
- `format="&#x30A4;"` specifies Katakana numbering in the „iroha“ order
- `format="&#x0E51;"` specifies Thai numbering
- `format="&#x05D0;" letter-value="traditional"` specifies „traditional“ Hebrew numbering
- `format="&#x10D0;" letter-value="traditional"` specifies Georgian numbering
- `format="&#x03B1;" letter-value="traditional"` specifies „classical“ Greek numbering
- `format="&#x0430;" letter-value="traditional"` specifies Old Slavic numbering

6.8 Stylesheet-Funktionen

Stylesheet Functions

Definition 6.151 (Stylesheet Functions)**.**

```
<!-- Category: declaration -->
<xsl:function
  name = qname
  as? = sequence-type
  override? = "yes" | "no">
  <!-- Content: (xsl:param*, sequence-constructor) -->
</xsl:function>
```

Der *Sequence-Constructor* definiert den Funktionswert.

Stylesheet-Funktionen können in beliebigen XPath-Ausdrücken verwenden werden, auch in Prädikaten in Patterns.

Stylesheet Functions

Beispiel 6.152 (Funktion).

```
<xsl:function name="num:roman" as="xs:string">
  <xsl:param name="value" as="xs:integer"/>
  <xsl:number value="$value" format="i"/>
</xsl:function>
```

Beispiel 6.153 (Funktionsaufruf).

```
<xsl:template match="/">
<output>
  <xsl:value-of select="upper-case(num:roman(42))"/>
</output>
</xsl:template>
```

Beispiel 6.154 (Ergebnis).

```
<output>XLII</output>
```

Stylesheet Functions

- Der Name einer Stylesheet-Funktion muss einen Präfix haben, der an einen unreservierten Namespace gebunden ist.
- Die `xsl:param`-Elemente definieren die Argumente und deren Anzahl die Stelligkeit der Funktion. (Positionsparameter)
- Die `xsl:param`-Elemente müssen leer sein und dürfen kein `select`-Attribut haben: kein Default, denn optionale Argumente sind nicht erlaubt.
- Stylesheet-Funktionen sind nicht polymorph.
- Im Rumpf der Funktion ist der Fokus zunächst undefiniert.
- Auf lokale Variablen der Aufrufstelle kann nicht zugegriffen werden (Static Scoping).
- Mit `override="yes"` können existierende (eingebaute) Funktionen überschrieben werden.

  Specifying `override='yes'` ensures interoperable behavior: the same code will execute with all processors.

  Specifying `override='no'` is useful when writing a fallback implementation of a function that is available with some processors but not others.

Stylesheet Functions

Beispiel 6.155.

```
<xsl:transform version="2.0" ...
  xmlns:str="http://example.com/namespace">

  <xsl:function name="str:reverse" as="xs:string">
    <xsl:param name="sentence" as="xs:string"/>
    <xsl:sequence select=
       "if (contains($sentence, ' '))
        then concat(str:reverse(substring-after($sentence, ' ')),
                    ' ',
                    substring-before($sentence, ' '))
        else $sentence"/>
  </xsl:function>

  <xsl:template match="/">
    <output>
      <xsl:value-of select="str:reverse('DOG BITES MAN')"/>
    </output>
  </xsl:template>

</xsl:transform>
```

6.9 Reguläre Ausdrücke

Reguläre Ausdrücke

Erinnerung: Reguläre Ausdrücke in XPath 2.0

- `fn:matches` returns a boolean result that indicates whether or not a string matches a given regular expression.
- `fn:replace` takes a string as input and returns a string obtained by replacing all substrings that match a given regular expression with a replacement string.
- `fn:tokenize` returns a sequence of strings formed by breaking a supplied input string at any separator that matches a given regular expression.

Reguläre Ausdrücke

Definition 6.156 (The `xsl:analyze-string` Instruction)**.**

```
<!-- Category: instruction -->
<xsl:analyze-string
  select = expression
  regex = { string }
  flags? = { string }>
  <!-- Content: (xsl:matching-substring?,
```

```
                xsl:non-matching-substring?,
                xsl:fallback*) -->
</xsl:analyze-string>
```

- Der mit `select` spezifierte String wird mit dem in `regex` gegebenen regulären Ausdruck verglichen.
- Die `flags` steuern die Auswertung (wie bei `fn:matches`, `fn:replace`, `fn:tokenize`).

Reguläre Ausdrücke

Definition 6.157.

```
<xsl:matching-substring>
  <!-- Content: sequence-constructor -->
</xsl:matching-substring>
```

Definition 6.158.

```
<xsl:non-matching-substring>
  <!-- Content: sequence-constructor -->
</xsl:non-matching-substring>
```

- `xsl:analyze-string` zerlegt den Eingabestring in eine Sequenz von Substrings, die zu dem regulären Ausdruck passen oder nicht passen.
- Passende Substrings werden mit `xsl:matching-substring` verarbeitet, nicht passende mit `xsl:non-matching-substring`.
- Dabei ist das *Context Item* der jeweilige Substring, die Kontextlänge die Anzahl der Substrings und die Kontextposition die relative Position des *Context Item* darin.

Reguläre Ausdrücke

Beispiel 6.159.

```
<xsl:template match="/">
<results>
   <xsl:analyze-string select="'abstract'" regex="[ab]+">
      <xsl:matching-substring>
        <yes><xsl:value-of select="."/></yes>
      </xsl:matching-substring>
      <xsl:non-matching-substring>
        <no><xsl:value-of select="."/></no>
      </xsl:non-matching-substring>
    </xsl:analyze-string>
  </results>
</xsl:template>
```

Beispiel 6.160 (Ausgabe).

```
<results>
   <yes>ab</yes> <no>str</no> <yes>a</yes> <no>ct</no>
</results>
```

Reguläre Ausdrücke

Definition 6.161 (Captured Substrings)**.**

```
regex-group($group-number as xs:integer) as xs:string
```

Innerhalb von `xsl:matching-substring` können *Captured Substrings* ausgelesen werden, die geklammerten Teilausdrücken des regulären Ausdrucks entsprechen.

Beispiel 6.162.

```
regex="([a-z]*)0([0-9]*)"
```

`regex-group(1)` liefert den Substring, der zum ersten Klammerausdruck `([a-z]*)` passt, usw.

Reguläre Ausdrücke

Beispiel 6.163 (Captured Substrings).

```
<xsl:template match="/">
  <results>
    <xsl:analyze-string select="'essen ist angerichtet'"
                        regex="([a-z]*)s([a-z]*)">
      <xsl:matching-substring>
        <yes pre="{regex-group(1)}" post="{regex-group(2)}">
          <xsl:value-of select="."/>
        </yes>
      </xsl:matching-substring>
    </xsl:analyze-string>
  </results>
</xsl:template>
```

Beispiel 6.164 (Ausgabe).

```
<results>
   <yes pre="es" post="en">essen</yes>
   <yes pre="i" post="t">ist</yes>
</results>
```

Reguläre Ausdrücke

Beispiel 6.165 (Replacing Characters by Elements). Problem: replace all `newline` characters in the abstract element by empty `br` elements.

```
<xsl:analyze-string select="abstract" regex="\n">
  <xsl:matching-substring>
    <br/>
  </xsl:matching-substring>
  <xsl:non-matching-substring>
    <xsl:value-of select="."/>
  </xsl:non-matching-substring>
</xsl:analyze-string>
```

Reguläre Ausdrücke

Beispiel 6.166 (Recognizing non-XML Markup Structure). Problem: replace all occurrences of `[...]` in the body by `cite` elements, retaining the content between the square brackets as the content of the new element.

```
<xsl:analyze-string select="body" regex="\[(.*?)\]">
  <xsl:matching-substring>
    <cite><xsl:value-of select="regex-group(1)"/></cite>
  </xsl:matching-substring>
  <xsl:non-matching-substring>
    <xsl:value-of select="."/>
  </xsl:non-matching-substring>
</xsl:analyze-string>
```

Reguläre Ausdrücke

Beispiel 6.167 (Parsing a Date). Problem: the input string contains a date such as `23 March 2002`. Convert it to the form `2002-03-23` (no error handling).

```
<xsl:variable name="months"
              select="'January', 'February', 'March', ..."/>

<xsl:analyze-string select="normalize-space($input)"
    regex="([0-9]{{1,2}})\s([A-Z][a-z]+)\s([0-9]{{4}})">
    <xsl:matching-substring>
        <xsl:number value="regex-group(3)" format="0001"/>
        <xsl:text>-</xsl:text>
        <xsl:number value="index-of($months, regex-group(2))"
                    format="01"/>
        <xsl:text>-</xsl:text>
        <xsl:number value="regex-group(1)" format="01"/>
    </xsl:matching-substring>
</xsl:analyze-string>
```

Reguläre Ausdrücke — Flags

Die Buchstaben im Attribut `flags` haben folgende Bedeutung:

`s`: *Dot-All Mode*, d. h. der Punkt `"."` passt auch zu Zeilenumbrüchen (`#x0A`). Default: der Punkt `"."` passt zu allen Zeichen außer Zeilenumbrüchen.

`m`: *Multi-Line Mode*, `^` und `$` passen zu Anfang bzw. Ende jeder Zeile im String. Default: `^` und `$` passen nur zu Anfang bzw. Ende des gesamten Strings.

`i`: *Case-Insensitive Mode* Default: Case-Sensitive

`x`: Whitespace im regulären Ausdruck wird ignoriert Default: Whitespace im regulären Ausdruck passt zu Whitespace im String.

6.10 Zusätzliche XPath-Funktionen

Zusätzliche XPath-Funktionen in XSLT

- Auf mehrere Eingabedokumente zugreifen: `document($uri-sequence)`
- Textdateien lesen: `unparsed-text($href)`
- Schlüssel und Zugriffspfade: `xsl:key`, `key(...)`
- Zahlen formatieren: `format-number(...)`
- Datum und Zeit formatieren: `format-dateTime(...)`, `format-date(...)`, `format-time(...)`
- `generate-id(...)`
- ...

Zusätzliche Funktionen — `document`

Definition 6.168 (Multiple Source Documents)**.**

```
document($uri-sequence as item()*) as node()*
```

Definition 6.169.

```
document($uri-sequence as item()*,
         $base-node as node()) as node()*
```

Zugriff auf weitere Dokumente: Eine Sequenz von URIs wird abgebildet auf eine Sequenz von Dokumentknoten.

Beispiel 6.170.

```
<xsl:variable name="cfg" select="document($configFile)"/>
```

Beispiel 6.171.

```
<xsl:template match="include-person-list">
  <xsl:apply-templates select="document(@href)//person"/>
</xsl:template>
```

Zusätzliche Funktionen — Textdateien lesen

Definition 6.172.

```
unparsed-text($href as xs:string?) as xs:string?
```

Definition 6.173.

```
unparsed-text($href as xs:string?,
              $encoding as xs:string) as xs:string?
```

The encoding of the external resource is determined as follows:

1. external encoding information is used if available, otherwise
2. if the media type of the resource is `text/xml` or `application/xml`, or if it matches the conventions `text/*+xml` or `application/*+xml`, then the encoding is recognized as specified in [XML 1.0], otherwise
3. the value of the `$encoding` argument is used if present, otherwise
4. UTF-8 is assumed.

Beispiel 6.174 (Splitting an Input File into a Sequence of Lines).

```
<xsl:for-each
    select="tokenize(unparsed-text($in), '\r?\n')">
 ...
</xsl:for-each>
```

Note that the unparsed-text function does not normalize line endings. This example has therefore been written to recognize both Unix and Windows conventions for end-of-line.

Zusätzliche Funktionen — Textdateien lesen

Because errors in evaluating the `unparsed-text` function are non-recoverable, two functions are provided to allow a stylesheet to determine whether a call with particular arguments would succeed:

Definition 6.175.

```
unparsed-text-available($href as xs:string?)
                                          as xs:boolean
```

Definition 6.176.

```
unparsed-text-available($href as xs:string?,
                $encoding as xs:string?) as xs:boolean
```

Zusätzliche Funktionen — Keys

- *Keys* machen die Arbeit mit Querverweisen einfacher.
- Implementierungen können Zugriffpfade für die deklarierten *Keys* optimieren.

Definition 6.177 (The `xsl:key` Declaration)**.**

```
<!-- Category: declaration -->
<xsl:key
  name = qname
  match = pattern
  use? = expression
  collation? = uri>
  <!-- Content: sequence-constructor -->
</xsl:key>
```

Zusätzliche Funktionen — Keys

- `xsl:key` definiert eine partielle, sequenzenwertige Funktion auf den Knoten der Eingabebäume und temporären Bäume.
- Das Pattern in `match` bestimmt den Definitionsbereich, der Ausdruck in `use` liefert die Sequenz der Werte (relativ zum Kontextknoten aus dem Definitionsbereich).

Beispiel 6.178.

```
<xsl:key name="authorId" match="author" use="@id"/>
```

It is possible to have:

- multiple `xsl:key` declarations with the same name;
- a node that matches the match patterns of several different `xsl:key` declarations, whether these have the same key name or different key names;
- a node that returns more than one value from its key specifier;
- a key value that identifies more than one node (the key values for different nodes do not need to be unique).

Zusätzliche Funktionen — Keys

Definition 6.179 (The `key` Function)**.**

```
key($key-name as xs:string,
    $key-value as xdt:anyAtomicType*) as node()*
```

Definition 6.180.

```
key($key-name  as xs:string,
    $key-value as xdt:anyAtomicType*,
    $top       as node()) as node()*
```

- `key()` invertiert die durch `xsl:key` definierten Funktionen.
- Jeder Aufruf von `key` sucht nur in einem Dokument oder temporären Baum.
- Die Ergebnissequenz ist in Dokument-Reihenfolge ohne Duplikate.

Zusätzliche Funktionen — Keys

Beispiel 6.181 (Using a Key to Follow Cross-References).

```
<xsl:key name="idkey" match="div" use="@id"/>
```

- `key("idkey", $ref)` is roughly equivalent to `(//div[@id = $ref])`

Zusätzliche Funktionen — Keys

Beispiel 6.182 (Zugriff auf Namen über Länge der Namen).

```
<xsl:template match="/">
  <xsl:variable name="doc" select="."/>
  <results>
    <xsl:for-each select="3 to 7">
      <names l="{.}">
        <xsl:value-of select="key('length', ., $doc)"/>
      </names>
    </xsl:for-each>
  </results>
</xsl:template>

<xsl:key name="length"
         match="my:data/name" use="string-length()"/>

<my:data>
  <name>Anton</name>  <name>Bert</name>
  <name>Cäsar</name>  <name>Dorian</name>
</my:data>
```

Beispiel 6.183 (Ausgabe).

```
<results>
   <names l="3"/>
   <names l="4">Bert</names>
   <names l="5">Anton Cäsar</names>
   <names l="6">Dorian</names>
   <names l="7"/>
</results>
```

Zusätzliche Funktionen — Keys

Beispiel 6.184 (mehrere Werte pro Knoten).

```
  <xsl:template match="/">
    <xsl:variable name="doc" select="."/>
    <results>
      <xsl:for-each select="3 to 7">
        <names l="{.}">
          <xsl:value-of select="key('length-ca', ., $doc)"/>
        </names>
      </xsl:for-each>
    </results>
  </xsl:template>

  <xsl:key name="length-ca" match="my:data/name"
           use="(string-length() - 1,
                 string-length(),
                 string-length() + 1)"/>
```

Beispiel 6.185 (Ausgabe).

```
<results>
   <names l="3">Bert</names>
   <names l="4">Anton, Bert, Cäsar</names>
   <names l="5">Anton, Bert, Cäsar, Dorian</names>
   <names l="6">Anton, Cäsar, Dorian</names>
   <names l="7">Dorian</names>
</results>
```

Zusätzliche Funktionen — `generate-id`

Definition 6.186.

```
generate-id() as xs:string
```

Definition 6.187.

```
generate-id($node as node()?) as xs:string
```

- The `generate-id` function returns a string that uniquely identifies a given node.
- The string is syntactically an XML name.

Zusätzliche Funktionen — `generate-id`

Beispiel 6.188 (`generate-id`).

```
generate-id(/)
generate-id(/descendant::n[1])
generate-id(/descendant::n[4])
generate-id(/descendant::n[1])
```

Beispiel 6.189 (Ausgabe Saxon).

```
d1
d1e23
d1e32
d1e23
```

Beispiel 6.190 (Ausgabe Xalan).

```
N10000
N10024
N1002D
N10024
```

Zusätzliche Funktionen — Keys, `generate-id`

Beispiel 6.191 (Using a Key to Follow Cross-References).

```
<xsl:template match="prototype">
  <p>
```

```
    <a name="{generate-id()}">
      <b>Function: </b>
      ...
    </a>
  </p>
</xsl:template>

<xsl:key name="func" match="prototype" use="@name"/>

<xsl:template match="function">
  <b>
    <a href="#{generate-id(key('func',.))}">
      <xsl:apply-templates/>
    </a>
  </b>
</xsl:template>
```

Beispiel 6.192 (Eingabe).

```
...

<prototype name="sqrt" return-type="xs:double">
  <arg type="xs:double"/>
</prototype>

...

<p>Here the function <function>sqrt</function>
is used to ...</p>

...
```

Zusätzliche Funktionen — Keys

Beispiel 6.193 (Using Keys to Reference other Documents).

```
<xsl:key name="bib" match="entry" use="@name"/>

<xsl:template match="bibref">
  <xsl:apply-templates
    select="key('bib', ., document('bib.xml'))"/>
</xsl:template>
```

Beispiel 6.194 (Source: `bib.xml`).

```
<entry name="XSLT">...</entry>
```

Zusätzliche Funktionen — Zahlen formatieren

Definition 6.195 (Number Formatting)**.**

```
format-number($value as numeric?, $picture as xs:string)
                                                as xs:string
```

```
format-number($value as numeric?,
              $picture as xs:string,
              $decimal-format-name as xs:string) as xs:string
```

Definition 6.196 (Defining a Decimal Format)**.**

```
<!-- Category: declaration -->
<xsl:decimal-format
  name? = qname
  decimal-separator? = char
  grouping-separator? = char
  infinity? = string
  minus-sign? = char
  NaN? = string
  percent? = char  per-mille? = char
  zero-digit? = char  digit? = char
  pattern-separator? = char />
```

Zusätzliche Funktionen — Zahlen formatieren

Beispiel 6.197. Variablenbindungen, Format-Definition:

```
<xsl:variable name="x" select="200000 div 3"/>
<xsl:variable name="y" select="0.653"/>

<xsl:decimal-format name="myformat"
                    decimal-separator=","
                    grouping-separator="." />
```

Funktionsaufrufe:

`format-number($x, '#,##0.00')` ⇒ `66,666.67`

`format-number($x, '#.##0,00', 'myformat')` ⇒ `66.666,67`

`format-number($y, '#.##0,00', 'myformat')` ⇒ `0,65`

`format-number($y, '#.##0,00%', 'myformat')` ⇒ `65,30%`

Zusätzliche Funktionen — Datum und Zeit formatieren (Beispiele)

Required Output	Expression
2002-12-31	`format-date($d, "[Y0001]-[M01]-[D01]")`
31 XII 2002	`format-date($d, "[D1] [MI] [Y]")`
December 31, 2002	`format-date($d, "[MNn] [D], [Y]", "en", (), ())`
einunddreißigste Dezember	`format-date($d, "[Dwo] [MNn]", "de", (), ())`
3:58:45 pm	`format-time($t, "[h]:[m01]:[s01] [Pn]", "en", (), ())`
15.58 Uhr GMT+02:00	`format-time($t,"[H01]:[m01] Uhr [z]", "de", (), ())`

Zusätzliche Funktionen —`current`

Definition 6.198 (current)**.**

```
current() as item()
```

Returns the item that was the context item at the point where the expression was invoked from the XSLT stylesheet.

Beispiel 6.199 (1, richtig)*.*

```
<xsl:apply-templates
  select="//glossary/entry[@name=current()/@ref]"/>
```

will process all entry elements that have a glossary parent element and that have a name attribute with value equal to the value of the current item's ref attribute. This is different from

Beispiel 6.200 (2, falsch)*.*

```
<xsl:apply-templates
  select="//glossary/entry[@name=./@ref]"/>
```

Beispiel 6.201 (3, falsch, äquivalent zu 2)*.*

```
<xsl:apply-templates
  select="//glossary/entry[@name=@ref]"/>
```

Zusätzliche Funktionen — `system-property`

Definition 6.202 (`system-property`)**.**

```
system-property($property-name as xs:string) as xs:string
```

Beispiel 6.203*.*

```
<p n="xsl:version">
  <xsl:value-of select="system-property('xsl:version')"/>
</p>
...
```

Beispiel 6.204 (Ausgabe)*.*

```
<p n="xsl:version">2.0</p>
<p n="xsl:vendor">Saxonica</p>
<p n="xsl:vendor-url">http://www.saxonica.com/</p>
<p n="xsl:product-name">SAXON</p>
<p n="xsl:product-version">HE 9.4.0.1</p>
<p n="xsl:is-schema-aware">no</p>
<p n="xsl:supports-serialization">yes</p>
<p n="xsl:supports-backwards-compatibility">yes</p>
```

Meldungen

Definition 6.205 (Messages)**.**

```
<!-- Category: instruction -->
<xsl:message
  select? = expression
  terminate? = { "yes" | "no" }>
  <!-- Content: sequence-constructor -->
</xsl:message>
```

- Ein Text wird gemeldet (Terminal, Pop-Up, Logfile, ...)
- Falls `terminate="yes"`, wird der XSLT-Prozessor beendet.

Beispiel 6.206.

```
  <message>I have a problem.</message>

  <message terminate="yes">
    An error was detected.
  </message>
```

6.11 Serialisierung

Serialisierung

Definition 6.207 (Serialization)**.**

```
<!-- Category: declaration -->
<xsl:output name? = qname
  method? = "xml" | "html" | "xhtml" | "text"
            | qname-but-not-ncname
  byte-order-mark? = "yes" | "no"
  cdata-section-elements? = qnames
  doctype-public? = string doctype-system? = string
  encoding? = string
  escape-uri-attributes? = "yes" | "no"
  include-content-type? = "yes" | "no"
  indent? = "yes" | "no"
  media-type? = string
  normalization-form? = "NFC" | "NFD" | "NFKC" | "NFKD"
                        | "fully-normalized" | "none" | nmtoken
  omit-xml-declaration? = "yes" | "no"
  standalone? = "yes" | "no" | "omit"
  undeclare-prefixes? = "yes" | "no"
  use-character-maps? = qnames
  version? = nmtoken />
```

Serialisierung

Beispiel 6.208 (XML).

```
<xsl:output method="xml" encoding="UTF-8" indent="yes"/>
```

Beispiel 6.209 (XHTML).

```
<xsl:stylesheet version="2.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns="http://www.w3.org/1999/xhtml">

  <xsl:output method="xhtml" encoding="UTF-8"
    indent="yes"
    doctype-public="-//W3C//DTD XHTML 1.0 Strict//EN"
    doctype-system=
      "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd"/>

  <xsl:template match="/">
    <html>...</html>
  </xsl:template>

</xsl:stylesheet>
```

Beispiel 6.210 (Ausgabe XHTML).

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
          "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  ...
</html>
```

Serialisierung

Beispiel 6.211 (HTML5). `<xsl:output method="html" version="5.0"/>`

Beispiel 6.212 (Ausgabe HTML5). `<!DOCTYPE html>`

...

Funktioniert nur mit Saxon (oder XSLT 3.0)

Serialisierung

Definition 6.213 (Character Maps)**.**

```
<!-- Category: declaration -->
<xsl:character-map
  name = qname
  use-character-maps? = qnames>
  <!-- Content: (xsl:output-character*) -->
</xsl:character-map>
```

Beispiel 6.214 (Character Map).

```
<xsl:character-map name="jsp">
  <xsl:output-character character="«" string="&lt;%"/>
  <xsl:output-character character="»" string="%&gt;"/>
  <xsl:output-character character="§" string='"'/>
</xsl:character-map>
```

Beispiel 6.215 (Gewünschte Ausgabe).

```
<jsp:setProperty name="user" property="id"
                 value='<%= "id" + idValue %>'/>
```

Beispiel 6.216 (im Stylesheet).

```
<jsp:setProperty name="user" property="id"
                 value='«= §id§ + idValue »'/>
```

Ausgabebäume

Ausgabebäume (*Final Result Trees*) werden erzeugt

- explizit mit `xsl:result-document` oder
- implizit, wenn keine `xsl:result-document`-Instruktion ausgewertet wird.

Ausgabebäume

Definition 6.217.

```
<!-- Category: instruction -->
<xsl:result-document format? = { qname }
  href? = { uri-reference }
  validation? = "strict" | "lax" | "preserve" | "strip"
  type? = qname
  method? = { "xml" | "html" | "xhtml" | "text"
              | qname-but-not-ncname }
  doctype-public? = { string } doctype-system? = { string }
  encoding? = { string }
  escape-uri-attributes? = { "yes" | "no" }
  include-content-type? = { "yes" | "no" }
  indent? = { "yes" | "no" } media-type? = { string }
  normalization-form? = { "NFC" | "NFD" | "NFKC" | "NFKD"
                          | "fully-normalized" | "none" | nmtoken }
  omit-xml-declaration? = { "yes" | "no" }
  ...
  <!-- Content: sequence-constructor -->
</xsl:result-document>
```

Ausgabebäume

- Jeder Ausgabebaum besteht aus einem neuen Dokumentknoten, darunter das Ergebnis der Auswertung eines *Sequence Constructor*.
- Das Attribut `href` spezifiziert (üblicherweise) den Namen der Datei, in die der Ausgabebaum serialisiert wird.
- Das Attribut `format` verweist auf eine `output`-Deklaration (Serialisierung) gleichen Namens.

Ausgabebäume

Beispiel 6.218 (Multiple Result Documents).

```
<xsl:stylesheet version="2.0"
    xmlns:xhtml="http://www.w3.org/1999/xhtml"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output name="toc-format" method="xhtml" indent="yes"
    doctype-system=
      "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd"
    doctype-public="-//W3C//DTD XHTML 1.0 Strict//EN"/>

  <xsl:output name="section-format" method="xhtml" indent="no"
    doctype-system=
      "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd"
    doctype-public="-//W3C//DTD XHTML 1.0 Transitional//EN"/>

  <xsl:template match="/">
    ... <!-- siehe nächste Folie -->
  </xsl:template>

</xsl:stylesheet>
```

Ausgabebäume

Beispiel 6.219 (Multiple Result Documents, continued).

```
<xsl:template match="/">
  <xsl:result-document href="toc.html" format="toc-format"
                       validation="strict">
    <html xmlns="http://www.w3.org/1999/xhtml">
      <head><title>Table of Contents</title></head>
      <body><h1>Table of Contents</h1>
        <xsl:for-each select="/*/xhtml:body/(*[1] | xhtml:h1)">
          <p><a href="section{position()}.html">
               <xsl:value-of select="."/></a></p>
        </xsl:for-each>
      </body></html>
  </xsl:result-document>
  <xsl:for-each-group select="/*/xhtml:body/*"
                      group-starting-with="xhtml:h1">
    <xsl:result-document href="section{position()}.html"
                         format="section-format" validation="strip">
      <html xmlns="http://www.w3.org/1999/xhtml">...</html>
    </xsl:result-document>
  </xsl:for-each-group>
</xsl:template>
```

6.12 Verschiedenes

Stylesheets und XML Schemas

- Bei Verwendung eines *Schema-Aware XSLT Processor*
- XSLT-Stylesheets können Schema-Informationen verwenden, z. B. in XPath-Ausdrücken.

Definition 6.220 (`xsl:import-schema`)**.**

```
<xsl:import-schema
  namespace? = uri-reference
  schema-location? = uri-reference>
  <!-- Content: xs:schema? -->
</xsl:import-schema>
```

Stylesheets und XML Schemas

Beispiel 6.221 (An Inline Schema Document).

```
<xsl:import-schema>
  <xs:schema
      targetNamespace="http://localhost/ns/yes-no">
    <xs:simpleType name="local:yes-no">
      <xs:restriction base="xs:string">
        <xs:enumeration value="yes"/>
        <xs:enumeration value="no"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:schema>
</xs:import-schema>

<xs:variable name="condition" select="'yes'"
             as="local:yes-no"/>
```

Stylesheets und XML Schemas

Beispiel 6.222 (Asserting the Required Type of the Source Document).

```
<xsl:template
     match="document-node(schema-element(my:invoice))"
     priority="2">
...
</xsl:template>

<xsl:template match="document-node()" priority="1">
  <xsl:message terminate="yes">
    Source document is not an invoice
  </xsl:message>
</xsl:template>
```

This example will cause the transformation to fail with an error message unless the document element of the source document is valid against the top-level element declaration `my:invoice`, and has been annotated as such.

XSLT-Prozessor, Konformität

Teilweise optionale Funktionalität des XSLT-Prozessors:

Basic XSLT Processor Implementiert alle Teile der Spezifikation, außer bestimmter Schema-Funktionalität.

Schema-Aware XSLT Processor Implementiert alle Teile der Spezifikation

Serialization Feature Alle Attribute von `xsl:output` und `xsl:character-map` unterstützt, alle Ausgabemethoden: `xml`, `xhtml`, `html` und `text`

Backwards Compatibility Feature Unterstützt auch XSLT 1.0

Einsatzgebiete

- XML direkt browsen mit geeigneten Browsern:
  `<?xml-stylesheet type="text/xml" href="abook.xsl"?>`
- XML auf dem Server (oder Proxy) in HTML übersetzen für ältere Browser
- Schemaübersetzung (XML → XML): Daten in anderes Format bringen
- Programmgenerierung, z. B. Java (via XSP, Cocoon) oder XSLT-Stylesheets
- ...

Weitere Interpretation der Elemente im Ausgabebaum:

- XSL-Formatierungobjekte
- HTML (bzw. XHTML)
- Beliebige XML-Daten
- Formatierungsobjekte für andere Renderer
- ...

XSLT — Werkzeuge

- Apache Xalan[95] (XSLT 1.0)
- Saxon[96]
- APIs (C, Java, JavaScript, Python, ...)
- ...

[95] http://xml.apache.org/xalan-j/index.html
[96] http://www.saxonica.com/

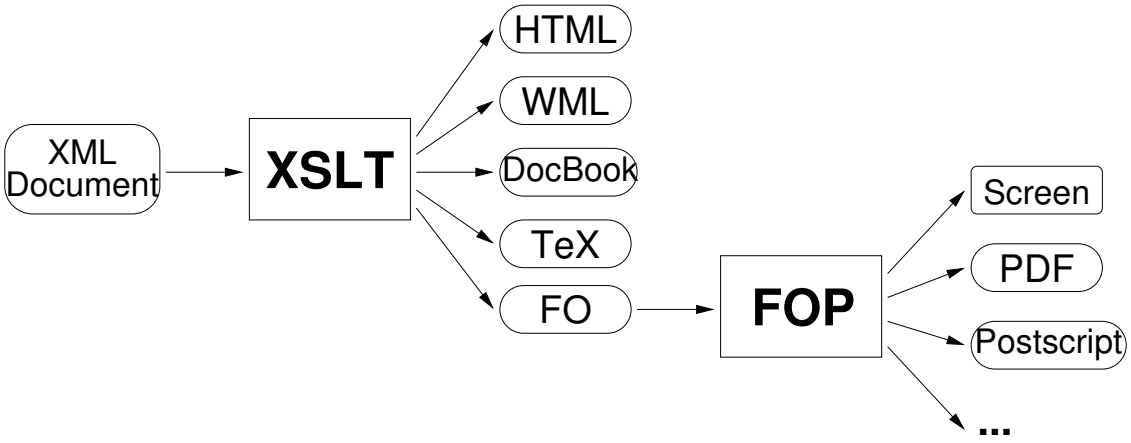
Beispiel 6.223.

```
...
TransformerFactory transFactory =
                          TransformerFactory.newInstance();
Source xsltSource = new StreamSource(xsltFile);
Transformer transformer =
                  transFactory.newTransformer(xsltSource);

DOMSource source = new DOMSource(inputFile);

File newXML = new File("newXML.xml");
FileOutputStream os = new FileOutputStream(newXML);
StreamResult result = new StreamResult(os);
transformer.transform(source, result);
...
```

Extensible Stylesheet Language (XSL)

XML-Elemente haben keine vorgegebene Präsentationssemantik. Diese Semantik wird zum Beispiel durch XSL-Stylesheets explizit angegeben.

1. Transformation (XSLT)
2. Textsatz:
   - *Formatierungsobjekte*, wie Seiten, Absätze, Tabellen, usw.
   - *Formatierungseigenschaften*, wie Einrückungen, Wort- und Zeichenabstände, Trennung, Farbe, usw.



7 XQuery und XML-Datenbanken

7.1 Übersicht

XML-Datenbanken und Anfragesprachen

XML in Datenbanken

- XML-Mapping auf relationale Datenbanken (SQL)
- XML-Erweiterungen relationaler Datenbanksysteme
- Echte/native XML-Datenbanken

- Neues Datenbankmodell $\Rightarrow$ neue adäquate Anfragesprachen benötigt
- Viele Entwicklungen: Lorel, XML-QL, XQL, Quilt, ..., auch: XSLT und XPath
- Ideen aus diesen Sprachen flossen ein in *XQuery*.

The mission of the XML Query working group is to provide flexible query facilities to extract data from real and virtual documents on the Web, therefore finally providing the needed interaction between the web world and the database world. Ultimately, collections of XML files will be accessed like databases.

Einen guten Vergleich von XQuery 1.0 und XSLT 2.0 findet man in Comparing XSLT and XQuery[97] von Michael Kay.

Konzeptionelle Ebenen

1. Datenmodell (*XQuery 1.0 and XPath 2.0 Data Model*)
2. Anfragesprache (*XQuery 1.0*)
3. XML-Datenbank (am Beispiel *eXist*)
4. Web Application Server (am Beispiel *eXist*)

7.2 XQuery 1.0

XQuery 1.0: An XML Query Language

W3C Recommendation 23. January 2007

(Second Edition 14. Dezember 2010, nur Errata eingearbeitet)

- Das *Datenmodell* definiert die Informationen, die dem XQuery-Processor zur Verfügung stehen.
- Statische und dynamische *Semantik*
- XML Schema liefert das *Typsystem*.
- *Funktionsbibliothek*: Funktionen und Operatoren
- Zwei *Syntaxen*

XQuery ist eine *funktionale* und *getypte* Sprache.

[97] http://www.saxonica.com/papers/XTech2005/mhkpaper.html

XML Query (XQuery)

Für XQuery relevante Spezifikationen[98]:

- *XQuery 1.0: An XML Query Language*
- XML Syntax for XQuery 1.0 (XQueryX)
- XML Path Language (XPath) 2.0
- XQuery 1.0 and XPath 2.0 Functions and Operators
- XQuery 1.0 and XPath 2.0 Data Model
- XSLT 2.0 and XQuery 1.0 Serialization
- XQuery 1.0 and XPath 2.0 Formal Semantics
- Diverse Requirements und Use-Cases
- ...

Einige nützliche Erweiterungen gibt es in:

- XQuery 3.1: An XML Query Language (W3C Recommendation 21 March 2017)

7.2.1 Ausdrücke

XQuery — Ausdrücke

- Alle Ausdrücke von XPath 2.0
  - Pfadausdrücke
  - Sequenzen
  - Arithmetik
  - Vergleiche
  - Filter
  - for-return
  - if-then-else
  - ...
- Konstruktoren für Elemente, Attribute, Textknoten, ...
- FLWOR Expressions (for, let, where, order by, return)
- ...

[98] http://www.w3.org/XML/Query/#specs

Rückblick: Pfadausdrücke in XPath

Beispiel 7.1 (Matrix, unvollständig).

```
<matrix>
  <row><c>1</c><c>2</c></row>
  <row><c>4</c><c>5</c><c>6</c></row>
  <row><c>7</c></row>
</matrix>
```

$$\begin{pmatrix} 1 & 2 & \\ 4 & 5 & 6 \\ 7 & & \end{pmatrix}$$

Beispiel 7.2 (Zeilenlängen zählen).

```
for $r in $matrix/row return fn:count($r/c)
```

⇒ `(2, 3, 1)`

Beispiel 7.3 (Zeilenlängen zählen, eleganter).

```
$matrix/row/fn:count(c)
```

⇒ `(2, 3, 1)`

Ein *Step* kann auch ein (beliebiger) Ausdruck sein.

Konstruktoren

Beispiel 7.4 (Elementkonstruktoren, *Direct Element Constructors*).

```
<example>
   <p> Here is a query. </p>
   <eg> $i//title </eg>
   <p> Here is the result of the query. </p>
   <eg>{ $i//title }</eg>
</example>
```

Beispiel 7.5 (Resultat).

```
<example>
   <p> Here is a query. </p>
   <eg> $i//title </eg>
   <p> Here is the result of the query. </p>
   <eg>
      <title>Hello World</title>
   </eg>
</example>
```

Konstruktoren

Beispiel 7.6 (Attributkonstruktoren).

```
<chapter ref="[{1, 5 to 7, 9}]"/>
<shoe size="As big as {$hat/@size}"/>
```

Beispiel 7.7 (Resultat). `<chapter ref="[1 5 6 7 9]"/>`
`<shoe size="As big as 11"/>`

Beispiel 7.8 (Textknoten).

```
<fact>I saw {5 + 3} cats.</fact>
```

Beispiel 7.9 (Resultat). `<fact>I saw 8 cats.</fact>`

Computed Constructors

```
"element" (QName | ("{" Expr "}")) "{" ContentExpr? "}"
```

Beispiel 7.10 (Computed Element Constructor).

```
element last { "Johnson" }
```

Beispiel 7.11 (Resultat).

```
<last>Johnson</last>
```

Beispiel 7.12 (Computed Element Constructor).

```
element
   {node-name($e)}
   {$e/@*, 2 * data($e)}
```

Beispiel 7.13 (Resultat). Für `$e` = `<length units="inches">5</length>`:

```
<length units="inches">10</length>
```

Computed Constructors

Beispiel 7.14 (Computed Attribute).

```
element test
   {attribute
      { if ($sex = "M") then "husband" else "wife" }
      { <a>Hello</a>, 1 to 3, <b>Goodbye</b> }
   }
```

Beispiel 7.15 (Resultat).

```
<test husband="Hello 1 2 3 Goodbye"/>
```

Weitere Konstruktoren für das Dokument, *CDATA Sections*, *Processing Instructions*, Kommentare.

FLWOR-Ausdrücke

For, Let, Where, Order by, Return

Beispiel 7.16 („komplett“).

```
for $d in doc("depts.xml")//deptno
let $e := doc("emps.xml")//emp[deptno = $d]
where count($e) >= 10
order by avg($e/salary) descending
return
   <big-dept>
      {
      $d,
      <headcount>{count($e)}</headcount>,
      <avgsal>{avg($e/salary)}</avgsal>
      }
   </big-dept>
```

FLWOR-Ausdrücke

```
FLWORExpr ::= (ForClause | LetClause)+ WhereClause?
              OrderByClause? "return" ExprSingle
```

Beispiel 7.17 (`let`: ohne Iteration).

```
let $s := (<one/>, <two/>, <three/>)
return <out>{$s}</out>
```

Beispiel 7.18 (Ausgabe).

```
<out>
   <one/>
   <two/>
   <three/>
</out>
```

FLWOR-Ausdrücke

Beispiel 7.19 (`for`-„Schleife“).

```
for $s in (<one/>, <two/>, <three/>)
return <out>{$s}</out>
```

Beispiel 7.20 (Ausgabe).

```
<out>
   <one/>
```

```
</out>
<out>
   <two/>
</out>
<out>
   <three/>
</out>
```

FLWOR-Ausdrücke

Beispiel 7.21 (`where`).

```
avg(for $x at $i in $inputvalues
    where $i mod 100 = 0
    return $x)
```

`$i` ist eine zu `$x` gehörende *Positional Variable*, die an die jeweilige Kontextposition gebunden ist.

Beispiel 7.22 (`order by`).

```
for $e in $employees order by $e/salary return $e/name
```

Beispiel 7.23 (`where` und `order by`).

```
for $b in $books/book
where $b/price lt 100
order by $b/title
return $b
```

7.2.2 Module

Module: Funktionsdeklarationen

Beispiel 7.24.

```
declare function local:summary($emps as element(employee)*) as element(dept)*
{
   for $d in fn:distinct-values($emps/deptno)
   let $e := $emps[deptno = $d]
   return
      <dept>
         <deptno>{$d}</deptno>
         <headcount> {fn:count($e)} </headcount>
         <payroll> {fn:sum($e/salary)} </payroll>
      </dept>
};

local:summary(fn:doc("acme_corp.xml")
              //employee[location = "Denver"])
```

Module und Prologe

- Unterscheidung *Main Module* / *Library Module*
- Eine Anfrage besteht aus genau einem *Main Module*.
- Module können *Library Modules* importieren.
- Deklarationen in Modulen: Version, Module, Boundary-space, Default Collation, Base URI, Construction, Ordering Mode, Empty Order, Copy-Namespaces, Schema Import, Module Import, Namespace, Default Namespace, Variable, Function, Option

Module und Prologe

Beispiel 7.25 (Main Module).

```
import module namespace demo="http://iai.uni-bonn.de/demo" at "libdemo.xqm";

let $x := 10
return (demo:binom1($x, 2), $demo:big, demo:binom1($demo:big, 1))
```

Beispiel 7.26 (Library Module in `libdemo.xqm`).

```
module namespace my="http://iai.uni-bonn.de/demo";

declare variable $my:big as xs:integer := 99999;

declare function my:binom1($a as xs:integer, $b as xs:integer) as xs:integer {
     $a * $a + 2 * $a * $b + $b * $b
};
```

Module und Prologe

Beispiel 7.27 (Main Module for Quicksort).

```
declare function local:qsort($l as xs:anyAtomicType*)
                                                  as xs:anyAtomicType* {
   if (count($l) gt 1) then
      ( local:qsort(subsequence($l, 2)[. lt subsequence($l, 1, 1)]),
        subsequence($l, 1, 1),
        local:qsort(subsequence($l, 2)[. ge subsequence($l, 1, 1)])
      )
   else
      $l
};

local:qsort((3,5,2,8,9,1,4,7,6))
```

⇒ `1 2 3 4 5 6 7 8 9`

7.2.3 Collections

Collections

Datenbankstruktur

Datenbank benötigt meist Zugriff auf mehrere Dokumente.

`fn:doc`

```
fn:doc($uri as xs:string?) as document-node()?
```

- Liefert ein Dokument anhand einer URI
- URI kann Fragment enthalten, kann relativ sein

`fn:collection`

```
fn:collection($arg as xs:string?) as node()*
```

- *Available Collections* im dynamischen Kontext ist eine Abbildung von Strings (URIs) auf Knotensequenzen
- `fn:collection` liefert eine solche Knotensequenz
- `fn:collection` ohne Argument liefert die *Default Collection* aus dem dynamischen Kontext
- Struktur der URI ist implementierungsabhängig.

7.2.4 Weitere Beispiele

(aus der XQuery-Spezifikation)

Hier gibt es noch mehr Beispiele: http://en.wikibooks.org/wiki/XQuery

Beispiele: XQuery-Anfragen

Beispiel 7.28 (`parts.xml`).

```
... <part>
      <partno>123</partno>
      <description>Schraube M6x30</description>
    </part> ...
```

Beispiel 7.29 (`suppliers.xml`).

```
... <supplier>
      <suppno>16</suppno>
      <suppname>IBO Baumarkt</suppname>
    </supplier> ...
```

Beispiel 7.30 (`catalog.xml`).

```
... <item>
      <partno>123</partno>
      <suppno>16</suppno>
      <price>0.05</price>
    </item> ...
```

XQuery-Anfragen

Beispiel 7.31 (Inner Join).

```
<descriptive-catalog>
{
  for $i in fn:doc("catalog.xml")/items/item,
      $p in fn:doc("parts.xml")/parts/part[partno = $i/partno],
      $s in fn:doc("suppliers.xml")/suppliers/supplier[suppno = $i/suppno]
  order by $p/description, $s/suppname
  return
     <item>
       {
        $p/description,
        $s/suppname,
        $i/price
       }
     </item>
}
</descriptive-catalog>
```

XQuery-Anfragen

Beispiel 7.32 (Left Outer Join).

```
for $s in fn:doc("suppliers.xml")/suppliers/supplier
order by $s/suppname
return
   <supplier>
      {
        $s/suppname,
        for $i in fn:doc("catalog.xml")/items/item[suppno = $s/suppno],
            $p in fn:doc("parts.xml")/parts/part[partno = $i/pno]
        order by $p/description
        return $p/description
      }
   </supplier>
```

XQuery-Anfragen

Beispiel 7.33 (Grouping).

```
for $pn in fn:distinct-values(fn:doc("catalog.xml")/items/item/partno)
let $i := fn:doc("catalog.xml")/items/item[partno = $pn]
where fn:count($i) >= 3
order by $pn
```

```
return
   <well-supplied-item>
      <partno>{$pn}</partno>
      <avgprice>{fn:avg($i/price)}</avgprice>
   </well-supplied-item>
```

XQuery-Anfragen — Recursive Transformation

Beispiel 7.34 (Table of Contents).

```
declare function local:sections-and-titles($n as node()) as node()?
 {
  if (fn:local-name($n) = "section")
  then element { fn:local-name($n) }
    { for $c in $n/* return local:sections-and-titles($c) }
  else if (fn:local-name($n) = "title")
  then $n
  else ( )
 };

local:sections-and-titles(fn:doc("cookbook.xml"))
```

XQuery-Anfragen — Recursive Transformation

Beispiel 7.35. `declare function local:swizzle($n as node()) as node() {`

```
  typeswitch($n)
    case $a as attribute(color)
      return element color { fn:string($a) }
    case $es as element(size)
      return attribute size { fn:string($es) }
    case $e as element()
      return element
        { fn:local-name($e) }
        { for $c in
            ($e/@* except $e/@color,     (: attr -> attr :)
             $e/size,                    (: elem -> attr :)
             $e/@color,                  (: attr -> elem :)
             $e/node() except $e/size )  (: elem -> elem :)
          return local:swizzle($c) }
    case $d as document-node()
      return document { for $c in $d/* return local:swizzle($c) }
    default return $n
};
```

Attribute `color` ⇒ element `color`, and element `size` ⇒ attribute `size`

XQuery-Anfragen

Beispiel 7.36 (Selecting Distinct Combinations).

```
for $p in fn:distinct-values(/orders/order/product),
```

```
$s in fn:distinct-values(/orders/order/size),
$c in fn:distinct-values(/orders/order/color)
order by $p, $s, $c
return
   if (fn:exists(/orders/order
                 [product eq $p and size eq $s and color eq $c]))
   then
      <option>
         <product>{$p}</product>
         <size>{$s}</size>
         <color>{$c}</color>
      </option>
   else ()
```

Neue Features in XQuery 3.0 und 3.1

- `group by` clause in FLWOR Expressions
- Tumbling window and sliding window in FLWOR Expressions
- `count` clause in FLWOR Expressions
- `allowing empty` in `for` clause, for functionality similar to outer joins in SQL.
- `try`/`catch` expressions
- Dynamic function call
- Inline function expressions
- Private functions
- Switch expressions
- Computed namespace constructors
- Output declarations
- Annotations
- A string concatenation operator
- Simple map operator (`!`)
- JSON

7.2.5 XML Syntax für XQuery

XQuery — XML Syntax

"One query language syntax MUST be expressed in XML in a way that reflects the underlying structure of the query."

Motivation

- Parser Reuse
- Queries on Queries
- Generating Queries
- Embedding Queries

Beispiel 7.37 (Standard-Syntax).

```
for $p in distinct(document("bib.xml")//publisher)
let $a := avg(document("bib.xml")//book[publisher = $p]/price)
return
    <publisher>
        <name>{ $p/text() }</name>
        <avgprice>{ $a }</avgprice>
    </publisher>
```

XQuery — XML Syntax

Beispiel 7.38 (Beispiel in XML-Syntax).

```
<q:query xmlns:q="http://www.w3.org/2001/06/xqueryx">
  <q:flwr>
    <q:forAssignment variable="$p">
      <q:function name="distinct">
        <q:step axis="SLASHSLASH">
          <q:function name="document">
            <q:constant datatype="CHARSTRING">bib.xml</q:constant>
          </q:function>
          <q:identifier>publisher</q:identifier>
        </q:step>
      </q:function>
    </q:forAssignment>
    <q:letAssignment variable="$a">
      <q:function name="avg">
        <q:step axis="CHILD">
          ...
        </q:step>
      </q:function>
    </q:letAssignment>
    <q:return>
      ...
    </q:return>
  </q:flwr>
</q:query>
```

XQuery — XML Syntax

Beispiel 7.39 (Beispiel in XML-Syntax, Forts.).

```
<q:query xmlns:q="http://www.w3.org/2001/06/xqueryx">
  <q:flwr>
```

```
    <q:forAssignment variable="$p">...</q:forAssignment>
    <q:letAssignment variable="$a">...</q:letAssignment>
    <q:return>
      <q:elementConstructor>
        <q:tagName>
          <q:identifier>publisher</q:identifier>
        </q:tagName>
        <q:elementConstructor>
          <q:tagName>
            <q:identifier>name</q:identifier>
          </q:tagName>
          <q:step axis="CHILD">
            <q:variable>$p</q:variable>
            <q:nodeKindTest kind="TEXT" />
          </q:step>
        </q:elementConstructor>
        <q:elementConstructor>
          <q:tagName>
            <q:identifier>avgprice</q:identifier>
          </q:tagName>
          <q:variable>$a</q:variable>
        </q:elementConstructor>
      </q:elementConstructor>
    </q:return>
  </q:flwr>
</q:query>
```

7.3 XQuery Update Facility 1.0

XQuery Update

XQuery Update Facility 1.0

- W3C Recommendation 17. März 2011
- Erweiterung von XQuery 1.0
- Ausdrücke, die Instanzen des Datenmodells persistent ändern

XQuery Update

Updates versus funktionale Sprache?

- Seiteneffekte von „Ausdrücken“ wie `insert`, `delete`, `update`
- Direkte Seiteneffekte zerstören die Semantik, Parallelausführung usw. nicht mehr möglich
- Idee: funktionale Berechnung und Update trennen (zwei Phasen)

- Ausdrücke geben Werte und/oder eine *Pending Update List* von *Update Primitives* zurück.
- Die Update List wird erst nach Berechnung des gesamten Ausdrucks einer Anfrage auf das Datenmodell angewendet.

XQuery Update

Expressions

***Basic Updating Expression*:** an insert, delete, replace, or rename expression, or a call to an updating function

***Updating Expression*:** a basic updating expression or any expression (other than a transform expression) that directly contains an updating expression

***Simple Expression*:** any XQuery expression that is not an updating expression

XQuery Update

Beispiel 7.40 (Insert).

```
insert node <year>2005</year>
   after fn:doc("bib.xml")/books/book[1]/publisher
```

Beispiel 7.41 (Delete).

```
delete nodes /email/message[fn:currentDate() - date >
                           xs:dayTimeDuration("P365D")]
```

Delete all email messages that are more than 365 days old.
Beispiel 7.42 (Replace).

```
replace node fn:doc("bib.xml")/books/book[1]/publisher
        with fn:doc("bib.xml")/books/book[2]/publisher
```

Replace the publisher of the first book with the publisher of the second book.

XQuery Update

Beispiel 7.43 (Replace).

```
replace value of node fn:doc("bib.xml")/books/book[1]/price
with fn:doc("bib.xml")/books/book[1]/price * 1.1
```

Increase the price of the first book by ten percent.
Beispiel 7.44 (Rename).

```
rename node fn:doc("bib.xml")/books/book[1]/author[1]
as "principal-author"
```

Rename the first `author` element of the first book to `principal-author`.

XQuery Update

Beispiel 7.45 (Transform).

```
for $e in //employee[skill = "Java"]
return
   copy $je := $e
   modify delete node $je/salary
   return $je
```

Return a sequence consisting of all `employee` elements that have Java as a skill, excluding their `salary` child-elements.

Transform

A transform expression is a *simple expression* because it does not modify the value of any existing nodes.

XQuery Update

Beispiel 7.46 (Transform).

```
let $oldx := /a/b/x
return
   copy $newx := $oldx
   modify (rename node $newx as "newx",
           replace value of node $newx with $newx * 2)
   return ($oldx, $newx)
```

XQuery Update

Beispiel 7.47 (FLWOR).

```
for $p in /inventory/part
let $deltap := $changes/part[partno eq $p/partno]
return
    replace value of node $p/quantity
    with $p/quantity + $deltap/quantity
```

Beispiel 7.48 (If-Then-Else).

```
if ($e/@last-updated)
then replace value of node
       $e/@last-updated with fn:currentDate()
else insert node
       attribute last-updated {fn:currentDate()} into $e
```

Beispiel 7.49 (Comma).

```
let $q := /inventory/item[serialno = "123456"]/quantity
return
   ( replace value of node $q with ( ),
     insert node attribute xsi:nil {"true"} into $q )
```

XQuery Update

Variablen

- In Variablendefinitionen keine Updates

Funktionen

- Ergebnis: *Entweder* Werte *oder* Updates

Beispiel 7.50 (Updating Function).

```
declare updating function
   upsert($e as element(),
          $an as xs:QName,
          $av as xs:anyAtomicType)
   {
   let $ea := $e/attribute()[fn:node-name(.) = $an]
   return
      if (fn:empty($ea))
      then insert node attribute $an $av into $e
      else replace value of node $ea with $av
   }
```

XQuery Update

Update Primitives

- `upd:insertBefore`, `upd:insertAfter`
- `upd:insertInto`
- `upd:insertIntoAsFirst`, `upd:insertIntoAsLast`
- `upd:insertAttributes`
- `upd:delete`
- `upd:replaceNode`, `upd:replaceValue`
- `upd:replaceElementContent`, `upd:rename`
- *`upd:put`* (stores a document or element to a location)

Update Routines

- *`upd:mergeUpdates`*
- *`upd:applyUpdates`*
- ...

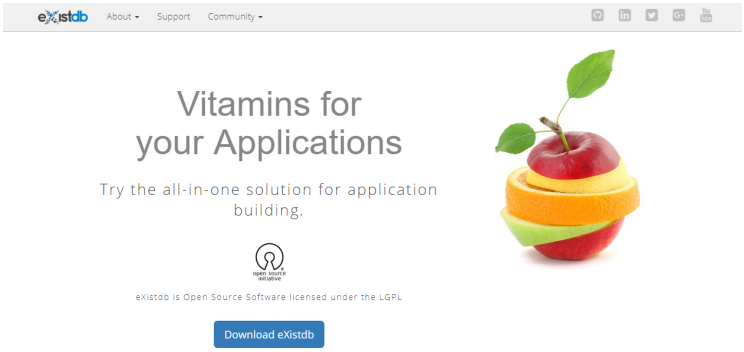
XQuery Update

Fazit

- Vollständiger Satz von Update-Instruktionen
- In die Ausdrücke integriert ...
- ... mit sauberer Semantik

7.4 eXist

Open Source XML DBMS: eXist



Das eXist-System

- entwickelt seit 2001
- Open Source, Java
- XPath- und Fulltext-Indexe
- integrierter Webserver
- REST, WebDAV, SOAP, XMLRPC, ...

Download von http://www.exist-db.org[99]

[99] http://www.exist-db.org

Installieren und Starten von eXist

Installation

- Download des Installations-Jar-Files[100] von der Webseite
- Installieren mit `java -jar eXist-setup-Version.jar`

Starten als...

Webserver Aufruf von `bin/startup.sh`

Browser `http://localhost:8080/exist/`

7.4.1 Datenbank

Datenbanken

Eigenschaften von (klassischen) Datenbanken

- Datenbanksprache zur Definition, Anfrage und Manipulation von Daten
- Transaktionen, Commit und Rollback, ACID
- Integritätsbedingungen
- Authentifikation und Autorisation
- Anwendungen, Reports, ...
- Einbettung von z. B. SQL in Programmiersprachen, Programmierschnittstellen

Eigenschaften von eXist

- Anfrage: XQuery, *optimized index-based XQuery engine*
- Manipulation: Update Extensions
- Data Definition: XML Schema
- Transaktionen implizit, *full crash recovery*
- Integrität: XML Schema
- Autorisation: *Unix-like access permissions for users/groups at collection- and document-level.*
- Security: *eXtensible Access Control Markup Language (XACML)*
- Web-Anwendungen und Web-Services können komplett in XQuery erstellt werden

[100] `http://prdownloads.sourceforge.net/exist/eXist-1.1.1-newcore-build4311.jar`

eXist als XML-Datenbank

Datenmodell

- Daten sind in einzelnen XML-*Dokumenten* gespeichert
- *Collections* sind Sammlungen von XML-Dokumenten
- Hierarchische Anordnung von Collections (wie Filesystem)
- Absolute und relative Adressierung

eXist: Dokumente und Collections

Beispiel 7.51 (Adressierung von Dokumenten).

`fn:doc('/db/web/abook.xml')` absoluter Pfad in der DB zum Dokument

`fn:doc('abook.xml')` Pfad relativ zu XQuery Base URI (abhängig von der Zugriffsmethode)

`fn:doc('web/abook.xml')` Subcollection "web" relativ zu Base URI

`fn:doc('http://w3c.org')` externe URI-Adressen

Achtung

Pfade hier sind weder XPath- noch Filesystem-Pfade, sondern „Adressen" in der eXist-Datenbank. Ausnahme: `fn:doc('file:///tmp/test.xml')`, usw.

Beispiel 7.52 (Adressierung von Collections).

`fn:collection('web')` alle Dokumente in der Collection 'web' und allen Subcollections

`fn:collection('/db/web/backup')` nur die Dokumente der Subcollection 'backup'

Beispiel 7.53 (direkte Adressierung von Knoten).

`fn:doc('abook.xml')//ort` alle Ort-Elemente im Dokument

`//ort` alle Ort-Elemente in der Default Collection

`fn:collection('/db/web/backup')//ort` alle Ort-Elemente in der Subcollection

Default Collection

- gehört zum *XQuery Dynamic Context*
- wird durch die Implementierung (hier: eXist) gesetzt

XQuery-Erweiterungen

XQuery Extension Modules

- Example Module
- Compression Module
- Date Time Module
- HTTP Client Module
- Image Module
- Mail Module
- Math Module
- Scheduler Module
- Simple Query Language Module
- Spatial module
- SQL Module
- XML Differencing Module
- XSL-FO Module
- ...

XQuery-Erweiterungen

Beispiel 7.54 (XSLT-Transformation, internes Stylesheet).

```
declare namespace xsl="http://www.w3.org/1999/XSL/Transform";
let $stylesheet :=
     <xsl:stylesheet version="2.0">
      <xsl:template match="person">
       <mensch><xsl:apply-templates select="node()|@*"/></mensch>
      </xsl:template>
      <xsl:template match="*|@*">
       <xsl:copy>
        <xsl:apply-templates select="node()|@*"/>
       </xsl:copy>
      </xsl:template>
     </xsl:stylesheet>,
    $input :=
     <data><person name="Homer"/><haus nummer="164"/></data>
return
    transform:transform($input, $stylesheet, <parameters/>)
```

Beispiel 7.55 (Ausgabe).

```
<data><mensch name="Homer"/><haus nummer="164"/></data>
```

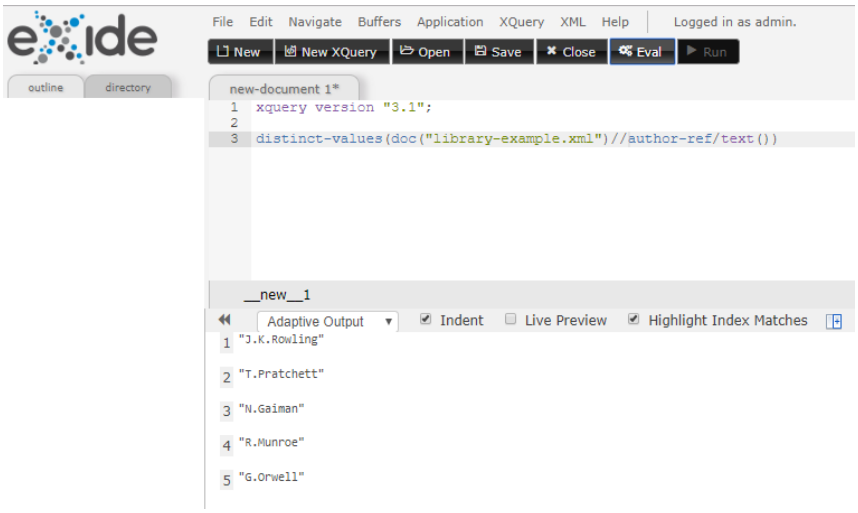
Beispiel 7.56 (XSLT-Transformation, externes Stylesheet).

```
transform:transform($input, "convert.xsl", <parameters/>)
```

XQuery-Anfragen in eXist

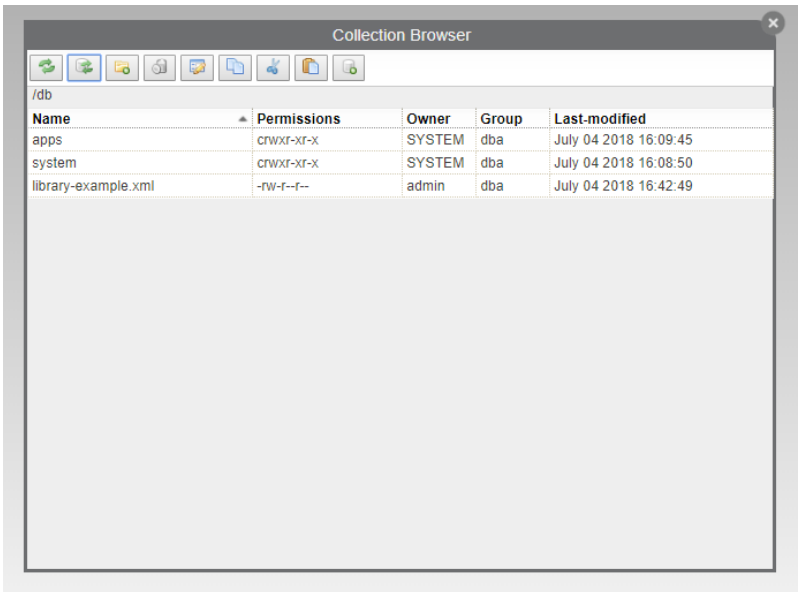
Anfragemöglichkeiten

- im Java-Client (embedded oder remote)
- interaktiv in eXide (XQuery-Sandbox)
- als XQuery-Servlet, via XML-RPC, XML:DB API...



eXist

Beispiel: Web-basierte Datenbankverwaltung



7.4.2 Application Server

Writing Web Applications using XQuery[101]

Anwendungsentwicklung mit eXist (1)

XQuery als Programmiersprache

- XQuery: komplette funktionale Programmiersprache.
- Auch zur Anwendungsprogrammierung geeignet.
- XQuery-Funktionsbibliothek + eXist-Erweiterungen
- Analogie: JSP oder PHP oder ... *plus SQL*
- Vorteil XQuery: alles in einer Sprache, ggf. plus XSLT
- Auch AJAX-Applikationen (Asynchronous JavaScript and XML) sehr einfach zu erstellen.

Verschiedene Szenarien

- *XQueryServlet* führt Xquery-Module, die im Filesystem des Servers liegen, in einer Servlet-Umgebung aus.
- *REST Server*
- ...

Anwendungsentwicklung mit eXist (2)

XQueryServlet

The XQueryServlet reads an XQuery script from the file system, usually from the directory in which the web application resides, and executes it with the current HTTP context. The result of the query is returned as the content of the HTTP response.

REST Server

The REST servlet can be used to execute stored XQueries on the server. If the target resource of a GET or POST request is a binary document with mime-type application/xquery, the REST server will try to load and execute this resource with the current HTTP context.

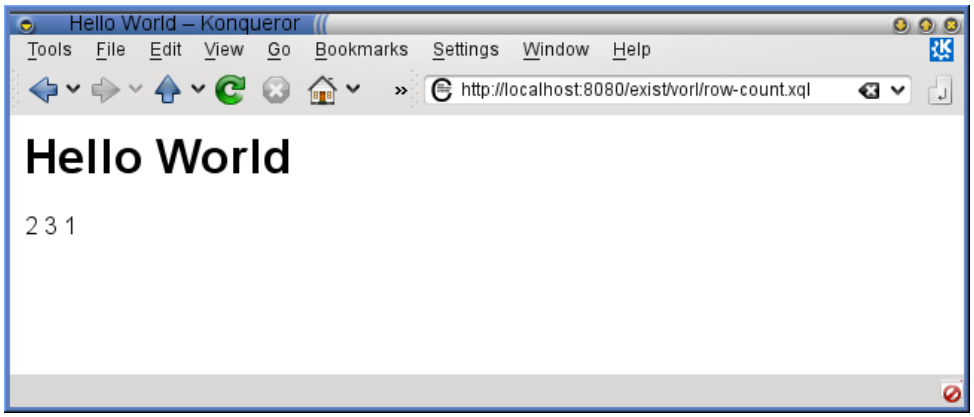
[101] http://www.exist-db.org/devguide_xquery.html

Anwendungsentwicklung mit eXist (3)

Beispiel 7.57 (Einfache Anfrage: `row-count.xql`).

```
declare namespace my = "http://example.org";

declare variable $my:mat1 as element(matrix) :=
        <matrix>
          <row><c>1</c><c>2</c></row>
          <row><c>4</c><c>5</c><c>6</c></row>
          <row><c>7</c></row>
        </matrix>;

let $my:result := $my:mat1/row/fn:count(c)
return
  <html>
    <head><title>Hello World</title></head>
    <body>
      <h1>Hello World</h1>
      <p>{$my:result}</p>
    </body>
  </html>
```

```
$ cd /opt/eXist/webapp/
$ ls
acknowledge.xml                     journal.xml
admin                               kwic.xml
...
documentation.xml                   vorl
...
irclog                              xquery.xml
jmx.xml
$ cd vorl
$ ls
books2html.xsl   isbn.xql          redir2.xql
books.xql        matrices.xqm      row-count.xql
evaltest         matrix-ops.xqm    simple.html
exist-howto.txt  matrix-test.xql   transform-example.xql
guess.xql        permute.xql
Images           redir1.xql
$
```

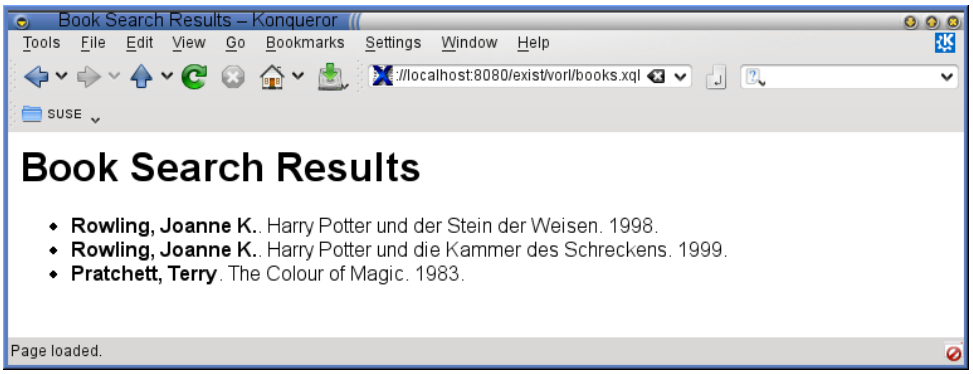


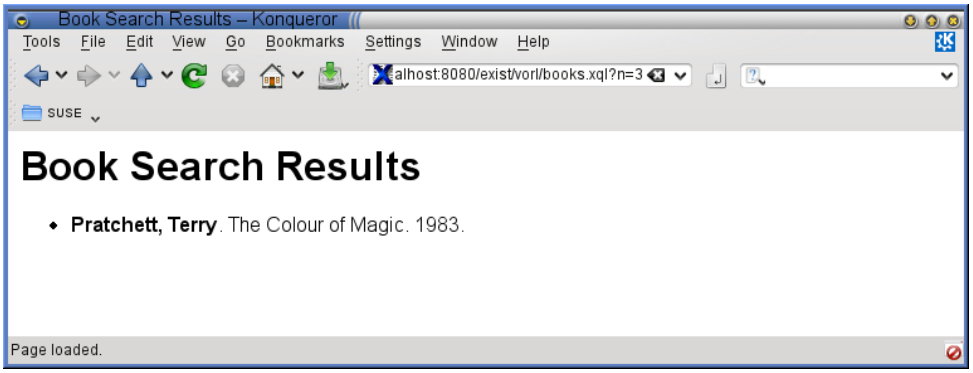
Separation of Concerns

- Bislang: Logik und Präsentation (HTML) gemischt
- Besser trennnen!

Beispiel 7.58 (View mit XSLT).

```
import module namespace transform="http://exist-db.org/xquery/transform";
import module namespace request="http://exist-db.org/xquery/request";

let $lib := doc("/db/demo/library.xml")
let $num := xs:integer(request:get-parameter("n", ()))
let $books :=
  <book-list>
    {if ($num) then $lib//book[position() eq $num]
     else $lib//book}
  </book-list>
return
  transform:transform($books, "books2html.xsl", <parameters/>)
```



Zusammenfassung

- Anfragesprache, Datenbank, Anwendung
- Alles mit einer Sprache: XQuery
- Plattform: eXist

Ausblick

- Viele weitere Möglichkeiten
- Lernkurve relativ flach (leichter Einstieg)
- Ausprobieren!

8 Programmierschnittstellen

XML-Programmierschnittstellen

- Simple API for XML (SAX)[102] ⇒ *Ereignisorientiert, Push*
- Document Object Model (DOM)[103] ⇒ *Baumdarstellung*
- Transformation API for XML (TrAX)[104] ⇒ u. a. *XSLT*-Einbindung
- APIs zum Parsen, Bearbeiten und Serialisieren von XML-Dokumenten
- Java Architecture for XML Binding (JAXB)
- Einige der APIs sind sprachunabhängig

8.1 SAX

Simple API for XML (SAX) Version 2.0

- Ein *ereignisorientiertes Interface* meldet Ereignisse während des Parsens (Start eines Elements, Ende eines Elements, usw.) an das Anwendungsprogramm mittels *Call-Backs*.
- SAX-Anwendungen bauen üblicherweise keinen (kompletten) Baum im Speicher auf.
  - Beispiel: Die einzelnen Daten werden sofort in eine Datenbank geschrieben.
  - Beispiel: Man sucht nur nach einem Datensatz in einem großen Dokument.

[102] http://www.saxproject.org/
[103] http://www.w3.org/DOM/
[104] http://java.sun.com/webservices/reference/tutorials/jaxp/html/xslt.html#gchlx

- Ein ereignisorientiertes API ist einfacher als ein baumbasiertes, ist aber auch *low-level*.
- Die Java-APIs sind so angelegt, daß der Parser ausgetauscht werden kann, ohne das Anwendungsprogramm neu zu übersetzen oder gar ändern zu müssen.

Ereignisorientiertes Parsing

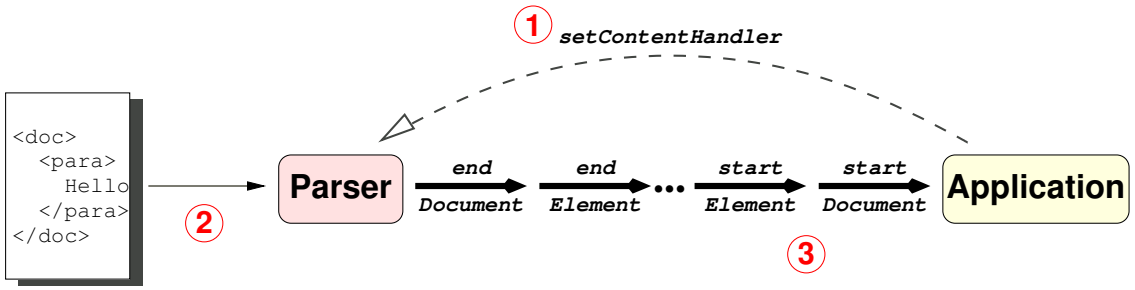
Beispiel 8.1 (XML-Dokument).

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

<doc><para>Hello, world!</para></doc>
```

Beispiel 8.2. Ereignisse beim Parsen:

1. start document
2. start element: `doc`
3. start element: `para`
4. characters: `Hello, world!`
5. end element: `para`
6. end element: `doc`
7. end document

Registrierung und Event-Fluss



1. Der *Event Handler* registriert sich beim Erzeuger der Events (hier: Parser).
2. Der Parser wird gestartet und liest das Dokument.
3. Dabei ruft der Parser die Call-Back-Methoden des Event-Handlers auf.

SAX — Beispiel

Beispiel 8.3 (Minimales SAX-Programm: Set-Up).

```
import org.xml.sax.XMLReader;
import org.xml.sax.Attributes;
import org.xml.sax.helpers.XMLReaderFactory;
import org.xml.sax.helpers.DefaultHandler;

public class XmlTree1 extends DefaultHandler {

  public static void main (String args[]) throws Exception {
    XMLReader parser = XMLReaderFactory.createXMLReader();
    XmlTree1 app = new XmlTree1();
    parser.setContentHandler(app);
    parser.setErrorHandler(app);

    // Parse each file provided on the command line.
    for (int i = 0; i < args.length; i++) {
      parser.parse(args[i]);
    }
  }
  ...
}
```

Die benötigten Klassen sind im JDK ab 1.4 enthalten.

Beispiel 8.4 (Übersetzen des Beispiels). `javac XmlTree1.java`

Beispiel 8.5 (Ausführen des Beispiels). `java XmlTree1 test1.xml`

Beispiel 8.6 (Ausführen mit expliziter Angabe der Parser-Klasse).

```
java -Dorg.xml.sax.driver=org.apache.xerces.parsers.SAXParser\
     XmlTree1 test1.xml
```

Beispiel 8.7 (Handling events).

```
  public void startDocument() {
    System.out.println("Start document");
  }

  public void endDocument() {
    System.out.println("End document");
  }

  public void startElement(String uri, String name, String qName,
                           Attributes attrs) {
    System.out.println("Start: " + name + " {" + uri + "}");
  }

  public void endElement(String uri, String name, String qName) {
    System.out.println("End:   " + name + " {" + uri + "}");
  }

  ...
```

Beispiel 8.8 (Handling events, Forts.).

```
  public void characters(char ch[], int start, int length) {
    System.out.print("Chars: \"");
    for (int i = start; i < start + length; i++) {
      switch (ch[i]) {
      case '\\':
        System.out.print("\\\\");  break;
      case '"':
        System.out.print("\\\"");  break;
      case '\n':
        System.out.print("\\n");   break;
      case '\r':
        System.out.print("\\r");   break;
      case '\t':
        System.out.print("\\t");   break;
      default:
        System.out.print(ch[i]);
      }
    }
    System.out.print("\"\n");
  }
```

Beispiel 8.9 (Testdaten).

```
<a>
  <b xmlns="http://test">
    <c1>Hello</c1>
    <c2>World</c2>
  </b>
  <d>
    <p:c3 xmlns:p="http://test"/>
    <q:c4 xmlns:q="http://test">
      <e/>
      <f/>
      <g/>
    </q:c4>
  </d>
</a>
```

Beispiel 8.10 (Testlauf).

```
Start document
Start: a {}
Chars: "\n  "
Start: b {http://test}
Chars: "\n     "
Start: c1 {http://test}
Chars: "Hello"
End:   c1 {http://test}
Chars: "\n     "
Start: c2 {http://test}
Chars: "World"
End:   c2 {http://test}
```

```
Chars: "\n  "
End:   b {http://test}
Chars: "\n  "
Start: d {}
Chars: "\n     "
Start: c3 {http://test}
End:   c3 {http://test}
Chars: "\n     "
Start: c4 {http://test}
Chars: "\n       "
Start: e {}
End:   e {}
Chars: "\n       "
Start: f {}
End:   f {}
Chars: "\n       "
Start: g {}
End:   g {}
Chars: "\n     "
End:   c4 {http://test}
Chars: "\n  "
End:   d {}
Chars: "\n"
End:   a {}
End document
```

Struktur von SAX-Programmen

- Aufgrund der Call-Back-Struktur der ereignisorientierten Schnittstelle ist ein „normaler“ Kontrollfluss des Programms nicht mehr gegeben.
- Der Zustand des Programms muss daher i. Allg. Kontrollkomponenten enthalten.
- Objektorientierte Programmierung ist dafür gut geeignet.

SAX — Erweitertes Beispiel

Beispiel 8.11 (Gewünschte Ausgabe).

```
Start document
1. Start: a {}
   1. Chars: "\n  "
   2. Start: b {http://test}
      1. Chars: "\n    "
      2. Start: c1 {http://test}
         1. Chars: "Hello"
      2. End:   c1 {http://test}
      3. Chars: "\n    "
      4. Start: c2 {http://test}
         1. Chars: "World"
      4. End:   c2 {http://test}
      5. Chars: "\n  "
```

```
   2. End:   b {http://test}
   3. Chars: "\n  "
   4. Start: d {}
      1. Chars: "\n    "
      2. Start: c3 {http://test}
      2. End:   c3 {http://test}
      3. Chars: "\n    "
      4. Start: c4 {http://test}
         ...
End document
```

Beispiel 8.12 (Zustand in Objektvariablen speichern).

```
public class XmlTree2 extends DefaultHandler {

  private Stack<Integer> stack = new Stack<Integer>();
  private int depth = 0;

  ...

  // Methode zum Einrücken
  private void indent() {
    for (int i = 0; i < depth; i++)
      System.out.print("   ");
  }

  ...
```

Für dieses Programm wird das JDK 1.5 benötigt.
Beispiel 8.13 (Dokumentebene).

```
  ...

  public void startDocument() {
    stack.push(0);
    System.out.println("Start document");
  }

  public void endDocument() {
    stack.pop();
    System.out.println("End document");
  }

  ...
```

Beispiel 8.14 (Elementebene).

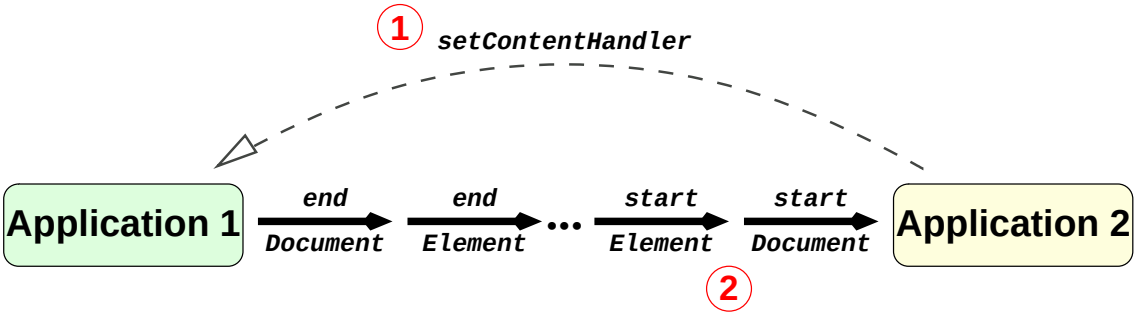
```
  ...

  public void startElement(String uri, String name, String qName,
                           Attributes attrs) {
    int n = stack.pop() + 1;
    stack.push(n);
    indent();
```

```
    System.out.println(n + ". Start: " + name + " {" + uri + "}");
    depth++;
    stack.push(0);
  }

  public void endElement(String uri, String name, String qName) {
    stack.pop();
    depth--;
    int n = stack.peek();
    indent();
    System.out.println(n + ". End:   " + name + " {" + uri + "}");
  }

  ...
```

Beispiel 8.15 (Textknoten).

```
  public void characters(char ch[], int start, int length) {
    int n = stack.pop() + 1;
    stack.push(n);
    indent();
    System.out.print(n + ". Chars: \"");
    for (int i = start; i < start + length; i++) {
      switch (ch[i]) {
      case '\\':
        System.out.print("\\\\");  break;
      case '"':
        System.out.print("\\\"");  break;
      case '\n':
        System.out.print("\\n");  break;
      case '\r':
        System.out.print("\\r");  break;
      case '\t':
        System.out.print("\\t");  break;
      default:
        System.out.print(ch[i]);
      }
    }
    System.out.print("\"\n");
  }
```

SAX-Kommunikation zwischen Programmteilen



- Der Event-Erzeuger ist nicht immer ein Parser.
- Komplexere Kommunikationsstrukturen sind möglich: Ketten, Verzweigungen

SAX — APIs

- Relevant für SAX 2.0 ist das Interface ***`org.xml.sax.ContentHandler`***.
- Wird implementiert z. B. von der Klasse ***`org.xml.sax.helpers.DefaultHandler`***.
- Alle Methoden von `DefaultHandler` sind leer.
- Programmierer können in Unterklassen von `DefaultHandler` einzelne Methoden überschreiben.

Interface ContentHandler

```
void startDocument()
void endDocument()

void startPrefixMapping(String prefix, String uri)

void endPrefixMapping(String prefix)

void startElement(String uri, String localName,
                  String qName, Attributes attributes)

void endElement(String uri, String localName,  String qName)

void characters(char[] ch, int start, int length)

void ignorableWhitespace(char[] ch, int start, int length)

void processingInstruction(String target, String data)

void skippedEntity(String name)

void setDocumentLocator(Locator locator)
```

Interface ErrorHandler

```
void error(SAXParseException e)

void fatalError(SAXParseException e)

void warning(SAXParseException e)
```

Interface DTDHandler

```
void notationDecl(String name, String publicId,
                  String systemId)

void unparsedEntityDecl(String name,
                        String publicId,
                        String systemId,
                        String notationName)
```

Serialisierung

Die Ausgabe von XML-Dokumenten mit `println` o. ä. ist sehr fehleranfällig: Probleme mit Encoding, Escaping, Namespaces, usw.

```
...
String aValue = "Jim said: \"1 < 2\"";
String text = "<<é>>";
```

Beispiel 8.16.

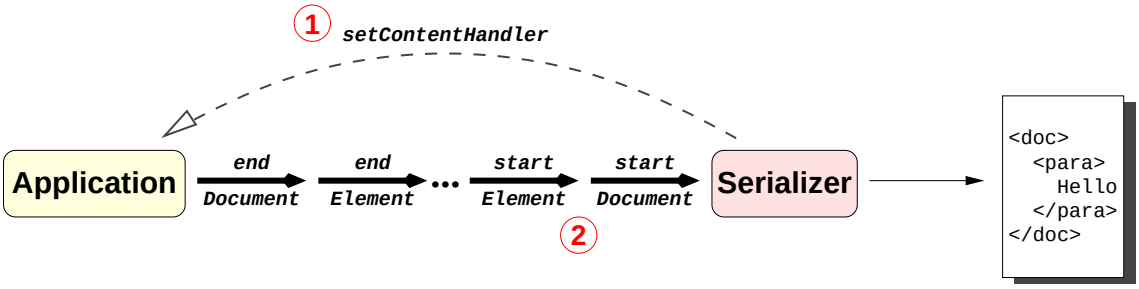
```
...
System.out.println("<p a=\"" + aValue + "\">");
System.out.println(text);
System.out.println("</p>");
...
```

Beispiel 8.17 (Ausgabe).

```
<p a="Jim said: "1 < 2"">
<<Ã©>>
</p>
```

Serialisierung

Besser: zuverlässigen *XML-Serialisierer* verwenden.



Für den Event-Erzeuger ist der Serialisierer ein normaler *ContentHandler*.

Serialisierung

Beispiel 8.18.

```
import com.sun.org.apache.xml.internal.serialize.XMLSerializer;
import com.sun.org.apache.xml.internal.serialize.OutputFormat;

public class SerializeExample {
  private static final String MY_URI = "http://iai.uni-bonn.de/xml/test";
  private static final String MY_PREFIX = "t";

  public static void main (String args[]) throws Exception {
    OutputFormat format = new OutputFormat("xml", "ISO-8859-1", true);
    format.setLineWidth(0);
    format.setPreserveSpace(true);
    XMLSerializer output = new XMLSerializer(System.out, format);

    output.startDocument();
    output.startPrefixMapping(MY_PREFIX, MY_URI);
    output.startElement("", "test-doc", null, null);
    output.startElement(MY_URI, "hallo", null, null);
    output.endElement(MY_URI, "hallo", null);
    output.endElement("", "test-doc", null);
    output.endPrefixMapping(MY_PREFIX);
    output.endDocument();
  }
}
```

Beispiel 8.19 (Ausgabe).

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<test-doc xmlns:t="http://iai.uni-bonn.de/xml/test"><t:hallo/></test-doc>
```

- Die Klasse `org.apache.xml.serialize.XMLSerializer` ist im *Xerces*-Paket vom *Apache XML Project* enthalten.

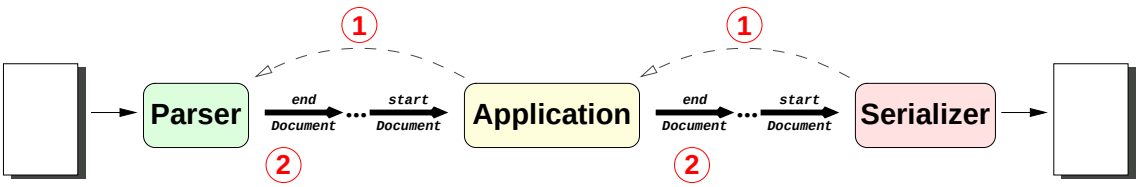
  Muss aber erst installiert werden...

- Die Klasse `com.sun.org.apache.xml.internal.serialize.XMLSerializer` ist im JDK enthalten (jedenfalls bis 1.8).

  Ist aber *deprecated*...

- Alternative: `javax.xml.transform.Transformer` mit `SAXSource` und `StreamResult` (später)

Pipelines und Filter



- Kombination der zuvor behandelten Szenarien.

SAX — Zusammenfassung

- Einfaches *ereignisorientiertes API*
- Programmierer muss den ungewohnten Kontrollfluss berücksichtigen.
- Gut geeignet für viele Anwendungen, zum Beispiel Datenextraktion
- Geeignet als Basis für Systeme, die Bäume aufbauen (z. B. DOM)

8.2 DOM

Document Object Model (DOM) Level 2

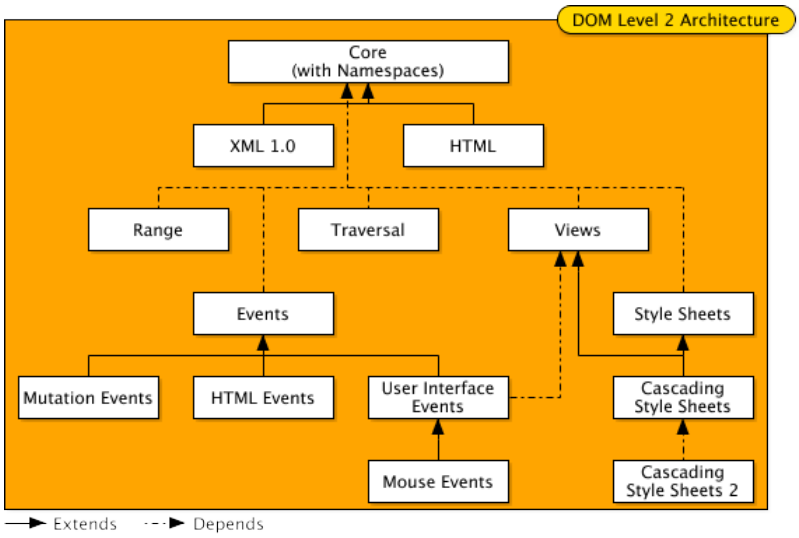
W3C Recommendations[105] (November 2000)

Auch: Document Object Model Level 3

W3C Recommendation (April 2004)

- Standard-API für XML (und HTML, SVG, usw.)
- Modell weitgehend sprachunabhängig (Java, Perl, Python, usw.)
- Weitere Motivation: HTML, SVG, usw. dynamisch verändern (mit JavaScript).
- Sehr umfangreich: DOM Core, DOM XML, DOM HTML, DOM Events, DOM CSS, DOM Load and Save, DOM Abstract Schemas, DOM XPath

Document Object Model (DOM) Level 2



[105] http://www.w3.org/DOM/DOMTR

DOM — Datenstrukturen

- Zentrale Datenstruktur ist ein Baum.
- Die Knoten haben das Interface ***`org.w3c.dom.Node`*** bzw. Sub-Interfaces davon.
- Sub-Interfaces von `Node`: `Attr`, `CDATASection`, `CharacterData`, `Comment`, `Document`, `DocumentFragment`, `DocumentType`, `Element`, `Entity`, `EntityReference`, `Notation`, `ProcessingInstruction`, `Text`

Beispielprogramm

Beispiel 8.20 (`DOMExample1.java`).

```
import java.io.*;
import org.apache.xerces.parsers.DOMParser;
import org.w3c.dom.Attr;
import org.w3c.dom.Document;
import org.w3c.dom.NamedNodeMap;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;
import org.w3c.dom.Element;

public class DOMExample1 {

    public static void main (String args[]) throws Exception {
        DOMParser parser = new DOMParser();
        parser.parse(args[0]);
        Document document = parser.getDocument();
        traverse(document, 0);
    }
```

Beispiel 8.21 (`DOMExample1.java`, Forts.).

```
    public static void traverse(Node node, int level) {
        if (node == null) { return; }
        int type = node.getNodeType();
        switch (type) {
        case Node.DOCUMENT_NODE:
            indent(level);
            System.out.println("DOCUMENT_NODE");
            traverse(((Document) node).getDocumentElement(),
                     level + 1);
            break;
        case Node.ELEMENT_NODE:
            indent(level);
            System.out.println("ELEMENT_NODE: "
                               + ((Element) node).getTagName());
            Node child = node.getFirstChild();
            while (child != null) {
                traverse(child, level + 1);
                child=child.getNextSibling();
            }
            break;
```

Beispiel 8.22 (`DOMExample1.java`, Forts.).

```
        case Node.TEXT_NODE:
            indent(level);
            System.out.println("TEXT_NODE: \""
                               + node.getNodeValue() + "\"");
            break;
        }
    }

    private static void indent(int level) {
        for (int i = 1; i <= level; i++)
            System.out.print("    ");
    }
}
```

Beispiel 8.23 (XML-Eingabe).

```
<fieldnames xml:lang="de">
        <fieldname name="lname">Familienname</fieldname>
        <fieldname name="fname">Vorname</fieldname>
        <fieldname name="interests">Hobbies</fieldname>
</fieldnames>
```

Beispiel 8.24 (Ausgabe).

```
DOCUMENT_NODE
    ELEMENT_NODE: fieldnames
        TEXT_NODE: "
        "
        ELEMENT_NODE: fieldname
            TEXT_NODE: "Familienname"
        TEXT_NODE: "
        "
        ELEMENT_NODE: fieldname
            TEXT_NODE: "Vorname"
        TEXT_NODE: "
        "
        ELEMENT_NODE: fieldname
            TEXT_NODE: "Hobbies"
        TEXT_NODE: "
"
```

Weiteres Beispiel

Beispiel 8.25 (Attribute verarbeiten).

```
    case Node.ELEMENT_NODE:
        indent(level);
        System.out.println("ELEMENT_NODE: "
                           + ((Element) node).getTagName());
        NamedNodeMap attrs = node.getAttributes();
        if (attrs != null) {
```

```
            int len = attrs.getLength();
            for (int i = 0; i < len; i++) {
                Attr a = (Attr) attrs.item(i);
                indent(level + 1);
                System.out.println("- ATTR: " + a.getName()
                                   + "=\"" + a.getValue() + "\"");
            }
        }
        Node child = currentNode.getFirstChild();
        while (child != null) {
            traverse(child, level + 1);
            child=child.getNextSibling();
        }
        break;
...
```

Beispiel 8.26 (XML-Eingabe).

```
<fieldnames xml:lang="de">
        <fieldname name="lname">Familienname</fieldname>
        <fieldname name="fname">Vorname</fieldname>
        <fieldname name="interests">Hobbies</fieldname>
</fieldnames>
```

Beispiel 8.27 (Ausgabe).

```
DOCUMENT_NODE
    ELEMENT_NODE: fieldnames
        - ATTR: xml:lang="de"
        TEXT_NODE: "
        "
        ELEMENT_NODE: fieldname
            - ATTR: name="lname"
            TEXT_NODE: "Familienname"
        TEXT_NODE: "
        "
        ELEMENT_NODE: fieldname
            - ATTR: name="fname"
            TEXT_NODE: "Vorname"
        TEXT_NODE: "
        "
        ELEMENT_NODE: fieldname
            ...
```

JAXP: DOM Plugability

Beispiel 8.28 (JAXP für DOM).

```
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.DocumentBuilder;

DocumentBuilder builder;
DocumentBuilderFactory factory =
               DocumentBuilderFactory.newInstance();
```

```
factory.setNamespaceAware(true);
// factory.setValidating(true);
String location = "http://myserver/mycontent.xml";
try {
  builder = factory.newDocumentBuilder();
  Document document = builder.parse(location);
} catch (SAXException se) {
  // handle error
} catch (IOException ioe) {
  // handle error
} catch (ParserConfigurationException pce) {
  // handle error }
```

DOM — Zusammenfassung

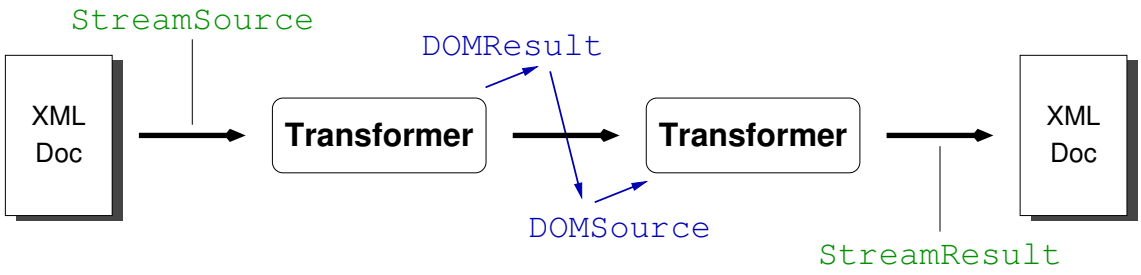
- Komplexe baumbasierte APIs
- Für Anwendungen, die das ganze Dokument im Zugriff haben müssen
- Viele weitere Funktionalität
- „Plug-In“ Parser und Serialisierer
- Mehr in DOM Level 3: Validierung, Load and Save, XPath, Events, ...
- Weitere DOMs für MathML 2.0, SMIL Animation, SVG, ...

8.3 TrAX

Transformation API for XML (TrAX)

- Verschiedene Transformationen
- *javax.xml.transform.TransformerFactory*
- *javax.xml.transform.Transformer*
- `transformer.transform(source, result);`
- Datenquellen: *javax.xml.transform.Source*
  - `SAXSource`
  - `DOMSource`
  - `StreamSource`
- Datensenken: *javax.xml.transform.Result*
  - `SAXResult`
  - `DOMResult`
  - `StreamResult`

TrAX – Beispiel



- Beispiel: Beliebig viele Transformationen hintereinander ausführen
- Mit Saxon: `saxon -s:input.xml t1.xsl | saxon -s:- t2.xsl > out.xml`
  Nachteil: Parsen/Serialisieren der Zwischenergebnisse
- Unser Beispiel: `java XSLTPipeline input.xml t1.xsl t2.xsl ... out.xml`
- Beispiel aus Platzgründen ohne Fehlerbehandlung

TrAX

Beispiel 8.29 (Setup).

```
public static void main(final String[] args) {
  Transformer transformer;
  Result result;
  DOMResult documentResult;

  File inputFile = new File(args[0]);
  File outputFile = new File(args[args.length - 1]);

  Source source = new StreamSource(inputFile);
  TransformerFactory transformerFactory =
                    TransformerFactory.newInstance();
  int lastXSLT = args.length - 2; // no. of real transforms
  if(lastXSLT == 0) {
    result = new StreamResult(outputFile);
    // identity transformer:
    transformer = transformerFactory.newTransformer();
    transformer.transform(source, result);
  } else {
    ...
```

TrAX

Beispiel 8.30 (Pipeline).

```
    ...
  } else {
    for(int i = 1; i <= lastXSLT; i++) {
      File xsltFile = new File(args[i]);
```

```
      Source xsltSource = new StreamSource(xsltFile);
      if(i == lastXSLT) {
        documentResult = null;
        result = new StreamResult(outputFile);
      }
      else {
        documentResult = new DOMResult();
        result = documentResult;
      }
      transformer = transformerFactory.newTransformer(xsltSource);
      transformer.transform(source, result);
      if (i < lastXSLT) {
        source = new DOMSource(documentResult.getNode());
      }
    }
  }
}
```

TrAX

Zusammenfassung

- Flexibles API
- Integriert die verschiedenen Paradigmen
- Auch für portable Serialisierung nutzbar

8.4 Python-APIs

Python Documentation — Structured Markup Processing Tools[106]

XML in Python

Module und APIS

- Parser, SAX, DOM, ...
- ElementTree XML API
- ...

8.4.1 ElementTree

`xml.etree.ElementTree` — The ElementTree XML API[107]

[106] http://docs.python.org/2/library/markup.html
[107] http://docs.python.org/2/library/xml.etree.elementtree.html

XML in Python — ElementTree

Each element has a number of properties associated with it:

- a tag which is a string identifying what kind of data this element represents (the element type, in other words),
- a number of attributes, stored in a Python dictionary,
- a text string,
- an optional tail string,
- a number of child elements, stored in a Python sequence.

XML in Python — ElementTree

Beispiel 8.31 (Eingabedokument).

```
<p>
  <q a="1" b="hello">A test.</q>
  intermediate text
  <r/>
</p>
```

Beispiel 8.32 (`element`-Objekt).

```
>>> from xml.etree import ElementTree
>>> tree = ElementTree.parse("bsp1.xml")
>>> top = tree.getroot()
>>> top
<Element 'p' at 0x7f7cc2228c50>
>>> list(top)
[<Element 'q' at 0x7f7cc2228c90>, <Element 'r' at 0x7f7cc2228d90>]

>>> e = list(top)[0]
>>> e.tag
'q'
>>> e.attrib
{'a': '1', 'b': 'hello'}
>>> e.text
'A test.'
>>> e.tail
'\n  intermediate text\n  '
>>>
```

ElementTree — Anwendungsbeispiel

Beispiel 8.33 (Matrix).

$$\begin{pmatrix} 1 & 2 & \\ 4 & 5 & 6 \\ 7 & & \end{pmatrix}$$

Beispiel 8.34 (Matrix in XML).

```
<matrix>
  <row><c>1</c><c>2</c></row>
  <row><c>4</c><c>5</c><c>6</c></row>
  <row><c>7</c></row>
</matrix>
```

ElementTree — Anwendungsbeispiel

Beispiel 8.35 (`testmatrix.py`).

```
import sys
from lxml.etree import ElementTree

tree = ElementTree().parse(sys.argv[1])

for e in tree.iter():
    print e.tag,

print
```

Ausgabe

```
$ python testmatrix.py matrix.xml
matrix row c c row c c c row c
```

ElementTree — Anwendungsbeispiel

Beispiel 8.36 (`readmatrix1.py`).

```
import sys
from xml.etree import ElementTree

tree = ElementTree.parse(sys.argv[1])
matrix = []
row = None

for e in tree.iter():
    if e.tag == "row":
        if row != None:
            matrix.append(row)
        row = []
    elif e.tag == "c":
        row.append(int(e.text))
```

```
if row != None:
    matrix.append(row)
print matrix
```

```
$ python readmatrix1.py matrix.xml
[[1, 2], [4, 5, 6], [7]]
```

ElementTree — Anwendungsbeispiel

Beispiel 8.37 (`readmatrix2.py` — mit List Comprehensions).

```
import sys
from xml.etree import ElementTree

m = ElementTree.parse(sys.argv[1])

matrix = [[int(c.text) for c in row.findall("c")] \
              for row in m.findall("row")]

print matrix
```

Ausgabe

```
$ python readmatrix2.py matrix.xml
[[1, 2], [4, 5, 6], [7]]
```

9 Zusammenfassung und Ausblick

Rückblick

XML (Extensible Markup Language)

- Baumstrukturen
- Syntax
- Unicode
- Namespaces
- Infoset

XML als Metasprache / Sprachdefinitionen

- DTD
- XML Schema

Rückblick

World Wide Web (WWW)

- Ausgangspunkt: Internet + Hypertext
- URL/URI
- HTTP
- HTML

Web-Technologien

- HTML5
- CSS
- JavaScript (ES6)
- DOM
- asynchrone Programmierung
- Promises, async function

Rückblick

XML-Anfragesprachen

- XPath und XQuery
- (XSLT)
- Datenmodell
- Typen
- Ausdrücke
- Funktionsbibliothek
- XML-Datenbank: eXist

Rückblick

XML-Programmierung I: XSLT 2.0

- Templates (Regeln), Patterns
- Instruktionen
- Sortieren, Gruppieren, Nummerierung, ...

XML-Programmierung II: XQuery 1.0

- Erweiterbare Anfragesprache
- Webapplikationen mit eXist

XML-Programmierung III: Java

- SAX (ereignisorientiert)
- DOM (Bäume)
- TrAX (Transformationen)

Ausblick

Projektgruppe im Wintersemester!

Auf Wiedersehen!