

## Netzprogrammierung 8. HTML und XML

Prof. Dr.-Ing. Robert Tolksdorf  
Freie Universität Berlin  
Institut für Informatik  
Netzbasierte Informationssysteme  
mailto: [tolk@inf.fu-berlin.de](mailto:tolk@inf.fu-berlin.de)  
<http://www.robert-tolksdorf.de>



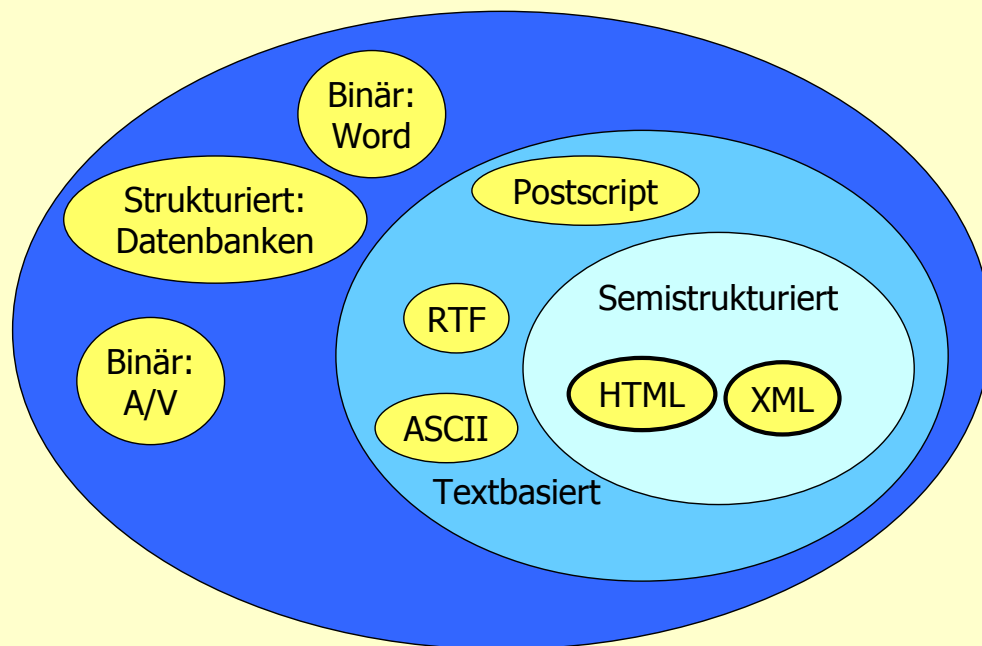
[1] © Robert Tolksdorf, Berlin

## Überblick

1. HTML
2. HTML Verarbeitung
3. Daten im Netz
4. XML Dokumente
5. XML Dokumententypen

[2] © Robert Tolksdorf, Berlin

## Daten im Netz



[3] © Robert Tolksdorf, Berlin

## HTML

[4] © Robert Tolksdorf, Berlin

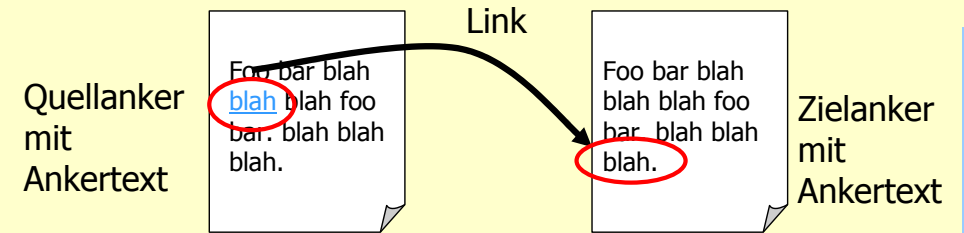
## Hypertext Markup Language

- Dominierende Sprache zur Auszeichnung von Dokumenten im Internet
- Definiert vom World Wide Web Consortium, W3C:
  - MIT (Massachusetts Institute of Technology, Laboratory for Computer Science)
  - ERCIM (European Research Consortium in Informatics and Mathematics)
  - Keio University of Japan
- Jedes Informationssystem im Netz muss:
  - HTML Informationen integrieren können
  - HTML Ausgaben erzeugen
  - Mit HTML-Mitteln mit Nutzern interagieren

[5] © Robert Tolksdorf, Berlin

## Hypertext Markup Language

- Konzepte:
  - Informationen werden als Dokumente aufgefasst
  - Dokumenteninhalte werden als Klartext dargestellt
  - Dokumententeile werden durch *Markierungen/Elemente/Tags* ausgezeichnet
    - Inhaltlich (<h1>Einleitung</h1>, <em>wichtig</em>)
    - Gestalterisch (<b>wichtig</b>)
  - Dokumente werden durch Links zu einem Hypertext verbunden (dadurch entsteht ein Netz, das Web)



[6] © Robert Tolksdorf, Berlin

## HTML

- Sprache umfaßt
  - Elemente (wie <h1>)

```
<h1>Neue Vorlesungen</h1>
<br>
<hr>
```
  - Attribute

```
<hr height="3">
```
  - Entitäten

```
&amp;
&auml;
```
  - Grammatikalische Regeln über Elemente (<html> ist Startsymbol, darin die Elemente <head> und <body>)

[7] © Robert Tolksdorf, Berlin

## HTML Beispiel/1

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
  <title>FU-Berlin: Institut für Informatik</title>
  <base href="http://www.inf.fu-berlin.de">
</head>
<body>
  <p><a href="http://www.fu-berlin.de/">Freie Universit&auml;t
  Berlin</a><br>
  <a href="http://www.math.fu-berlin.de/">Fachbereich Mathematik
  und Informatik</a></p>
  <h1>Institut für Informatik</h1>
  <p><a href="http://www.inf.fu-berlin.de/index_en.html">Homepage
  in English</a>.</p>
```

[8] © Robert Tolksdorf, Berlin

## HTML Beispiel/2

```
<form method="get" action="http://www.google.com/search">
<a href="http://www.google.de">Google</a>-Sitesearch:
  <nobr><font size=2>
<input type=text name=q size=15 maxlength=255 value="">
</font></nobr>
<input type=hidden name=sitesearch value="inf.fu-berlin.de">
  <input type=hidden name=domains value="inf.fu-berlin.de">

</form>
```

<h2>Aktuelle Meldungen</h2>

```
<p>Das Ferienprojekt <a href=
"http://www.inf.fu-berlin.de/~block/schachprojekt.html">
Schachprogrammierung</a>
wird wegen der umfassenden Bauarbeiten im Institut auf die nächsten
Semesterferien verschoben!</p>
</body>
</html>
```

[9] © Robert Tolksdorf, Berlin

## HTML – Elemente für Struktur

- Struktur
  - !DOCTYPE gibt Art des Dokuments an:  
<!DOCTYPE HTML PUBLIC  
-//W3C//DTD HTML 4.01 Transitional//EN>
  - <html>...</html> umfaßt Dokument
  - <head>...</head> enthält Informationen zur Seite
  - <body>...</body> umfaßt Inhalt der Seite

- Festes Seitenschema:

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

[10] © Robert Tolksdorf, Berlin

## HTML – Elemente im Kopfteil

- <base> enthält Basis-Adresse der Seite
- <title> enthält Titel der Seite  
<title>FU-Berlin: Institut für Informatik</title>
- <meta> enthält
  - Inhaltsklassifikation der Seite  
<meta scheme="ISBN" name="identifizier"  
content="0-8230-2355-9">
  - oder Protokollinformation  
<meta http-equiv="Expires"  
content="Tue 24 Sep 2002 00:00:00 GMT">
- <link> gibt Beziehung zu anderer Seite an  
<link rel="Glossary" href="URL">

[11] © Robert Tolksdorf, Berlin

## HTML – Elemente für Gestaltung

- Umbruch, Trennungen  
wbr br nobr p spacer  
Neue<br>Zeile
- Schriftarten  
b i s strike tt u blink bdo marquee  
Das ist <b>wirklich wichtig</b>
- Schriftauszeichnung  
abbr acronym cite code del dfn em ins  
kbd samp strong var ruby rt rb
- Formeln  
sub sup
- Schriftgröße  
basefont font big small

[12] © Robert Tolksdorf, Berlin

## HTML – Elemente für inhaltliche Strukturen

- Überschriften  
h1 h2 h3 h4 h5 h6  
`<h2>Aktuelle Meldungen</h2>`
- Blöcke  
comment hr div span address pre xmp  
plaintext listing blockquote q banner  
multicol center  
`<center>Blah blah bla <hr> blah  
blah</center>`

[13] © Robert Tolksdorf, Berlin

## HTML – Elemente für inhaltliche Strukturen

- Listen  
ol ul dir menu li dl dt dd  
  
`<font SIZE=+1>Allgemeines</font> <p>  
<ul>  
<li><a HREF="/inst/ag-sek/index.html"  
>Institutsleitung</a>  
<li><a HREF=  
"/inst/lageplan.html">Lageplan</a>  
<li><a HREF=  
"http://www.math.fu-berlin.de/cgi-bin/telefon"  
>Telefonverzeichnis</a>  
<li><a HREF="/inst/announce/index.html"  
>Votr&auml;ge</a>  
<li><a HREF=  
"http://www.math.fu-berlin.de/org/frauen/index.html"  
>Frauenbeauftragte</a  
</ul><p>`

[14] © Robert Tolksdorf, Berlin

## HTML – Elemente für inhaltliche Strukturen

- Tabellen  
table th tr td thead tbody tfoot col  
colgroup

```
<table border="1">  
<tr><th align="center">Währung</th>  
  <th align="center">1 EUR</th></tr>  
<tr><td>Deutschland (DEM)</td>  
  <td align="right">1,95583</td></tr>  
<tr><td>Frankreich (FRF)</td>  
  <td align="right">6,55957</td></tr>  
</table>
```

Währung	1 EUR
Deutschland (DEM)	1,95583
Frankreich (FRF)	6,55957

- Abbildungen  
img overlay  
caption map area

[15] © Robert Tolksdorf, Berlin

## HTML – Elemente für Interaktion

- Formulare  
form input select option textarea  
htmlarea button label fieldset legend

- `<form ACTION="/cgi-bin/telefon.cgi"  
METHOD="GET">  
<i>Die Nummer von</i>  
<input NAME="x" VALUE="" SIZE="30">  
<input TYPE="submit" VALUE="bitte">  
</form>`

Die Nummer von

[16] © Robert Tolksdorf, Berlin

## HTML – Elemente für komplexe Darstellung und Inhalte

- Browserdarstellung  
style frameset frame iframe noframes layer ilayer
- Applets, Skripte und Objekte  
applet param textflow script noscript object embed  
bodytext
- Hyperlinks  
a
  - Zielanker:  
<a name="Ziel">Hier Ihre Infos</a>
  - Quellanker:  
<a href="URI">Quellankertext</a>  
<a href="http://x.y.com/seite.html#Ziel"  
>Weitere Infos</a>

[17] © Robert Tolksdorf, Berlin

## Information aus HTML-Seiten erschliessen

- Erschliessen des Hypergraphen selber
  - Crawling
- Erschliessen der Dokumente
  - HTTP Protokoll über Internet
- Erschliessen der Inhalte von Seiten
  - Extraktion aus HTML-Text
  - Nutzung der Informationen in Tags (<address>, <title>)
- Problem: Semantik der Inhalte
  - Wie extrahieren ("Produkt" etc.)
  - Markierung nutzen (<address>, <h\*> etc.)
  - Gestaltung (Bilder, Framesets etc.)

[18] © Robert Tolksdorf, Berlin

## HTML Verarbeitung

[19] © Robert Tolksdorf, Berlin

## Verarbeitung von HTML

- Für die Verarbeitung von HTML notwendig
  - Zugriff aus Ressourcen
    - Zugriffsprotokolle
      - Sockets, TCP/UDP
      - HTTP, FTP etc.
    - Implementierungen der Protokolle
      - URLConnection in Java
  - Verarbeitung von HTML Ressourcen
    - Analyse des Inhalts
      - Parser
        - Z.B. Paket javax.swing.text.html.parser
    - Datenstruktur für Inhalt
      - DOM HTML

[20] © Robert Tolksdorf, Berlin

## javax.swing.text.html.parser

- Paket javax.swing.text.html.parser enthält einen Parser für HTML
- Basiert auf Rückrufen bei Auftreten einer bestimmten Informationseinheit in HTML:

Parserfortschritt

<html><head><title>HTML Seite</title>...

Ereignis:  
Start des html Tags gefunden  
Aufruf:  
handleStartTag(HTML.tag.HTML)

Ereignis:  
Start des head Tags gefunden  
Aufruf:  
handleStartTag(HTML.tag.HEAD)

Ereignis:  
Start Text gefunden  
Aufruf: handleText(„HTMLSeite“,0)

Ereignis:  
Ende des title Tags gefunden  
Aufruf:  
handleEndTag(HTML.tag.TITLE)

[21] © Robert Tolksdorf, Berlin

## Beispiel: Anzahl der Tags zählen

```
import java.net.*;           import java.io.*;
import javax.swing.text.*;   import javax.swing.text.html.*;
import javax.swing.text.html.parser.*;

class TagCounter extends HTMLToolkit.ParserCallback {
    int tagCount;

    public void handleStartTag(HTML.Tag t, MutableAttributeSet a, int pos) {
        tagCount++;
    }

    public void handleEndTag(HTML.Tag t, int pos) {
        if (t==HTML.Tag.HTML) {
            System.out.print(tagCount);
        }
    }

    public void handleSimpleTag(HTML.Tag t, MutableAttributeSet a, int pos) {
        tagCount++;
    }
}
```

[22] © Robert Tolksdorf, Berlin

## Beispiel: Anzahl der Tags zählen

```
public static void main(String[] argv) {
    try { // URL holen
        URL page=new URL(argv[0]);
        URLConnection connection=page.openConnection();
        connection.connect();
        if (connection.getContentType().startsWith("text/html")) {
            // bei HTML Inhalt Ströme aufstecken
            InputStream is = connection.getInputStream();
            InputStreamReader isr = new InputStreamReader(is);
            BufferedReader br = new BufferedReader(isr);
            TagCounter tagCounter= new TagCounter();
            // Parser erzeugen und aufrufen
            ParserDelegator parser = new ParserDelegator();
            parser.parse(br,tagCounter, false);
        }
    } catch (Exception e) {}
}
```

[23] © Robert Tolksdorf, Berlin

## javax.swing.text.html.HTMLToolkit.ParserCallback

- void handleStartTag(HTML.Tag t, MutableAttributeSet a, int pos)  
Start-Tag ist an Position *pos* im Strom aufgetreten
- void handleEndTag(HTML.Tag t, int pos)  
Ende-Tag ist aufgetreten
- void handleSimpleTag(HTML.Tag t, MutableAttributeSet a, int pos)  
Einfaches Tag (<br>) ist aufgetreten
- void handleComment(char[] data, int pos)  
Kommentar (<!-- .... -->) aufgetreten
- void handleText(char[] data, int pos)  
Markierter Text aufgetreten

[24] © Robert Tolksdorf, Berlin

## Beispiel: Umformatierung von HTML

```
class HTMLStripper extends HTMLToolkit.ParserCallback {
    public void handleStartTag(HTML.Tag t, MutableAttributeSet a, int pos) {
        if ((t==HTML.Tag.P) ||
            (t==HTML.Tag.H1) || (t==HTML.Tag.H2) || (t==HTML.Tag.H3) ||
            (t==HTML.Tag.H4) || (t==HTML.Tag.H5) || (t==HTML.Tag.H6)) {
            System.out.println(); // Leerzeile als neuer Absatz
            System.out.println();
        }
        if ((t==HTML.Tag.B) || (t==HTML.Tag.STRONG) || (t==HTML.Tag.EM)) {
            System.out.print("**"); // Hervorhebung
        }
    }

    public void handleEndTag(HTML.Tag t, int pos) {
        if ((t==HTML.Tag.B) || (t==HTML.Tag.STRONG) || (t==HTML.Tag.EM)) {
            System.out.print("**"); // Hervorhebung
        }
    }
}
```

[25] © Robert Tolksdorf, Berlin

## Beispiel: Umformatierung von HTML

```
public void handleSimpleTag(HTML.Tag t,
                           MutableAttributeSet a, int pos) {
    if (t==HTML.Tag.BR) {
        System.out.println(); // Neue Zeile
        System.out.println();
    }
}

public void handleText(char[] data, int pos) {
    System.out.print(data); // Eigentlichen Text unverändert
}
```

[26] © Robert Tolksdorf, Berlin

## Beispiel: Umformatierung von HTML

```
>java HTMLStripper http://www.inf.fu-berlin.de
```

FU-Berlin: Institut für Informatik

Freie Universität Berlin

[...]

Überblick

**\*\*Arbeitsgruppen\*\*** - Auf diesen dezentral verwalteten Seiten finden Sie Themen bezogene Informationen zu Forschung & Lehre.

**\*\*Kontakt\*\*** - Geschäftsführung, Anschrift, Telefon, Standort

**\*\*Service\*\*** - Stellenausschreibungen, Sekretariate, Rechnerbetrieb, Bibliothek, Zentrum für Digitale Medien, Technical Reports, Prüfungsberatung

**\*\*Gremien\*\*** - Studierende, Frauenbeauftragte, Prüfungsausschuss

**\*\*Leute\*\*** - Private Homepages

Studium

**\*\*Lehre\*\*** - Studiengänge, Kommentiertes Vorlesungsverzeichnis, Allgemeines zu den Informatik-Lehrveranstaltungen, aktuelle und frühere Lehrveranstaltungen, Praktika

[27] © Robert Tolksdorf, Berlin

## javax.swing.text.html.HTML.Tag

- static HTML.Tag A
- static HTML.Tag ADDRESS
- static HTML.Tag APPLET
- static HTML.Tag AREA
- static HTML.Tag B
- static HTML.Tag BASE
- static HTML.Tag BASEFONT
- static HTML.Tag BIG
- static HTML.Tag BLOCKQUOTE
- static HTML.Tag BODY
- static HTML.Tag BR
- static HTML.Tag CAPTION
- static HTML.Tag CENTER
- static HTML.Tag CITE
- static HTML.Tag CODE
- ...

[28] © Robert Tolksdorf, Berlin

## Beispiel: Testen von Attributen

```
class ImgAltTest extends HTMLToolkit.ParserCallback {  
  
    public void handleSimpleTag(HTML.Tag t, MutableAttributeSet a, int pos) {  
        if (t==HTML.Tag.IMG) {  
            Object attr = a.getAttribute(HTML.Attribute.ALT);  
            if (attr==null) {  
                System.out.println("** alt Attribut bei img fehlt bei "+  
                    pos+" für "+a.getAttribute(HTML.Attribute.SRC));  
                System.out.println("Vorhandene Attribute: ");  
                for (Enumeration e = a.getAttributeNames();  
                    e.hasMoreElements();) {  
                    Object at=e.nextElement();  
                    System.out.print(at+" mit Wert ");  
                    System.out.println(a.getAttribute(at));  
                }  
            }  
        }  
    }  
}
```

[29] © Robert Tolksdorf, Berlin

## Beispiel: Testen von Attributen

```
>java ImgAltTest http://www.robert-tolksdorf.de  
** alt Attribut bei img fehlt bei 14019 für  
    http://m1.nedstatbasic.net/n?id=ACJn3g5wJP8TxLTpbkIYQs7V  
    YRcA  
Vorhandene Attribute:  
nosave mit Wert #DEFAULT  
border mit Wert 0  
height mit Wert 18  
width mit Wert 18  
src mit Wert  
    http://m1.nedstatbasic.net/n?id=ACJn3g5wJP8TxLTpbkIYQs7V  
    YRcA
```

[30] © Robert Tolksdorf, Berlin

## javax.swing.text.html.HTML.Attribute

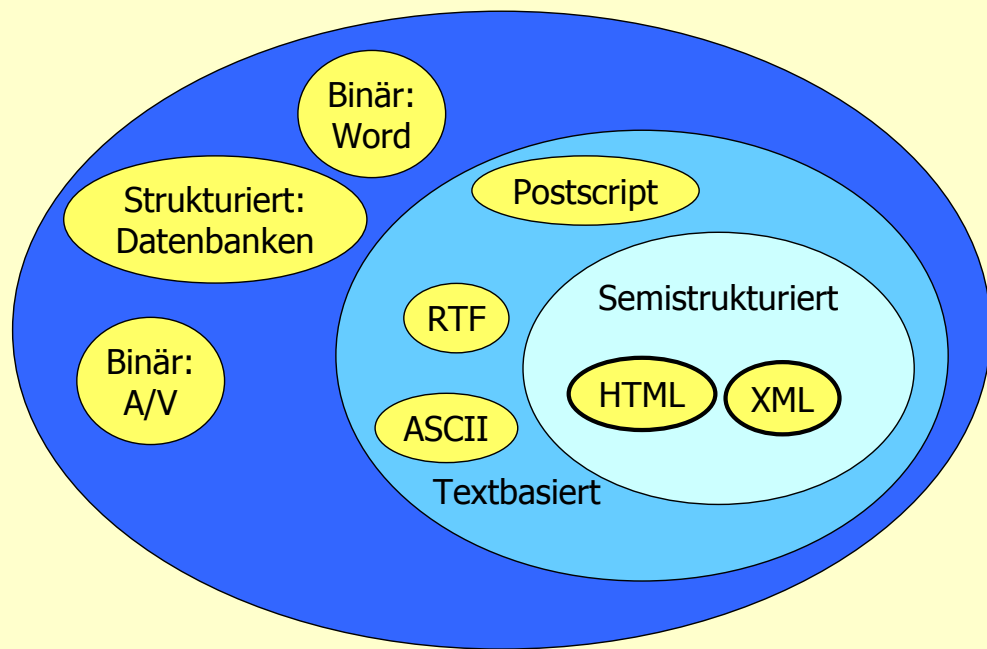
- static HTML.Attribute ACTION
- static HTML.Attribute ALIGN
- static HTML.Attribute ALINK
- static HTML.Attribute ALT
- static HTML.Attribute ARCHIVE
- static HTML.Attribute BACKGROUND
- static HTML.Attribute BGCOLOR
- static HTML.Attribute BORDER
- static HTML.Attribute CELLPADDING
- static HTML.Attribute CELLSPACING
- static HTML.Attribute CHECKED
- static HTML.Attribute CLASS
- static HTML.Attribute CLASSID
- static HTML.Attribute CLEAR
- static HTML.Attribute CODE
- static HTML.Attribute CODEBASE
- static HTML.Attribute CODETYPE
- static HTML.Attribute COLOR
- static HTML.Attribute COLS
- ...

[31] © Robert Tolksdorf, Berlin

Daten im Netz

[32] © Robert Tolksdorf, Berlin

## Daten im Netz



[33] © Robert Tolksdorf, Berlin

## Auszeichnungssprachen

- Auszeichnungssprachen fügen *Markierungen* zu einem Text hinzu
- Beispiel HTML:

```
<U>Robert Tolksdorf</U>  
<ADDRESS>  
TU Berlin<BR>  
FR 6-10<BR>  
Franklinstr. 28/29<BR>  
D-10578 Berlin<BR>  
</ADDRESS>
```

Robert Tolksdorf  
TU Berlin  
FR 6-10  
Franklinstr. 28/29  
D-10578 Berlin

- *Tags* haben *logische* oder *visuelle* Bedeutung

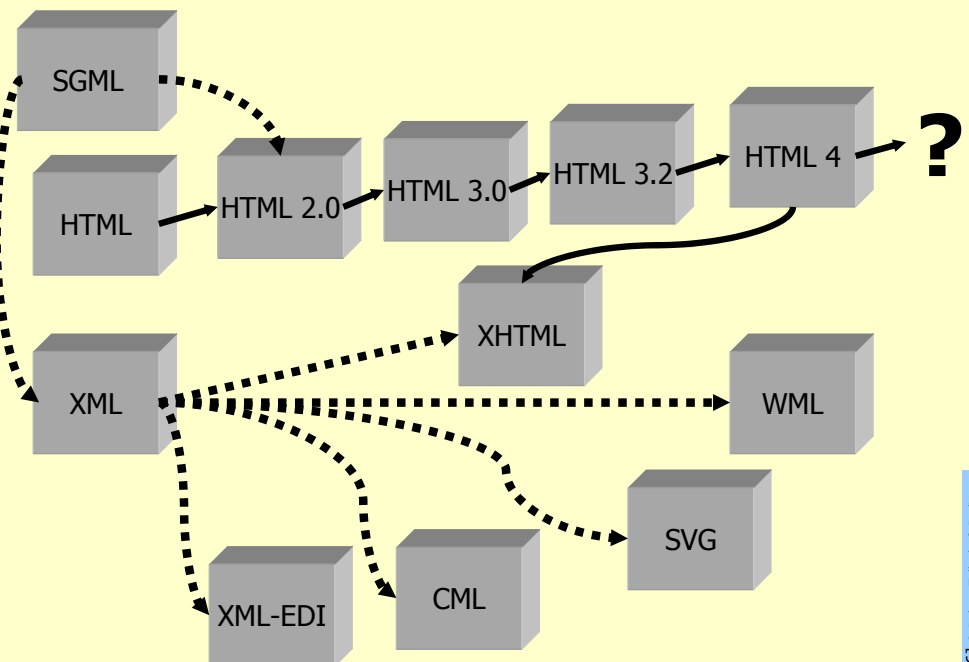
[34] © Robert Tolksdorf, Berlin

## Auszeichnungssprachen

- Kann es eine universelle Auszeichnungssprache geben?
  - Alle visuellen und sonstigen Möglichkeiten aller Ausgabegeräte müßten durch Tags steuerbar sein
  - Alle semantischen Konzepte aller Domänen müßten durch Tags repräsentierbar sein
  - Alle notwendigen Granularitäten der Auszeichnung müßten unterstützt werden:
    - `<ADRESSE>...</ADRESSE>`
    - `<ADRESSE><STRASSE>...</STRASSE><ORT>...</ORT></ADRESSE>`
    - `<ADRESSE><STRASSE>...</STRASSE><ORT><PLZ>...</PLZ><ORTSNAME>...</ORTSNAME></ORT></ADRESSE>`
- Nein: Anwendungsspezifische Auszeichnung nötig

[35] © Robert Tolksdorf, Berlin

## XML als Ergebnis der HTML Entwicklung

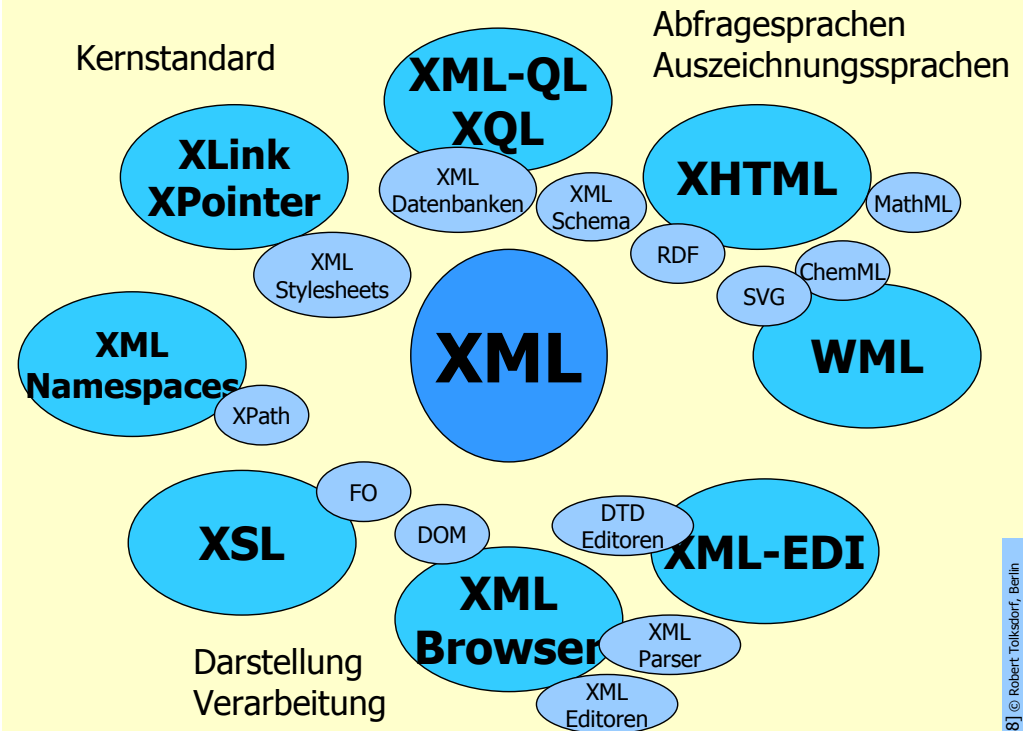


[36] © Robert Tolksdorf, Berlin

## XML Q&A

- *Was ist XML?*  
Die Extensible Markup Language ist die Definition einer Untermenge von SGML, mit der man einfach Auszeichnungssprachen definieren kann
- *Woher kommt XML?*  
XML ist ein Standard des World Wide Web Konsortiums W3C
- *Was macht man mit XML?*  
Anwendungsspezifische Auszeichnungssprachen definieren und standardisieren
- *Was ist der Vorteil von XML-basierten Auszeichnungssprachen?*  
Standardisierung ermöglicht Datenaustausch

[37] © Robert Tolksdorf, Berlin



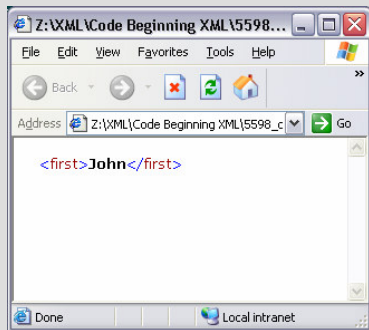
## XML Dokumente

[Folien: Klaus Schild, NBI]

[39] © Robert Tolksdorf, Berlin

## Lernziele

- Wie ist ein XML-Dokument aufgebaut?
- Was sind Elemente und was sind Attribute?
- Wann sollten Attribute und wann Elemente verwendet werden?



- `<first>` ist ein *Anfangs-Tag* (engl. *start tag*).
- `</first>` ist das dazugehörige *End-Tag*.
- `<first>John</first>` wird *Element* und „first“ *Elementname* genannt.
- „John“ wird auch *Inhalt* des Elementes genannt.

1. Elemente können aus einfachem Text bestehen:

```
<first>John</first>
```

Der Inhalt wird dann auch als *parsed character data* (PCDATA) bezeichnet.

2. Elemente können auch weitere Elemente enthalten:

```
<name>  
<first>John</first>  
<last>Doe</last>  
</name>
```

Wird auch als *strukturierter Inhalt* bezeichnet

3. Elemente können *gleichzeitig* Text (PCDATA) und Elemente enthalten:

```
<section>  
Text  
<subsection> ... </subsection>  
Text  
</section>
```

Wird auch als *gemischter Inhalt* (engl. *mixed content*) bezeichnet

4. Elemente können auch leer sein:

```
<name>  
<first>John</first>  
<middle></middle>  
<last>Doe</last>  
</name>
```

Beachte Unterschied zu nicht vorhandenen Element!

Leeres Element `<tag-name></tag-name>` kann mit `<tag-name/>` *abgekürzt* werden:

```
<name>  
<first>John</first>  
<middle/>  
<last>Doe</last>  
</name>
```

## PCDATA



- Für XML reservierte Symbole wie < und & dürfen *nicht* in PCDATA verwendet werden.
- Statt dessen &lt bzw. &amp verwenden.
- &lt und &amp werden auch *entity references* genannt.
- In XML gibt es insgesamt fünf entity references:
  - &amp für &
  - &lt für <
  - &gt für >
  - &apos für '.
  - &quot für ".

## CDATA



- Text mit vielen reservierten Symbolen besser als sog. *character data* (CDATA) darstellen.
- XML-Parser betrachtet dann den entsprechenden Text als einfachen String und parst den Inhalt nicht.
- XML-Parser sucht ]]> und analysiert die Zeichenkette bis dahin nicht.

```
- <comparison>
  <![CDATA[ 6 is < 7 & 7 > 6 ]]>
</comparison>
```

## Regeln für wohlgeformte Dokumente



1. Jedes Anfangs-Tag muss ein zugehöriges End-Tag haben.
2. Elemente dürfen sich nicht überlappen.
3. XML-Dokumente haben genau ein Wurzel-Element.
4. Elementnamen müssen den Namenskonventionen von XML entsprechen.
5. XML beachtet grundsätzlich Groß- und Kleinschreibung.
6. XML belässt Formatierungen (*white space*) im Text.

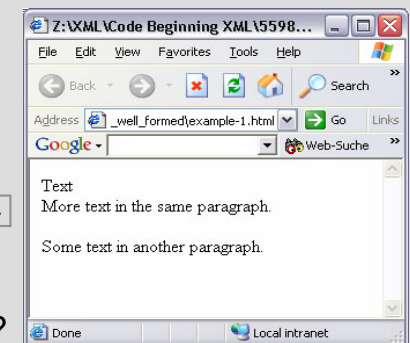
## Regel 1: Anfangs- und End-Tags



- In XML muss jedes Anfangs-Tag ein zugehöriges End-Tag haben.
- In HTML gilt diese Regel nicht:

```
<HTML>
<BODY>
  <P>Text
    <BR>More text in the same paragraph.
  <P>Some text in another paragraph.</p>
</BODY>
</HTML>
```

- Wo endet das erste P-Element?
- Syntaktisch mehrdeutig
- ➔ XML-Dokumente strukturierter

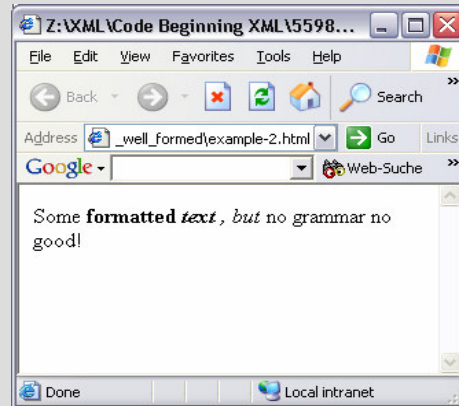


## Regel 2: Überlappung von Elementen



- In XML dürfen sich Elemente *nicht* überlappen.
- In HTML dürfen sich Tags überlappen:

```
<HTML>
<BODY>
<P>Some
<STRONG>formatted
<EM>text
</STRONG>, but
</EM>
no grammar no good!
</p>
</BODY>
</HTML>
```



- semantisch mehrdeutig!

## Regel 3: Wurzel-Elemente



- Jedes XML-Dokumente hat *genau ein* Wurzel-Element.
- Wurzel-Element entspricht dem Dokument-Typ.
- Also z.B. statt zweier Wurzel-Elemente

```
<name>John</name>
```

```
<name>Jane</name>
```

ein zusätzliches Element einführen:

```
<names>
```

```
<name>John</name>
```

```
<name>Jane</name>
```

```
</names>
```

## Regel 4: Namenskonventionen



- Namen beginnen entweder mit einem Buchstaben oder „\_“: z.B. first, First oder \_First
- Nach dem ersten Zeichen sind zusätzlich Zahlen sowie „-“ und „.“ erlaubt: z.B. \_1st-name oder \_1st.name
- Namen enthalten keine Leerzeichen.
- Namen enthalten kein „:“.
- Namen beginnen nicht mit „xml“, unabhängig davon, ob die einzelnen Buchstaben groß- oder kleingeschrieben sind.
- Diese Konventionen gelten für alle Namen, nicht nur für Elementnamen.

## Regel 4: Namenskonventionen



- `<résumé>` ✓
- `<xml-tag>` nicht wohlgeformt: beginnt mit „xml“
- `<123>` nicht wohlgeformt: beginnt mit Zahl
- `<fun=xml>` nicht wohlgeformt: enthält „=“
- `<first name>` nicht wohlgeformt: enthält Leerzeichen

## Regel 5: Groß- und Kleinschreibung

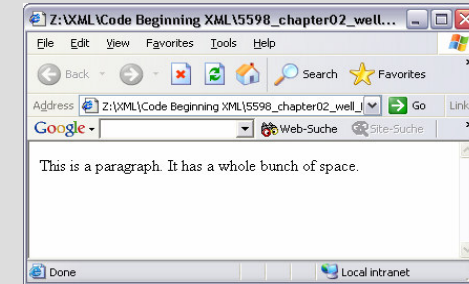


- XML beachtet grundsätzlich Groß- und Kleinschreibung.
- Im Gegensatz zu HTML unterscheidet XML also z.B. zwischen `<P>` und `<p>`.
- Dennoch möglichst nicht gleichzeitig `<First>` und `<first>` verwenden!

## Regel 6: White Space



- Beispiel:  
`<P>`This is a paragraph. It has a whole bunch of space.`</P>`
- HTML reduziert alle Formatierungen (*white spaces*) im Text auf ein einziges Leerzeichen :



## Regel 6: White Space



- XML belässt alle Formatierungen im Text.
- Beispiel: Der Inhalt von  
`<paragraph>`This is a paragraph. It has a whole bunch of space.`</paragraph>`  
ist also:  
This is a paragraph. It has a whole bunch of space.
- Beachte: In Browsern wie dem IE 5 und IE 6 sind diese Formatierungen allerdings *nicht sichtbar*.
- Grund: XML-Dokumente werden zur Darstellung im Browser in HTML umgewandelt.

## Attribute

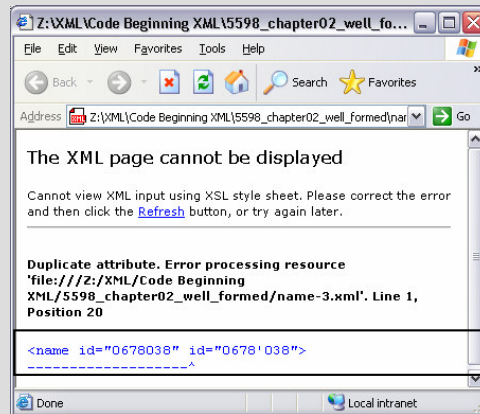


- Elemente können eine beliebige Anzahl von Attributen haben.
- *Attribute* sind Name-Wert-Paare der Form `name="wert"` oder `name='wert'`.
- Der Wert eines Attributes ist immer ein String.
- Die Reihenfolge der Attribute ist belanglos.

## Attribute



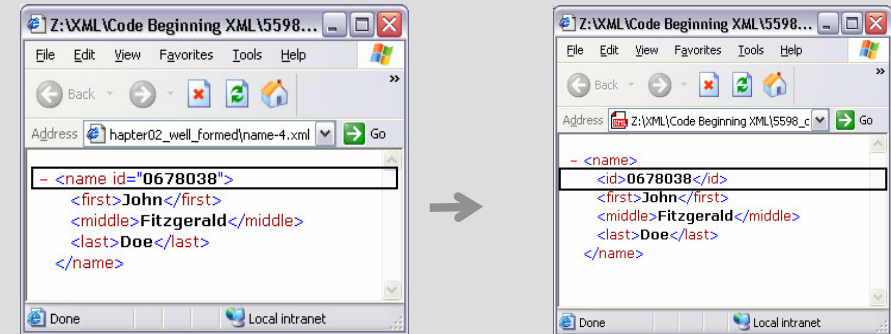
- Ein Element darf niemals zwei Attribute mit dem selben Attribut-Namen haben:



## Element statt Attribut



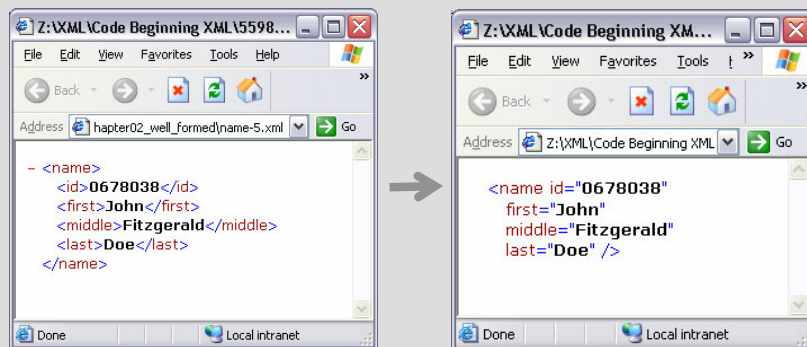
- Jedes Attribut kann auch als Kind-Element repräsentiert werden:



## Attribut statt Element



- Jedes *unstrukturierte* Kind-Element kann auch als Attribut dargestellt werden:



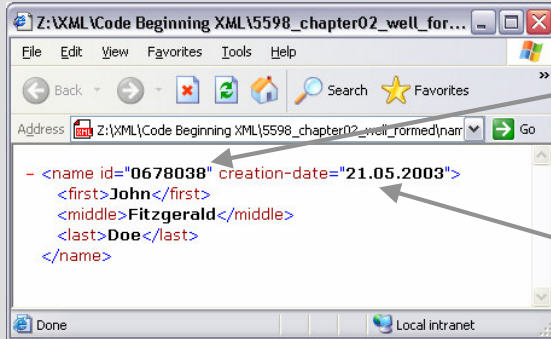
## Attribut oder Element?



Wann also Attribute und wann Elemente verwenden?

- Ein Attribut kann nur einen String als Wert haben, ein Element kann beliebig strukturiert werden.
- Die Reihenfolge der Attribute ist belanglos, diejenige von Elementen nicht.
- Einheitliche Darstellung mit Elementen ist eleganter, Darstellung mit Attributen kompakter.
- Fazit:** Attribute eignen sich besonders für einfache, unstrukturierte Metadaten.

## Attribut oder Element?



interner Schlüssel  
des Datensatzes

Erstellungsdatum  
des Datensatzes

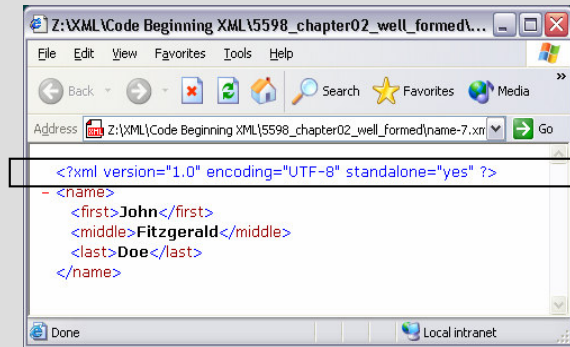
- Beide Attribute stellen Meta-Information dar.
- Reihenfolge ist egal.
- Deshalb Repräsentation als Attribut
- Problem: Datum "21.05.2003" ist unstrukturierter String.

© Klaus Schild, XML Clearinghouse 2003

## Die XML-Deklaration



- Enthält Informationen für Parser, insbesondere die verwendete XML-Version und Kodierung
- Muss immer am Anfang der Datei erscheinen
- Ist zwar optional, sollte aber dennoch immer vorhanden sein!

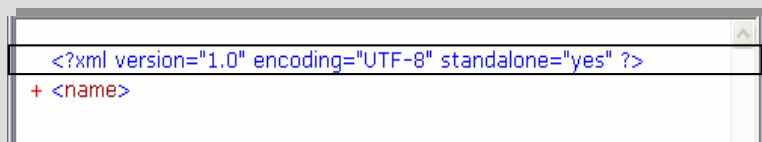


© Klaus Schild, XML Clearinghouse 2003

## Die XML-Deklaration



- version (obligatorisch): Verwendete XML-Version, aktuelle Version ist "1.0".
- standalone (optional): Gibt an, ob es eine zugehörige Dokument-Typ-Definition gibt ("no") oder nicht ("yes").
- encoding (optional): Verwendete Kodierung des XML-Dokumentes
- Die drei Attribute müssen *immer* in dieser Reihenfolge erscheinen.



© Klaus Schild, XML Clearinghouse 2003

## XML-Deklaration: Kodierung



- ASCII (*American Standard Code for Information*) stellt einzelne Zeichen entweder mit 7 oder 8 Bit dar.
- 7-Bit-ASCII deckt mit 128 Zeichen nur einen kleinen Teil der weltweit verwendeten Zeichen ab.
- Es gibt verschiedene 8-Bit-ASCII-Versionen (u.a. ISO-8859-1 und windows-1252).
- Nur die ersten 128 Zeichen der 8-Bit-ASCII-Versionen sind bei allen 8-Bit-ASCII-Versionen identisch.
- Fazit: ASCII ist nur beschränkt zum weltweiten Austausch von Textdateien geeignet.

© Klaus Schild, XML Clearinghouse 2003

## XML-Deklaration: Kodierung



- Unicode deckt mit 65356 Zeichen alle weltweit verwendeten Zeichen ab.
- Unicode gibt es auf Basis von 8 Bit (UTF-8) und 16 Bit (UTF-16).
- Sowohl UTF-8 als auch UTF-16 decken den kompletten Unicode-Zeichensatz ab, UTF-8 benutzt lediglich eine intelligentere Kodierung.
- 7-Bit-ASCII ist eine Teilmenge von UTF-8.
- Textdateien, die größtenteils aus dem 7-Bit-ASCII-Zeichensatz bestehen, können in UTF-8 kompakter sein, alle anderen sind meist in UTF-16 kompakter.

## XML-Deklaration: Kodierung

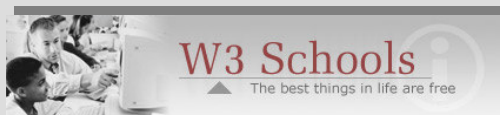


- Intern müssen alle XML-Parser mit Unicode arbeiten.
- Das Attribut `encoding` gibt an, welches Kodierungsschema das *betreffende XML-Dokument* verwendet.
- Fehlt dieses Attribut in der XML-Deklaration, dann wird angenommen, dass das Dokument entweder in UTF-8 oder UTF-16 kodiert ist.
- Tipp: Das XML-Dokument in Unicode abspeichern. Das Attribut `encoding` kann dann in der XML-Deklaration weggelassen werden.

## Online-Test XML



→ [http://www.w3schools.com/xml/xml\\_quiz.asp](http://www.w3schools.com/xml/xml_quiz.asp)



## Zusammenfassung



- *Elemente* bestehen aus einem *Anfangs-Tag*, einem dazugehörigen *End-Tag* und dem dazwischen eingeschlossenem *Inhalt*.
- Ein Element kann nur Text, nur Kind-Elemente oder gleichzeitig Text und Kind-Elemente enthalten.
- Ein Element kann auch leer sein.

## Zusammenfassung



- Wohlgeformte XML-Dokumente müssen folgenden syntaktischen Regeln entsprechen:
- Jedes Anfangs-Tag muss ein zugehöriges End-Tag haben.
- Elemente dürfen sich nicht überlappen.
- XML-Dokumente haben genau ein Wurzel-Element.
- Elementnamen müssen den Namenskonventionen von XML entsprechen, müssen u.a. entweder mit einem Buchstaben oder mit „\_“ beginnen.
- XML beachtet grundsätzlich Groß- und Kleinschreibung.

© Klaus Schild, XML Clearinghouse 2003

## Zusammenfassung



- Jedes Element kann eine beliebige Anzahl von Attributen haben.
- *Attribute* sind Name-Wert-Paare.
- Der Wert eines Attributes ist immer ein einfacher String.
- Die Reihenfolge der Attribute ist belanglos.
- Jedes Attribut kann auch als Kind-Element repräsentiert werden.
- Umgekehrt kann jedes unstrukturierte Kind-Element auch als Attribut dargestellt werden.
- Attribute eignen sich zur Darstellung von einfachen, unstrukturierten Metadaten.

© Klaus Schild, XML Clearinghouse 2003

## Definition von Dokumententypen

[Folien: Klaus Schild, NBI]

## Lernziele

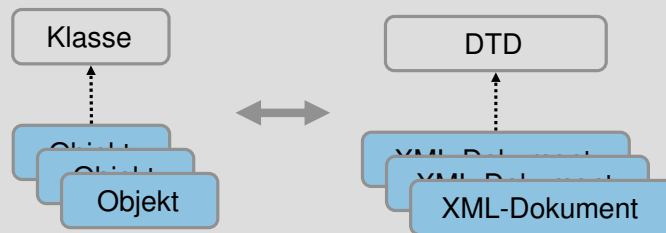


- Wie kann mit einer DTD der Aufbau von XML-Dokumenten eines bestimmten Typs beschrieben werden?
- Wie wird die Elementstruktur eines Dokumentes deklariert?
- Wie werden Attribute von Elementen deklariert?
- Was sind die Nachteile von DTDs?

© Klaus Schild, XML Clearinghouse 2003

## Dokument-Typen

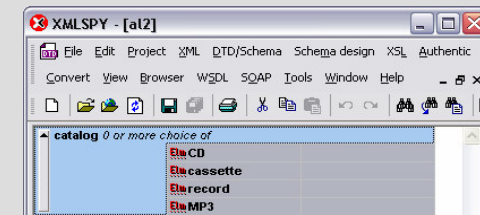
- Beschreiben den *prinzipiellen Aufbau* von Dokumenten eines bestimmten Typs.
- Können mit DTDs (*Document Typ Definitions*) spezifiziert werden.
- DTDs wurden von SGML übernommen.
- DTDs benutzen Syntax, die regulären Ausdrücken ähnlich ist.



## Deklaration der Elementstruktur

```
<!ELEMENT catalog (CD | cassette | record | MP3)*>
```

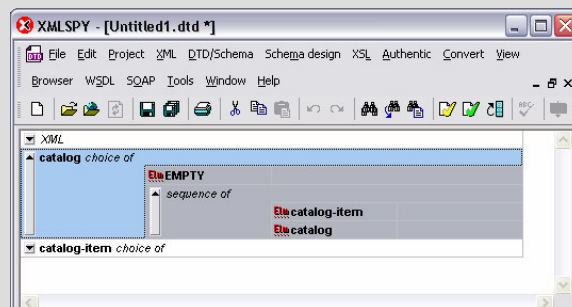
- Deklariert Element catalog
- \* bedeutet n Wiederholung mit  $n \geq 0$ .
- | bedeutet Auswahl.
- CD, cassette, record und MP3 sind jeweils Kind-Elemente.



## Deklaration der Elementstruktur

```
<!ELEMENT catalog ( EMPTY | (catalog-item, catalog) )>
<!ELEMENT catalog-item (CD | cassette | record | MP3)>
```

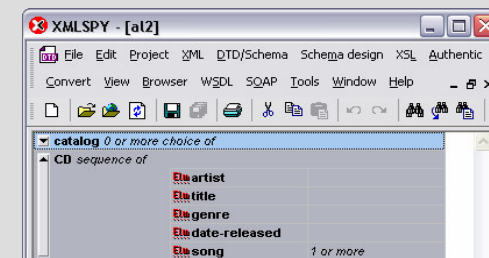
- Rekursive Element-Deklarationen sind erlaubt.
- EMPTY bedeutet leerer Element-Inhalt.



## Deklaration der Elementstruktur

```
<!ELEMENT CD (artist, title, genre, date-released, song+)>
```

- Deklariert Element „CD“
- + bedeutet n Wiederholung mit  $n \geq 1$ .
- , bedeutet Sequenz: Aufeinanderfolge von Elementen

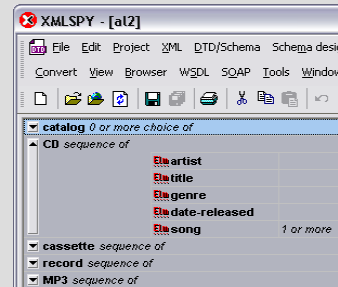


## Deklaration der Elementstruktur



```
<!ELEMENT catalog (CD | cassette | record | MP3)*>
<!ELEMENT CD (artist, title, genre, date-released, song+)>
<!ELEMENT cassette (artist, title, genre, date-released, song+)>
<!ELEMENT record (artist, title, genre, date-released, song+)>
<!ELEMENT MP3 (artist, title, genre, date-released, song+)>
```

- Deklariert Elemente cassette, record und MP3
- Analog zur Deklaration von CD

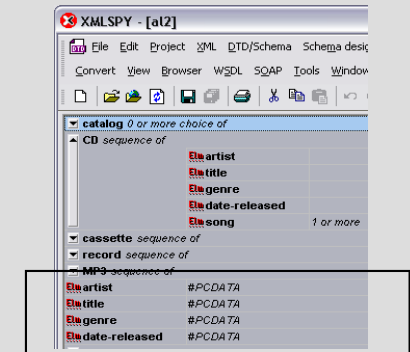


## Deklaration der Elementstruktur



```
<!ELEMENT artist (#PCDATA)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT genre (#PCDATA)>
<!ELEMENT date-released (#PCDATA)>
```

- Deklariert jeweils Elemente artist, title, genre und date-released.
- #PCDATA bedeutet unstrukturierter Text ohne reservierte Symbole wie „<“ oder „>“.

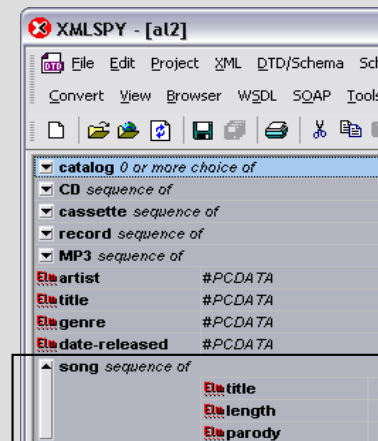


## Deklaration der Elementstruktur



```
<!ELEMENT song (title, length, parody)>
```

- title, length und parody sind Kind-Elemente von song.
- title, length und parody erscheinen immer in dieser Reihenfolge.

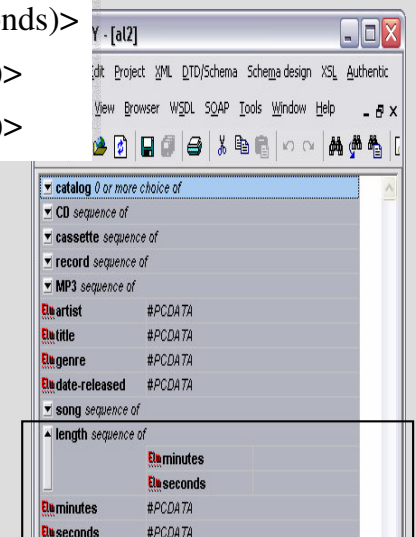


## Deklaration der Elementstruktur



```
<!ELEMENT length (minutes, seconds)>
<!ELEMENT minutes (#PCDATA)>
<!ELEMENT seconds (#PCDATA)>
```

- minutes und seconds werden als unstrukturierter Text deklariert.
- Datentypen wie positiveInteger oder Integer stehen in DTDs *nicht* zur Verfügung.



## Deklaration der Elementstruktur



```
<!ELEMENT catalog (CD | cassette | record | MP3)*>
<!ELEMENT CD (artist, title, genre, date-released, song+)>
<!ELEMENT cassette (artist, title, genre, date-released, song+)>
<!ELEMENT record (artist, title, genre, date-released, song+)>
<!ELEMENT MP3 (artist, title, genre, date-released, song+)>
```

- Struktur von CD, cassette, record und MP3 sind identisch.
- Änderungen dieser Struktur müssen an mehreren Stellen nachvollzogen werden.

## Vermeidung von Wiederholungen



```
<!ELEMENT catalog (CD | cassette | record | MP3)*>
<!ENTITY % albumContentModel "(artist, title, genre, date-released, song+)">
<!ELEMENT CD %albumContentModel;>
<!ELEMENT cassette %albumContentModel;>
<!ELEMENT record %albumContentModel;>
<!ELEMENT MP3 %albumContentModel;>
```

- `%albumContentModel;` ist ein Parameter.
- Allerdings nur einfache textuelle Ersetzung, keine Vererbungshierarchien.

## Primitive Datentypen



- Neben den komplexen Datentypen zur Strukturierung von Element-Inhalten, stehen drei primitive Datentypen (engl. *built-in datatypes*) zu Verfügung:
- `#PCDATA`: unstrukturierter Text ohne reservierte Symbole.
- `EMPTY`: Der Element-Inhalt ist leer. Das Element kann allerdings Attribute haben.  
`<!ELEMENT br EMPTY>` → `<br/>`
- `ANY`: beliebige XML-Strukturen  
`<!ELEMENT title ANY>`

## Primitive Datentypen

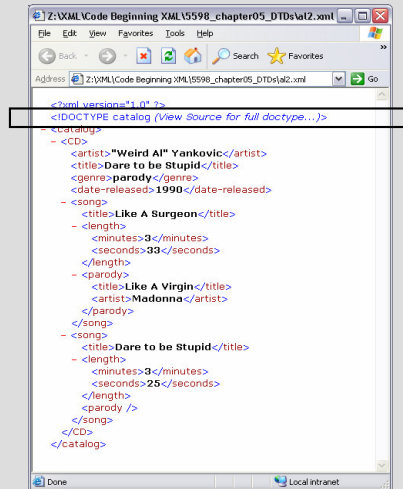


- Beachte: Elementare Datentypen, wie sie von Programmiersprachen bekannt sind, stehen in DTDs *nicht* zur Verfügung.
- `<!ELEMENT minutes (INTER)>`

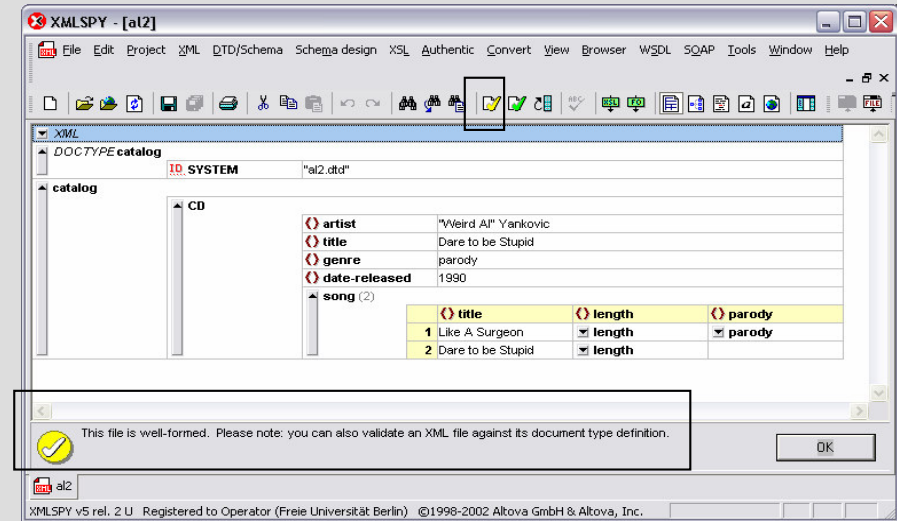
# Zulässige Dokumente



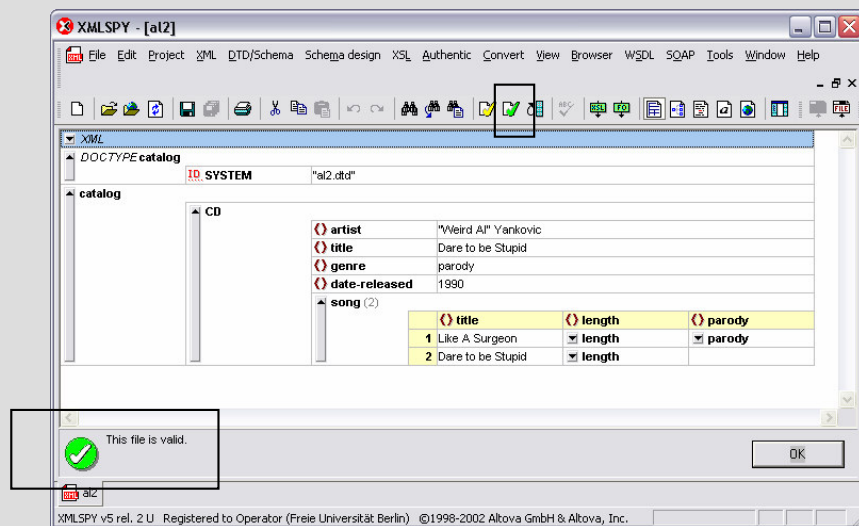
- In einem XML-Dokument kann ein Dokument-Typ spezifiziert werden.
- Das Wurzelement des Dokumentes muss genau der Struktur dieses Elementes entsprechen, wie sie im Dokument-Typ festgelegt ist.
- In diesem Fall bezeichnet man das Dokument als zulässig (engl. *valid*).



# Prüfung der Zulässigkeit



# Prüfung der Zulässigkeit



# Deklaration von Attributen



```
<!ATTLIST catalog  
  version CDATA #IMPLIED "1.0">
```

- Das Element catalog hat ein Attribut mit dem Namen version.
- Außer version hat catalog keine weiteren Attribute.
- Das Attribut ist vom Typ String (CDATA).
- **#IMPLIED**: Das Attribut ist optional.
- "1.0" ist der Standard-Wert.
- **#REQUIRED**: Das Attribut ist obligatorisch.
- **#FIXED**: Das Attribut hat immer den gleichen Wert.

## Deklaration von Attributen



```
<!ATTLIST artist
  gender (male | female) "female">
```

- Das Element `artist` hat ein Attribut mit dem Namen `gender` und keinen weiteren Attribute.
- Das Attribut hat entweder den Wert `male` oder `female` (Aufzählungstyp).
- `"female"` ist der Standard-Wert von `gender`.

## Datentypen für Attribute



- Neben Strings (CDATA) und Aufzählungstypen stehen im wesentlichen folgende Datentypen zur Verfügung:
- NMTOKEN: ein String, der den Namenskonventionen von XML entspricht
- NMTOKENS: eine Liste von solchen Namen, jeweils getrennt durch ein Leerzeichen
- ID: Bezeichner, der den Namenskonventionen von XML entspricht und innerhalb des Dokumentes eindeutig ist.
- IDREF: eine Referenz auf einen eindeutigen Bezeichner
- IDREFS: eine Liste von solchen Referenzen

## Ein Nachteil von DTDs



- Reihenfolge von Kind-Elementen ist festgelegt:  
`<!ELEMENT song (title, length)>`
- Dadurch sehr starre Struktur in XML-Dokumenten!
- Um Reihenfolgeunabhängigkeit zu garantieren, müssen alle Permutationen explizit aufgezählt werden:  
`<!ELEMENT song ((title, length) | (title, length))>`
- Für  $n$  Element gibt es  $n!$  verschiedene Permutationen.

## Weitere Nachteile von DTDs



- keine XML-Syntax, daher eigene Parser nötig
- kaum primitive Datentypen, insbesondere für Element-Inhalte
- keine eigenen Datentypen definierbar
- keine Namensräume: Bereits existierende DTDs können nur dann kombiniert werden, wenn es *keine* Namenskonflikte gibt!
- keine Vererbungshierarchien, nicht objekt-orientiert

## Zusammenfassung



- Dokument-Typen beschreiben den *prinzipiellen Aufbau* von Dokumenten eines bestimmten Typs.
- Dokument-Typen können mit Klassen und XML-Dokumente mit Objekten verglichen werden.
- In einem XML-Dokument kann ein bestimmter Dokument-Typ spezifiziert werden.
- Das Wurzelement des Dokumentes muss dann genau der Struktur dieses Elementes entsprechen, wie sie im Dokument-Typ festgelegt ist.
- In diesem Fall bezeichnet man das Dokument als zulässig (engl. *valid*).

© Klaus Schild, XML Clearinghouse 2003

## Zusammenfassung



- Dokument-Typen können mit DTDs (*Document Typ Definitions*) spezifiziert werden.
- DTDs wurden von SGML übernommen.
- DTDs haben entscheidende Nachteile:
  - keine XML-Syntax, daher eigene Parser nötig
  - kaum elementare Datentypen, insbesondere für Element-Inhalte
  - keine eigenen Datentypen definierbar
  - keine Namensräume: DTDs können nur kombiniert werden, wenn es keine Namenskonflikte gibt.
  - keine Vererbungshierarchien, nicht objekt-orientiert

© Klaus Schild, XML Clearinghouse 2003

## Zusammenfassung

## Überblick

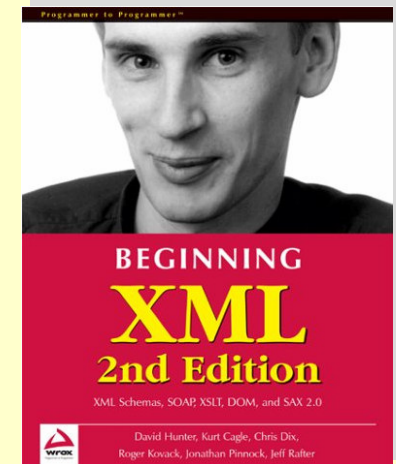
1. HTML
  1. Einfache Auszeichnungssprache
  2. Elemente (Tags), Attribute, Entitäten
  3. Elemente für
    1. Struktur
    2. Gestaltung
    3. Inhaltsstrukturen
    4. Interaktion
    5. Komplexe Inhalte
2. HTML Verarbeitung
  1. Parser notwendig
  2. Swing Paket enthält Callback-Parser
  3. Element- und Attributdefinitionen
3. Daten im Netz
4. XML Dokumente
5. XML Dokumententypen

## Literatur

- Dave Raggett (ed). HTML 4.01 Specification. W3C Recommendation. 24 December 1999  
<http://www.w3.org/TR/html4>
- Robert Tolksdorf. XHTML und HTML - die Sprachen des Web. dpunkt.verlag, Heidelberg, 5. Auflage, 2002. ISBN 3-89864-155-4.
- Stefan Münz. SELFHTML. <http://www.selfhtml.org/>

## Literatur

- Hunter, David; Cagle, Kurt; Dix, Chris: Beginning XML XML Schemas, SOAP, XSLT, DOM, and SAX 2.0  
2nd ed. 2001. Wrox Press  
1-86100-559-8



- Empfehlenswertes Skript einer anderen XML-Vorlesung: <http://www.jeckle.de/vorlesung/xml/>
- mehrere Interaktive XML-Kurse <http://www.zvon.org>