

XML-Einführung

**Jochen Topf
jochen@remote.org**

<http://www.remote.org/jochen/>

Lesen und Schreiben

Eine Einführung

Lesen und Schreiben lernen ist einfach

- man nehme: ein paar Buchstaben und andere Zeichen

Lateinische Schrift

a b c d e f g h
i j k l m n o p
q r s t u v w x
y z ä ö ü ß tt
A B C D E F G H
I J K L M N O P
Q R S T U V W X
Y Z Ä Ö Ü - 1 2
3 4 5 6 7 8 9 0

Lesen und Schreiben lernen ist einfach

- man nehme: ein paar Buchstaben und andere Zeichen
- füge sie zu Wörtern zusammen



Lesen und Schreiben lernen ist einfach

- man nehme: ein paar Buchstaben und andere Zeichen
- füge sie zu Wörtern zusammen
- forme daraus Sätze
- Der Hund hat Flöhe
- Das Wetter ist schön
- Marie geht einkaufen

Lesen und Schreiben lernen ist einfach

- man nehme: ein paar Buchstaben und andere Zeichen
- füge sie zu Wörtern zusammen
- forme daraus Sätze
- und daraus z.B. ein Buch



Lesen und Schreiben lernen ist kompliziert

- Extra Regeln für Gedichte

DAS AESTHETISCHE WIESEL

Ein Wiesel
saß auf einem Kiesel
inmitten Bachgeriesel.

Wißt ihr,
weshalb?

Das Mondkalb
verriet es mir
im stillen:

Das raffinier-
te Tier
tats um des Reimes willen.

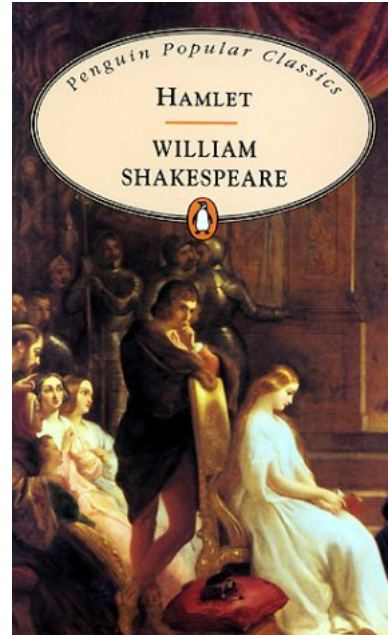
Lesen und Schreiben lernen ist kompliziert

- Extra Regeln für Gedichte
- ... und Telefonbücher

Kunzmann A.	49 59 73	Kupferschmid Barbara	38 99 17
- Achim (Wo) Nordost-24	49 47 65	- V. Leopold-7	Fax 9 21 22 71
Kunzmann Alfred Repräsentations-		- Volker und Mahler L	9 21 22 70
Artikel GmbH Geschenkartikel		- Werner Hohenzollern-8	3 37 11
Motive	0 17 59 74 10 86	Kupietz M. u. Mundorf Thomas	57 42 54
Kunzmann Amd		- Peter Wickenweg 4	8 93 17 77
(Grü) Nidda-18	0 17 95 95 66 42	- T. Krokusweg 45	88 27 10
- Arnd (Grü) Nidda-18	9 09 09 25	Kupin Nikkai Ostmark-35	49 69 41
- Arno Dr. Göllnitzer-6	0 17 27 24 26 72	Kupke Heidi	9 43 18 00
- Arno Dr. und Marita Göllnitzer-6	40 66 38	- Wolfgang	
- Brigitte Plinz-66	4 35 62	Reinhold-Schneider-131 - 0 17 27 12 56 81	
- Christiane u. Nagel Egon		- Wolfgang	
Redtenbacher-18	82 60 74	Reinhold-Schneider-131	88 49 28
- Conny Laney-22	9 54 68 57	Kuppe Lieselotte Basler-Tor-55	4 44 29
- D.	48 22 54	Kuppek Thomas Kanonier-17	59 16 31
- E.	70 75 24	Kuppel Edmund Stresemann-40	75 66 95
- E. Wilhelm-80	3 32 06	Kupper Elmar Handt-64	9 55 49 71
- Elisabeth Ersinger-18	40 36 05	Kupper & Partner Computerservice Gbr	
- Erika Badener-33	49 37 88	Am Lerchenberg 20	4 90 81 53
- Frieder (Grü) Marbacher-24	45 01 88	Kupper & Partner Computerservice Gbr	
Kunzmann-Gerdes Mechthild		Am Lerchenberg 20	4 90 81 54
(Grü) Im Speitel 2	4 85 11	Kupper & Partner Computerservice Gbr	
Kunzmann Gertrud Kaiser-201	2 61 63	Am Lerchenberg 20	9 41 82 98
Kunzmann GmbH Party-Service		Kuppig M. (Nrt)	7 81 91 14
Ostmark-33	2 01 32-0	Kuppinger Alfred u. Else	
Kunzmann H. Ostmark-33	4 84 66 06	(Ho) Neuer Weg 20	47 22 97
- Hans-Peter Kriegs-18	38 81 97	- Bruno Maurer/Mstr. Erzberger-107	75 43 95
- Heinz Ind.Mstr. Hofacker-7	68 54 91	- Christa Reinhold-Frank-6	2 76 33
- I.	70 64 87	- Ewald (Ho) Zweibrückener-5	47 34 83
- I. Fecht-2	49 33 74	- Frank Mobiltelefon	0 17 27 27 62 06
- Jana Ostmark-33	4 84 66 08	- Frank Tiertaxi	7 03 30
- Joachim Körner-52	8 30 78 27	Günther-Klotz-Anlage	81 49 98
- Jürgen Wichem-6	9 52 97 36	Funkshop Eisa-Brändström-8	47 36 67
- Jürgen (Grü) Im Speitel 104	46 57 01	Kuppinger Frank Stefan	
- Kai Ostmark-33	4 84 66 07	Am Friedhof 2	0 17 28 75 63 19
- Karl Belchen-12	88 59 18	Kuppinger Gertrud	
- Karl Uhmacher Lindenallee 62	50 29 57	(Grü) Augustenburg-58	48 26 85
- Karl Maxauer-6	56 49 51	- Günter	48 33 96
- Karl-Martin Bach-33	55 22 43	- Hans Klüßfeld 28	40 12 05
- Katja Spital-3	9 41 84 43	- Hans Rintheimer Haupt-119	61 72 92
- Klaus Gebhard-7	81 45 01	- Helmut Seubert-5	61 52 14
- Kristin	6 23 98 40	- Joachim Badener-45	49 26 70
- Kurt (Pab) Tal-58	4 56 65	- Jürgen (Wo) Nordost-3	49 66 22
- Liselotte (Grü) Eugen-Kleiber-3	48 27 42	- K. u. G. (Grü) Bruchwald-23	48 10 62
- Marco Main-19	9 89 66 11	- Martin (Ho) Taglöhnergärten 28	47 46 23
- Marco Metzgerei Main-19	9 89 66 55	- Paula u. Robert Schreiner/Mstr.	75 55 66
.....	Fax 9 89 66 33	Bosch-1	

Lesen und Schreiben lernen ist kompliziert

- Extra Regeln für Gedichte
- ... und Telefonbücher
- und was ist mit anderen Sprachen?



Lesen und Schreiben lernen ist kompliziert

- Verschiedene Varianten
- Viele Möglichkeiten
- Mehrere Ebenen

XML: Erste Schritte

Syntax: Ein Beispiel

```
<?xml version="1.0"?>
<gedicht>
  <autor>Joachim Ringelnatz</autor>
  <titel>Die Ameisen</titel>
  <vers>
    <zeile>In Hamburg lebten zwei Ameisen,</zeile>
    <zeile>Die wollten nach Australien reisen.</zeile>
    <zeile>Bei Altona auf der Chaussee</zeile>
    <zeile>Da taten ihnen die Beine weh,</zeile>
    <zeile>Und da verzichteten sie weise</zeile>
    <zeile>Dann auf den letzten Teil der Reise.</zeile>
  </vers>
</gedicht>
```

Syntaxregeln 1

- XML-Deklaration am Anfang
- Baum von Elementen
 - ◆ Nur 1 Root-Element!
- Öffnende und schließende Tags

Syntaxregeln 2: Attribute

```
<?xml version="1.0"?>
<buch isbn="3873290995" jahr="1992">
  <autor id="jr17">Joachim Ringelnatz</autor>
  <titel sprache='de'>Gedichte - Prosa</titel>
  <verlag>Henssel</verlag>
  <kategorie name="Gedichte"/>
</buch>
```

- Werte in " oder ' einschließen
- Leere Elemente mit abschließendem /

Syntaxregeln 3: Entities

```
<?xml version="1.0"?>
<firma>
  <name>Heckler & Koch</name>
</firma>
```

- Eingebaute spezielle Zeichen:

```
< &lt;
> &gt;
" &quot;
' &apos;
& &amp;
```

- Eigene Definitionen möglich

Syntaxregeln 4: Kommentare und Pls

```
<?xml version="1.0"?>  
<?xml-stylesheet href="chrome://global/skin/"?>  
<window>  
  <!-- Hier kommt der Inhalt -->  
</window>
```

- Kommentare in <!-- -->
- Processing instructions in <? ?>

Syntaxregeln 4: CDATA

```
<?xml version="1.0"?>
<text>
  <![CDATA[
Dies ist viel "Text" mit <tags> drin.
  ] ]>
</text>
```

- längere Passagen mit speziellen Zeichen

Syntaxregeln 5: Erlaubte Zeichen

- In Elementnamen, Attributenamen, etc. erlaubt:
 - ◆ a-z, A-Z
 - ◆ 0-9
 - ◆ _ (Underscore)
 - ◆ . (Punkt)
 - ◆ - (Bindestrich)
 - ◆ Aber auch andere "Buchstabenzeichen" wie ä, µ, ... (Unicode!)
- Bezeichner sind case-sensitiv (!= HTML)
- Darf nicht mit "xml" anfangen

Unicode

- XML nutzt überall Unicode
- Default encoding ist UTF-8
- Explizit angegeben:

```
<?xml version="1.0" encoding="utf-8"?>  
<foo />
```

アカサタナハマヤラワン
イキシチニヒミリ
ウクスツヌフムユル
エケセテネヘメレ
オコソトノホモヨロヲ

- Oder mit ISO Latin1:

```
<?xml version="1.0" encoding="iso-8859-1"?>  
<foo />
```

Es ist da und doch nicht zu sehen: Whitespace

- SPACE (0x20), CR (0x0d), LF (0x0a), ...
- unwichtig:

```
<autor id="jr17">Joachim Ringelnatz</autor>
<autor id="jr17">
  Joachim Ringelnatz
</autor>
```

- wichtig:

```
<a href="autor.html?id=jr17">
   Joachim Ringelnatz
</a>
```

Dokumenten- vs. Datenorientierung

- XML eignet sich für zwei eigentlich sehr verschiedene Dinge:
 - ◆ 1. Als Auszeichnungssprache für Dokumente

```
<document>
  <author>James Bond</author>
  <title>Report on Mr. Evil</title>
  <chapter id="chap1">
    <title>World Domination</title>
    <para align="center">
      ...
    </para>
  </chapter>
</document>
```



Dokumenten- vs. Datenorientierung

- XML eignet sich für zwei eigentlich sehr verschiedene Dinge:
 - ◆ 2. Als Format für Datensammlungen

```
<briefmarkensammlung>
  <briefmarke id="bm1783" zustand="perfekt">
    <land>Mauritius</land>
    <jahr>1847</jahr>
    <name>Die Blaue Mauritius</name>
  </briefmarke>
  <briefmarke id="tz1230">
    ...
  </briefmarke>
  ...
</briefmarkensammlung>
```



More whitespace

- In der Regel ist Whitespace...
 - ◆ ... unwichtig für datenorientiertes XML
 - ◆ ... (mehr oder weniger) wichtig für documentenorientiertes XML
- Spezielles Attribut: `xml:space`
- Formale Beschreibungen...



Formale Beschreibung

DTD - Document Type Definition

- Welche Elemente und Attribute dürfen wo erscheinen?
- Default-Werte

```
<!ELEMENT briefmarkensammlung (briefmarke+)>  
<!ELEMENT briefmarke (land, jahr?, name?)>  
<!ELEMENT land (#PCDATA)>  
<!ELEMENT jahr (#PCDATA)>  
<!ELEMENT name (#PCDATA)>
```

```
<!ATTLIST briefmarke zustand (perfekt|gut|schlecht) "gut">
```

```
<!ENTITY dbp "Deutsche Bundespost">
```

- ◆ (Indirekt auch eine Aussage zu Whitespace)

Definition von Entities

- Entities sind eine Art Macro
- DTD enthält:

```
<!ENTITY dbp "Deutsche Bundespost">
```

- XML-Dokument kann dann benutzen:

```
<herausgeber>&dbp; </herausgeber>
```

DOCTYPE-Deklaration

- Verweis von einem XML-Dokument auf eine DTD

```
<?xml version="1.0"?>
<!DOCTYPE briefmarkensammlung SYSTEM
    "http://die.briefmarke.de/sammlung.dtd">
<briefmarkensammlung/>
```

```
<?xml version="1.0"?>
<!DOCTYPE html PUBLIC
    "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "DTD/xhtml1-transitional.dtd">
<html/>
```

- Auch inline möglich

DTD ist nicht gut genug

- Nicht flexibel genug
- Typdefinitionen nicht fein genug
- Kein XML-Format
- Kein Namespace-Support

Stattdessen: XML Schema

- Mega-flexibel
- Feine Typdefinitionen
- Hyperkompliziert
- Ist ein W3C-Standard, aber niemand mag es
- Vielleicht doch besser etwas anderes?
 - ◆ Schematron
 - ◆ Relax NG
 - ◆ ...

Und wozu das ganze?

- Automatische Validierung
 - ◆ wohlgeformt
 - ◆ validiert



Wohlgeformt vs. validiert

- Wohlgeformt ("well-formed") heißt:
- Das Dokument entspricht den XML-Syntax-Regeln



Wohlgeformt vs. validiert

- Validiert ("validated") heißt:
- Das Dokument entspricht einer DTD bzw. einem Schema.



Und wozu das ganze?

- Automatische Validierung
- Nutzung von Default-Werten
- Nutzung von Entities
- Codegenerierung

Namespaces

Das Problem

```
<?xml version="1.0"?>
<document>
  <title>XML im Wandel der Zeiten</title>
  <author>
    <title>Dr.</title>
    <first-name>Xaver</first-name>
    <last-name>Xenophon</last-name>
  </author>
  <body>
    ...
  </body>
</document>
```

- Fällt jemandem was auf?

title vs. title

- Titel des Dokuments

```
<title>  
  XML im Wandel der Zeiten  
</title>
```

- Titel des Autors

```
<title>Dr.</title>
```



Die Lösung: Namespaces

```
<author:author
  xmlns:author="http://www.example.com/xml/author/">
  <author:title>Dr.</author:title>
  <author:first-name>Xaver</author:first-name>
  <author:last-name>Xenophon</author:last-name>
</author:author>
```

- Deklaration mit "xmlns:foo"
- Benutzung mit "foo:element"
- Entscheidend ist die URL
- Der Namespace-Prefix ist in einem Dokument überall unterhalb der Deklaration nutzbar

Die Lösung: Namespaces

```
<brezel:author
  xmlns:brezel="http://www.example.com/xml/author/">
  <brezel:title>Dr.</brezel:title>
  <brezel:first-name>Xaver</brezel:first-name>
  <brezel:last-name>Xenophon</brezel:last-name>
</brezel:author>
```

- Die Namen sind willkürlich!
- In der Praxis aber häufig quasi-standardisiert

Default-Namespace

- Der Default-Namensraum wird mit dem xmlns-Attribut definiert

```
<?xml version="1.0"?>
<document xmlns="http://www.example.com/xml/doc/">
  <title>...</title>
  <author:author
    xmlns:author="http://www.example.com/xml/author/">
    <author:title>Dr.</author:title>
  </author:author>
</document>
```

Attribute und Namespaces

- Attribute können in einem eigenen Namespace sein
- Attribute erben den Namensraum von ihrem Element

```
<?xml version="1.0"?>
<html:html xmlns:html="http://www.w3.org/1999/xhtml">
  <html:body xmlns:content="...">
    <html:p
      align="center"
      content:important="false"
      xml:lang="de">
      Die einzige Möglichkeit, das Spiel zu gewinnen,
      ist die, es nicht zu spielen.
    </html:p>
  </html:body>
</html>
```

Namespaces

- ... erlauben das zusammenpuzzeln von XML-Files
- ... vertragen sich nur schlecht mit DTDs
- ... sind nicht immer nötig
- ... aber häufig unerlässlich
- ... und extrem cool



XPath und XSLT

XPath

- dient zum Bezeichnen von Bestandteilen eines XML-Baums
- Beispiele:

```
<warehouse>  
  <item id="ta318" cat="food" num="17">500g Butter</item>  
  <item id="xa911" cat="comp" num="3">USB-Maus</item>  
</warehouse>
```

- XPath-Ausdrücke dazu:

```
/warehouse/item  
item  
/warehouse/item/@id  
/warehouse/item[@cat='comp']  
item[@num<10]  
item[position()=last()]
```

XPath-Datenmodell

- Root-Node
- Element-Nodes
- Attribute-Nodes
- Text-Nodes
- ...

- Ein XPath-Ausdruck gibt zurück:
 - ◆ Einen String,
 - ◆ eine Zahl,
 - ◆ einen Boolean-Wert oder
 - ◆ einen Node-Set

XPath-Pfade (1)

- element
- element/kind
- /element
- /element/kind
- Wie in Directories:
 - ◆ /
 - ◆ .
 - ◆ ..

XPath-Pfade (2)

- Attribute: element/@attribut
- Text: element/text()
- Kommentare: element/comment()
- Alle: element/node()

XPath-Pfade (3)

- `element/*/enkel`
- `element//kindeskinder`
- `element/following-sibling::bruder`

XPath-Pfade (4)

- `element[3]`
- `element[last()]`
- `element[position() mod 2 = 0]`
- `element[@attribute='true']`
- `element[@attribute=/foo/@bar]`

XSLT (Extensible Stylesheet Language - Transformations)

- XSLT ist die Beschreibung einer Transformation
- Ein XSLT-Parser
 - ◆ liest ein XML file,
 - ◆ liest ein XSLT file und
 - ◆ schreibt ein XML oder Text file

XSLT-Beispiel: XML-Source

```
<?xml version="1.0"?>
<adressbuch>
  <adresse>
    <name>Dr. Hannibal Lecter</name>
    <email>honey@lecter.com</email>
  </adresse>
  <adresse>
    <name>Jack the Ripper</name>
    <email>jackie@ripper.co.uk</email>
  </adresse>
</adressbuch>
```



XSLT-Beispiel: XSLT-Code

```
<?xml version="1.0"?>
<xsl:stylesheet
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  version="1.0">

  <xsl:template match="adressbuch">
    <table>
      <xsl:apply-templates select="adresse"/>
    </table>
  </xsl:template>

  <xsl:template match="adresse">
    <tr>
      <td> <xsl:value-of select="name"/> </td>
      <td> <xsl:value-of select="email"/> </td>
    </tr>
  </xsl:template>

</xsl:stylesheet>
```

XSLT-Beispiel: Ergebnis

```
<?xml version="1.0"?>
<table>
  <tr>
    <td>Dr. Hannibal Lecter</td>
    <td>honey@lecter.com</td>
  </tr>
  <tr>
    <td>Jack the Ripper</td>
    <td>jackie@ripper.co.uk</td>
  </tr>
</table>
```



XSLT kann noch mehr

- nutzt die vollen Fähigkeiten von XPath
- weitere Dateien einbinden
- kann sortieren
- hat Variablen und Parameter
- if-then-else ähnliche Konstrukte
- man kann sogar XSLT-Stylesheets mit XSLT erzeugen
- ...

Weiterer Buchstabensalat

XInclude

- Einbinden von externen Dateien in ein Dokument

```
<?xml version="1.0"?>
<document xmlns:xi="http://www.w3.org/2001/XInclude">
  <xi:include href="example.xml">
    <xi:fallback>
      Include-File nicht gefunden.
    </xi:fallback>
  </xi:include>
</document>
```

XLink und XPointer

- Zum verlinken von beliebigen Dokumenten
- Ähnlich wie in HTML aber auf beliebigen Elementen

```
<?xml version="1.0"?>
<document xmlns:xlink="...">
  <element id="foo" xlink:type="simple"
           xlink:href="http://www.example.org/" />
  <ref xlink:type="simple" xlink:href="#foo" />
</document>
```

- XPointer: href="#xpointer(/element/path/)"
- Auch "ranges" möglich

XML Encryption und Signing

- Verschlüsseln und/oder Signieren von XML-Dokumenten oder Teilen davon
- Ergebnis ist selber wieder XML-konform
- Braucht Canonical XML

Canonical XML

- Semantisch gleich, aber textuell verschieden
- Beispiele:

```
<foo></foo>  
<foo/>
```

```
<x:foo xmlns:x="http://www.example.com/" />  
<y:foo xmlns:y="http://www.example.com/" />
```

XML-Anwendungen

XML-Anwendungen

- XHTML
- MathML
- VoiceML
- XForms
- SVG (Scalable Vector Graphics)
- XUL (Mozilla)

XHTML

- HTML als Anwendung von XML neu formuliert
- Strengere XML-Syntaxregeln
 - ◆ Elementschachtelung
 - ◆ Leere Elemente: `
`
 - ◆ Attribute in " einschließen
 - ◆ Keine leeren Attribute: `<option selected="selected">`
- Wird von allen Browsern verstanden
 - ◆ (Leerzeichen vor / beachten!)

XHTML

- Vorteile:
 - ◆ Eindeutige Syntax führt zu verlässlicherem Rendering
 - ◆ Alle XML-Tools können verwendet werden
- Modularisierung (XHTML 1.1)
- Verknüpfung mit anderen XML-Sprachen:
 - ◆ MathML
 - ◆ SVG
 - ◆ XForms

Kleine Exkursion: Stylesheets

- CSS (Cascading Stylesheets)
 - ◆ Nutzbar für HTML, XHTML und XML im Browser

```
p { text-align: right; color: blue }
```

- XSL ("Extensible Stylesheet Language")
 - ◆ XSLT ("Transformation")
 - ◆ XSL-FO ("Formatting Objects")

```
<fo:block text-align="right" color="blue"/>
```

XUL und Mozilla

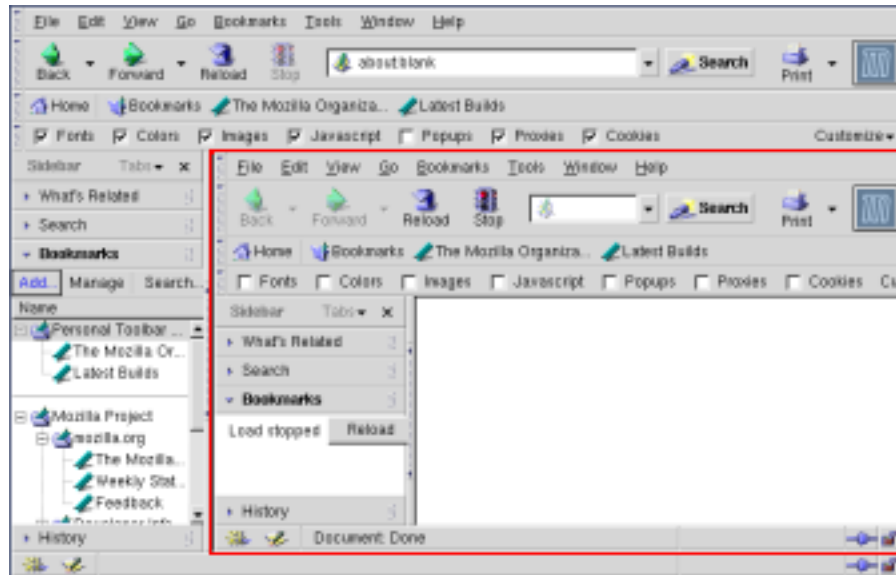
- Die Benutzeroberfläche von Mozilla ist komplett mit XML und Javascript realisiert
 - ◆ XML-Files enthalten die Beschreibung der Oberfläche
 - ◆ Javascript-Files die Funktionalität
 - ◆ Entities für die Lokalisierung

Ein XUL-Beispiel

```
<?xml version="1.0"?>
<?xml-stylesheet href="chrome://global/skin"
                 type="text/css"?>
<window title="Hello World"
        xmlns="http://www.mozilla.org/keymaster/
              gatekeeper/there.is.only.xul"
        xmlns:html="http://www.w3.org/1999/xhtml">
  <script type="application/x-javascript"
        src="chrome://global/content/dialogOverlay.js"/>
  <vbox align="left">
    <label value="Hello World"/>
    <button label="Press me"
           oncommand="alert('Hello World!');"/>
    <html:h1>Hello HTML</html:h1>
  </vbox>
</window>
```

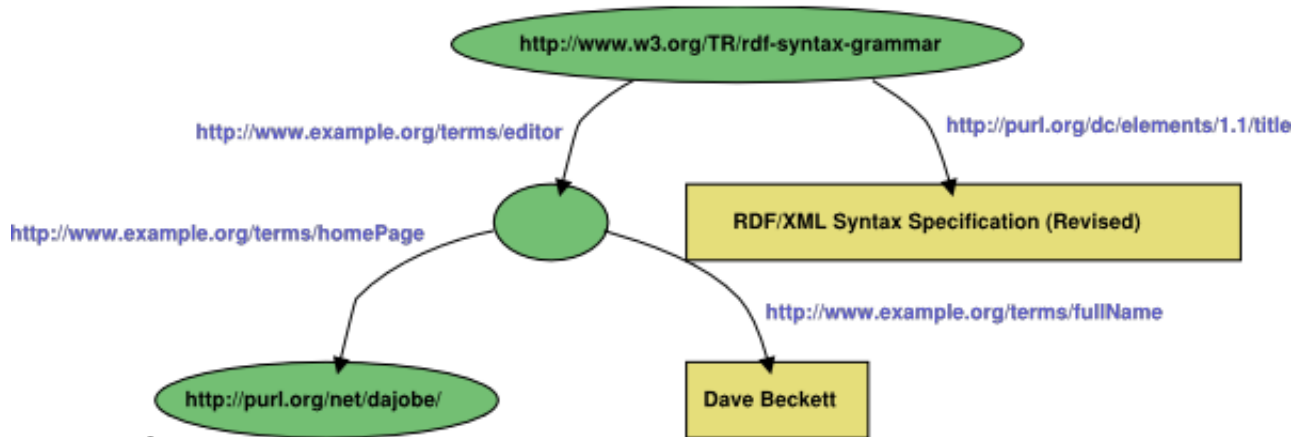
Mozilla Rendering-Engine

- Nur eine Rendering-Engine für "Chrome" und HTML
 - ◆ `chrome://navigator/content/`



RDF (Resource Description Framework)

- Serialisierung von Tripeln: Subjekt - Prädikat - Objekt
- Alle Teile können URLs sein



- "Semantic Web"

RSS (Rich Site Summary)

- Zusammenfassung von Newsmeldungen
- z.B. Slashdot:

```
<?xml version="1.0"?>
<rdf:RDF ...>
  <channel>
    <title>Slashdot</title>
    <link>http://slashdot.org/</link>
    <description>
      News for nerds, stuff that matters
    </description>
  </channel>
  <item>
    <title>Linus Torvalds On Linux 2.6</title>
    <link>http://slashdot.org/article.pl?sid=...</link>
  </item>
  ...
</rdf:RDF>
```

Andere Anwendungen

- ebXML
- OpenOffice.org Datei-Format
- ...

Programmiermodelle

SAX (Simple API for XML)

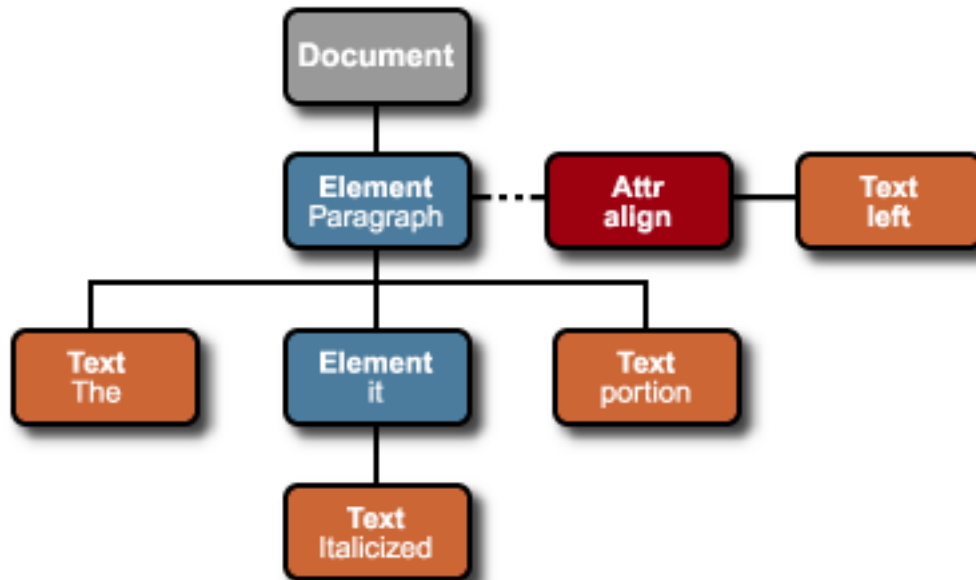
- Über Sprachen hinweg standardisiert
- Callback-Modell
 - ◆ `start_document()`, `end_document()`
 - ◆ `start_element()`, `end_element()`
 - ◆ `characters()`, `comment()`, ...
- "nie wieder Parser schreiben"

SAX

- Vorteile:
 - ◆ Einfach
 - ◆ Schnell
 - ◆ Kaum Speicherbedarf
 - ◆ Streaming-fähig
- Nachteile:
 - ◆ Viel Handarbeit erforderlich
 - ◆ Zugriff auf nicht-lokale Daten unmöglich
- Gut für große Dokumente
- Gut als Basis für andere Toolkits
- SAX als allgemeines Streaming-Modell

DOM (Document Object Model)

- XML-Datei wird als Baum von Objekten dargestellt



DOM

- Vorteile:
 - ◆ Einfach zu handhaben
 - ◆ Alle Daten liegen einfach zugreifbar vor
 - ◆ Über Sprachen hinweg standardisiert
- Nachteile:
 - ◆ Braucht viel Speicher
 - ◆ Das ganze Dokument muß erst eingelesen werden

Programmiermodelle

- SAX für einfache Transformationen
- DOM für komplexe Arbeiten
- Aufbauten auf DOM: JDOM, ...
- XPath
- XSLT
- "W3C-DOM" für HTML-Seiten

XML und Datenbanken

XML und Datenbanken

- XML-Datenmodell
 - ◆ XML speichert Daten in einem Baum
 - ◆ != Tabellen in relationalen Datenbanken
- Speicherformat
 - ◆ Text so wie er reinkommt
 - ◆ Nur semantische Struktur



XML und relationale Datenbanken

- Export aus SQL-Datenbanken sehr einfach
- Allgemeine Abbildung von einem rekursiven Baum in SQL schwierig
- Sinnvolle Queries nicht mehr möglich
- Import in der Regel nicht ohne zusätzliches Wissen möglich
- Ergebnis: Im- und Export zu relationalen Datenbanken ok, aber die Abbildung von allgemeinen XML-Dateien auf relationale Datenbanken macht keinen Sinn

XML und "objektorientierte" Datenbanken

- Was ist eine OO-Datenbank überhaupt?
- Geht im Prinzip
- Aber: Sehr viele kleine Objekte
- Abfragesprachen?

Spezielle XML-Datenbanken

- Unabhängig von DTDs/Schemata
- Hat passende Interfaces: DOM, XPath
- Query per XPath und XQL
- Dokumentenübergreifende Queries
- Spezielle Indices für Geschwindigkeit
- Entwicklung noch sehr im Fluß



XML zur Transportcodierung

XML zur Transportcodierung

- XML-RPC (Remote Procedure Call)
- SOAP (Simple Object Access Protocoll)

SOAP-Anbieter

- Google
- Amazon
- www.xmethods.net

Beispiel: EBay-Preis und Umrechnung

```
#!/usr/bin/perl

use SOAP::Lite;

$auction="2073219671";

$soap1 = SOAP::Lite -> service("http://www.xmethods.net/
                               sd/2001/EBayWatcherService.wsdl");
$biddollar = $soap1->getCurrentPrice($auction);

$soap2 = SOAP::Lite -> service("http://www.xmethods.net/
                               sd/2001/CurrencyExchangeService.wsdl");
$rate = $soap2->getRate("usa", "euro");

printf "current bid in US-\$: %.2f\n", $biddollar;
printf "current bid in EUR:  %.2f\n", $biddollar * $rate;
```

Beispiel: Ergebnis

current bid in US-\$:	87.00
current bid in EUR:	87.91



Beispiel: Hinter den Kulissen

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
  SOAP-ENV:encodingStyle=
    "http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/1999/XMLSchema">
  <SOAP-ENV:Body>
    <namespace1:getCurrentPrice
      xmlns:namespace1="urn:xmethods-EbayWatcher">
      <auction_id xsi:type="xsd:string">
        2073219671
      </auction_id>
    </namespace1:getCurrentPrice>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Ist das nicht ein bisschen kompliziert?

- Eigentliche Info versinkt im Overhead
- Sowas möchte man nicht mehr "von Hand" schreiben
- Dafür gibt es wieder eine XML-Datei zur Beschreibung...

Beispiel: Die WSDL-Datei

```
<?xml version="1.0"?>
<definitions name="CurrencyExchangeService" [...]
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  <message name="getRateRequest">
    <part name="country1" type="xsd:string"/>
    <part name="country2" type="xsd:string"/>
  </message>
  <message name="getRateResponse">
    <part name="Result" type="xsd:float"/>
  </message>
  <portType name="CurrencyExchangePortType">
    <operation name="getRate">
      <input message="tns:getRateRequest" />
      <output message="tns:getRateResponse" />
    </operation>
  </portType>
  [...]
</definitions>
```

SOAP vs. HTTP GET/POST

- Was ist denn falsch mit diesem hier?

```
GET /getCurrentPrice.xml?auction_id=2073219671 HTTP/1.0
```

- Small is beautiful
- Jede Resource = Eine URL (Links!)
- Browser reicht
- Eierlegende Wollmilchsau
- XML-Namensräume etc.
- Applikation erforderlich

The Making of ...

The Making of ...

- XML::Handler::AxPoint
- Source XML

```
<slide>  
<title>The Making of ...</title>  
<point level="1">XML::Handler::AxPoint</point>  
...
```

- "Compiliert" nach PDF

Zusammenfassung

Warum XML?

- Arbeitserleichterung
 - ◆ Nie wieder Parser schreiben
 - ◆ Klare Escape-Regeln
 - ◆ Automatische Validierung
- Unicode-Unterstützung
- Flexibel
- Dateien weitgehend selbsterklärend
- "Von Hand" les- und schreibbar
- Standard

Warum XML?

- Weil man sich aus vielen Teilen die passenden Stücke zusammensuchen kann
- Ideal als gemeinsames Datenaustausch-Format zwischen Applikationen



XML und ...

- ... SGML
 - ◆ SGML ist wesentlich komplizierter als XML
- ... HTML
 - ◆ HTML ist nicht annähernd so flexibel
- ... CSV
 - ◆ CSV ist zu simpel
- ... relationales Datenmodell
 - ◆ XML ist flexibler, einfach erweiterbar

XML ist nicht geeignet...

- für sehr einfache Dinge (z.B. CSV verwenden)
- für Binärdaten (SOAP!)
- für Aufgaben, die einen geringen Overhead brauchen
- für Aufgaben, die sehr schnell sein müssen

Weitere Informationen

- W3C (www.w3.org)
- O'Reilly (www.xml.com)
- Apache Project (xml.apache.org)
- IBM developerWorks (www.ibm.com/developerworks/xml/)
- SAX (www.saxproject.org)

Fragen?

Jochen Topf

- jochen@remote.org
- <http://www.remote.org/jochen/>