

- Was ist XML?
- Wie verhält sich diese Buchstabensuppe aus SGML, HTML, XML, XHTML zueinander?
- Was sollen Sie am Ende dieser Vorlesung gelernt haben?



Aufbau von XML-Dokumenten

Robert Tolksdorf
Freie Universität Berlin
Institut für Informatik
Netzbasierende Informationssysteme
markus.luczak-roesch@fu-berlin.de

Ziel der heutigen Vorlesung

- Sie wissen wie wohlgeformte XML-Dokumente aufgebaut sind.
- Wir besprechen dazu
 - XML-Syntax
 - Elemente
 - Attribute
 - Deklaration
 - ...
 - Namensräume

Wiederholung: Was ist XML?



- XML ist eine Methode, um strukturierte Daten in einer Textdatei darzustellen.



- XML ist Text, aber nicht zum Lesen.



- XML ist eine Familie von Techniken.



- XML ist modular



- XML ist lizenzfrei und plattformunabhängig





XML-Syntax: XML-Dokumente

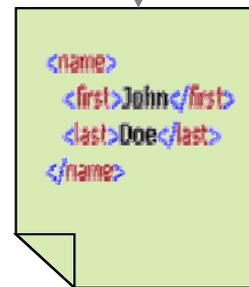
Was ist ein XML-Dokument?

Inhalt: Text oder Daten



kodiert als

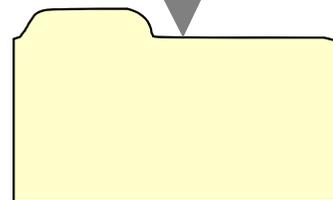
XML-Dokument



Objekt, das
Syntaxregeln von XML
entspricht (wohlgeformt
ist)

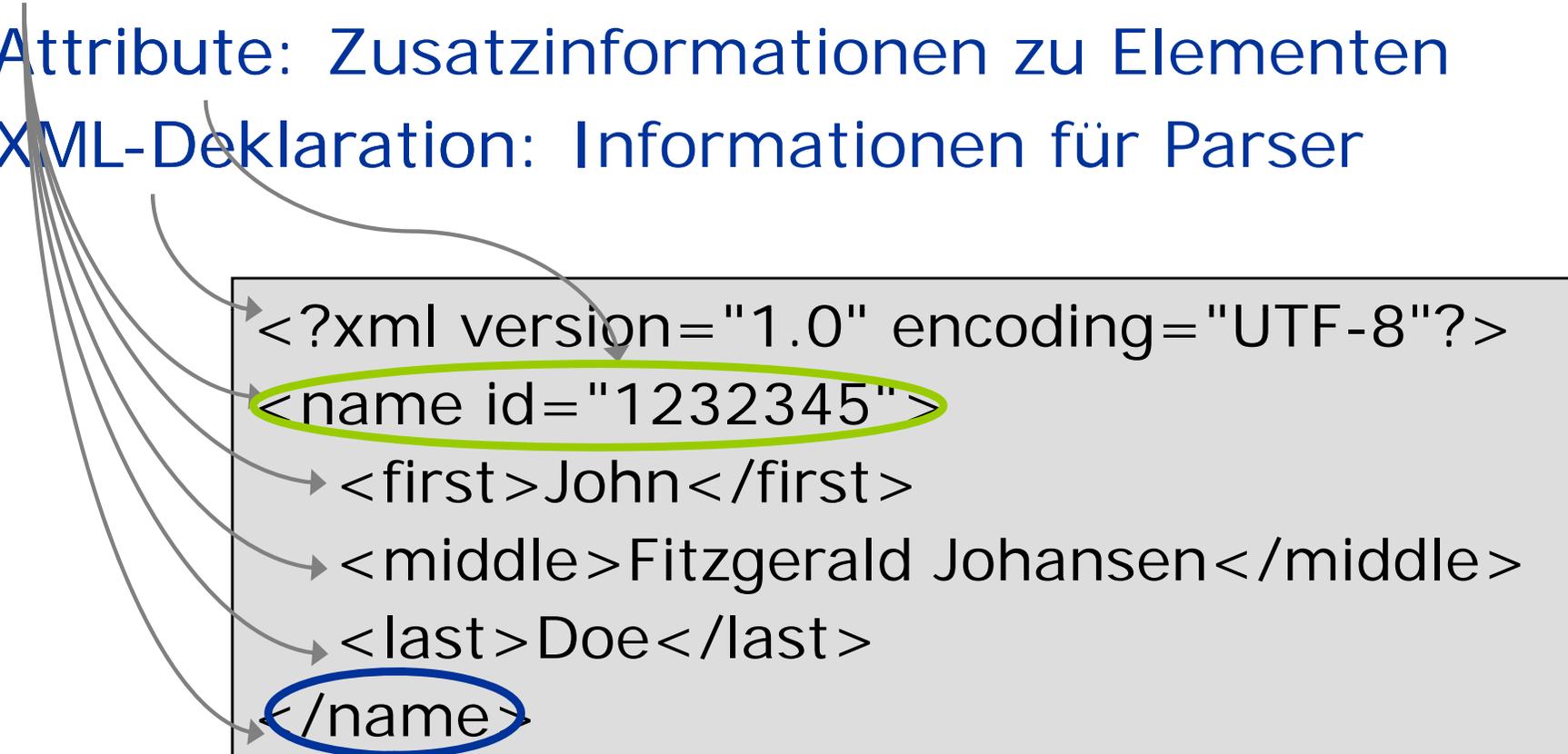
gespeichert in

XML-Datei

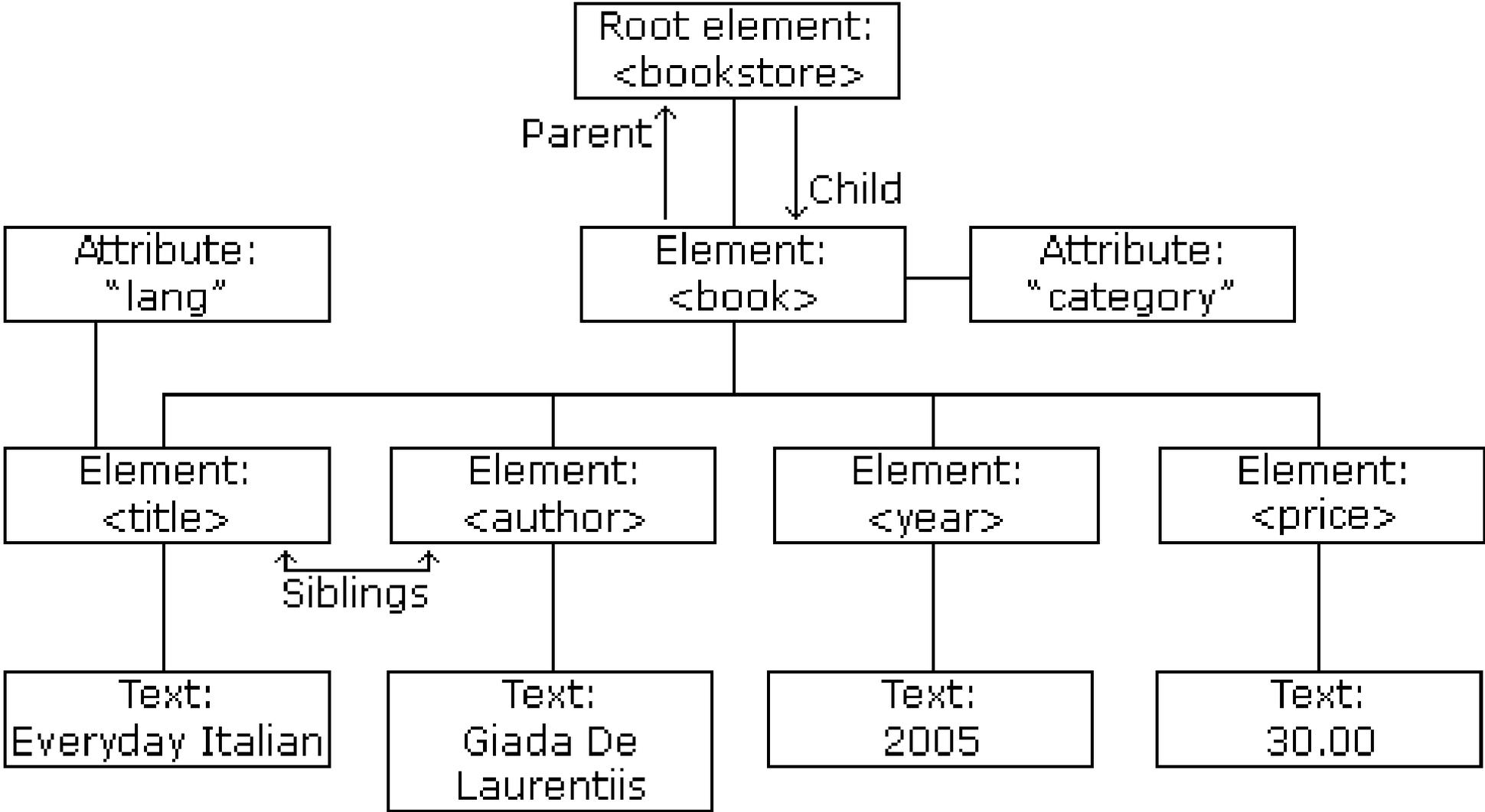


- Elemente: strukturieren das XML-Dokument
- Attribute: Zusatzinformationen zu Elementen
- XML-Deklaration: Informationen für Parser

```
<?xml version="1.0" encoding="UTF-8"?>  
<name id="1232345">  
  <first>John</first>  
  <middle>Fitzgerald Johansen</middle>  
  <last>Doe</last>  
</name>
```



XML-Baumstruktur



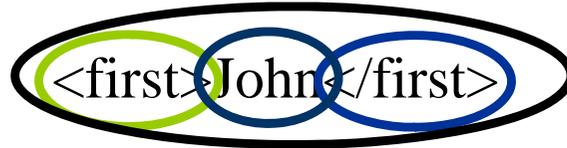
Beispiel von http://www.w3schools.com/xml/xml_tree.asp



XML-Syntax

Grundbausteine: XML-Elemente

- Beispiel:



- besteht aus:

- einem **Anfangs-Tag** (engl. *start tag*): hier `<first>`
- einem dazugehörigen **Ende-Tag** (engl. *end tag*): hier `</first>`
- einem **Inhalt**: hier „John“

- alles zusammen bildet ein Element

- haben einen Namen: hier „first“

1. **unstrukturierter Inhalt:**
 - einfacher Text ohne Kind-Elemente
2. **strukturierter Inhalt:**
 - Sequenz von $> \emptyset$ Kind-Elementen
3. **gemischter Inhalt:**
 - enthält Text mit mind. einem Kind-Element
4. **leerer Inhalt**

1. Unstrukturierter Inhalt

- Beispiel: `<first>John</first>`
- einfacher Text ohne Kind-Elemente
Kind-Element: Element, das im Inhalt eines Elementes vorkommt
- unstrukturierter Inhalt → *Parsed Character Data (PCDATA)*:
 - character data: einfache Zeichenkette
 - parsed: Zeichenkette wird vom Parser analysiert, um Ende-Tag zu identifizieren
 - Normalisierung: u.a. Zeilenumbruch (CR+LF) → `
`

Anmerkung: Auf den Folien schreibe ich der besseren Lesbarkeit wegen *Kind-Element* statt *Kindelement*!

- Reservierte Symbole < und & in PCDATA nicht erlaubt.
- Symbole wie >, /, (,), {, }, [,], % allerdings erlaubt
- statt < und & Entity References & bzw. < benutzen
- Entity References in XML:

& ⇒ &

< ⇒ <

> ⇒ >

' ⇒ '

" ⇒ "

- Unstrukturierten Inhalt mit vielen reservierten Symbolen → Character Data (CDATA)

- Beispiel:

```
<formula>  
  <![CDATA[ X < Y & Y < Z ]]>  
</formula>
```

- Inhalt: String zwischen inneren Klammern []
hier: X < Y & Y < Z
- XML-Parser sucht in CDATA lediglich]]>, analysiert den Inhalt aber ansonsten nicht.
- "[]>" als Inhalt von CDATA nicht erlaubt

Nested CDATA?

- XML-Parser sucht in CDATA lediglich `]]>`, analysiert den Inhalt aber ansonsten nicht.*

```
<formula>  
  <![CDATA[<![CDATA[X < Y & Y < Z ]]]>]]>  
</formula>
```

CDATA Sections

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

“Within a CDATA section, only the CDEnd string is recognized as markup, so that left angle brackets and ampersands may occur in their literal form; they need not (and cannot) be escaped using "<" and "&". **CDATA sections cannot nest.**”

<http://www.w3.org/TR/xml11/#sec-cdata-sect>

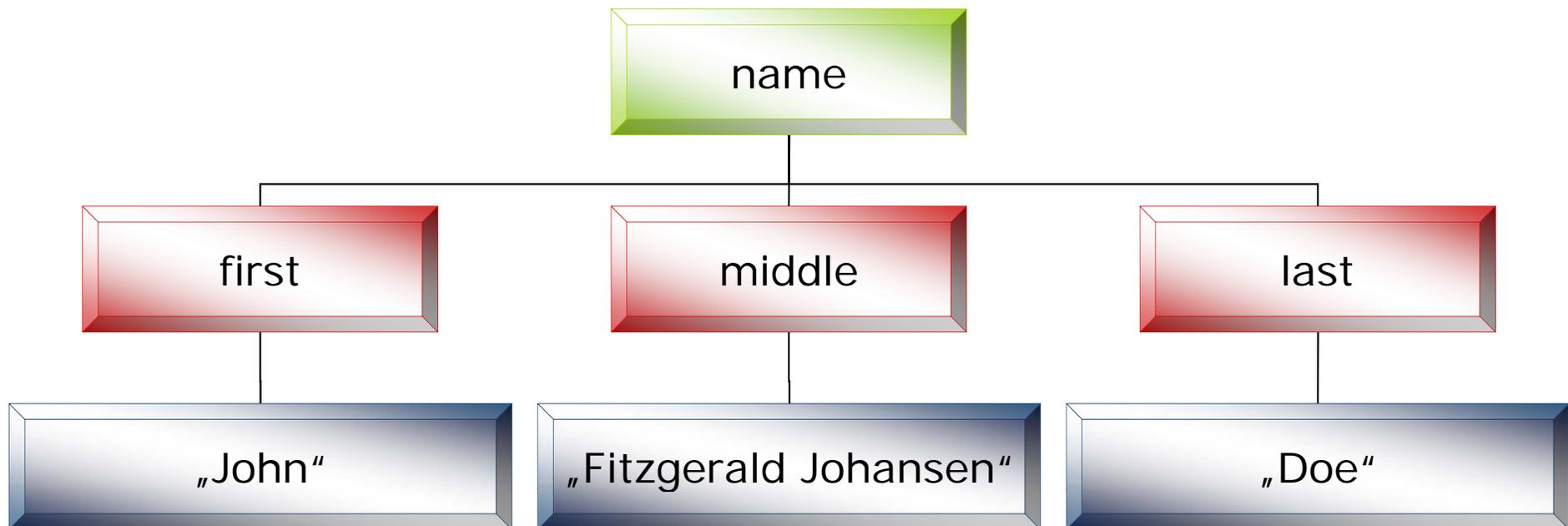
2. Strukturierter Inhalt

- Beispiel:

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

- Sequenz von > 0 Kind-Elementen:
 - hier: `<first>John</first>` und `<last>Doe</last>`
- kein Text vor, nach oder zwischen den Kind-Elementen
- Kind-Elemente immer geordnet:
Reihenfolge, so wie sie im XML-Dokument erscheinen
- Elemente können beliebig tief geschachtelt werden.

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



3. Gemischter Inhalt (mixed content)

- enthält Text mit mindestens einem Kind-Element
- Beispiel:

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

- Wie unterscheidet ein Parser strukturierten und gemischten Inhalt, wenn Text = leerer String?
- Antwort: Nur mit zugehöriger DTD oder XML-Schema möglich!

```
<letter>
  Dear Mr.<name>John Doe</name>.
  Your order <orderid>1234</orderid>
  will be shipped on <shipdate>2010-07-14</shipdate>.
</letter>
```

Element

Zugehöriges
Schema

```
<xs:element name="letter">
  <xs:complexType mixed="true">
    <xs:sequence>
      <xs:element name="name" type="xs:string"/>
      <xs:element name="orderid" type="xs:positiveInteger"/>
      <xs:element name="shipdate" type="xs:date"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Mixed Content mit DTD

```
<?xml version="1.0" standalone="yes"?>
  <!DOCTYPE rootelement [
    <!ELEMENT rootelement (#PCDATA|childelement1|childelement2)*>
    <!ELEMENT childelement1 (#PCDATA)>
    <!ELEMENT childelement2 (#PCDATA)> ]>
  <rootelement>
    PCDATA Content
    <childelement2>Child element 2.</childelement2>
  </rootelement>
```

Mixed Content ohne zugehöriges Schema?

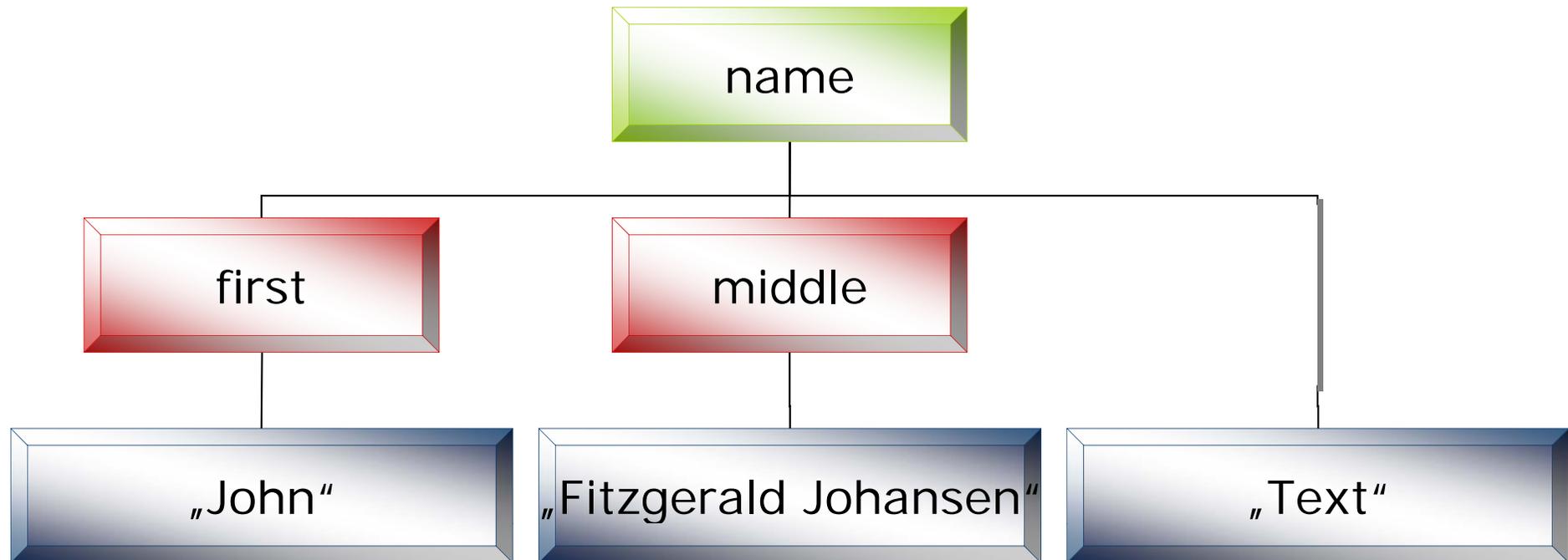
```
<letter>
  Dear Mr.<name>John Doe</name>.
  Your order <orderid>1234</orderid>
  will be shipped on <shipdate>2010-07-14</shipdate>.
</letter>
```

```
<xs:element name="letter">
  <xs:complexType mixed="true">
    <xs:sequence>
      <xs:element name="name" type="xs:string"/>
      <xs:element name="orderid" type="xs:positiveInteger"/>
      <xs:element name="shipdate" type="xs:date"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

```
<?xml version="1.0" standalone="yes"?>
  <!DOCTYPE rootelement [
  <!ELEMENT rootelement (#PCDATA|childelement1|childelement2)*>
  <!ELEMENT childelement1 (#PCDATA)>
  <!ELEMENT childelement2 (#PCDATA)> ]>
  <rootelement>
    PCDATA Content
    <childelement2>Child element 2.</childelement2>
  </rootelement>
```

Baumstruktur von XML

```
<name>  
  <first>John</first>  
  <middle>Fitzgerald Johansen  
  </middle>  
  Text  
</name>
```



4. Leerer Inhalt

- Beispiel:

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

- weder Text noch Kind-Element
- `<middle></middle>` auch **leeres Element** genannt
- Abkürzung: **selbstschließendes Tag** (engl.: *self-closing tag*) `<middle/>` :

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

Warum leere Elemente?

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

VS.

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

- leeres Element evtl. von DTD oder XML-Schema vorgeschrieben
- einfacher später mit Inhalt zu füllen
- leeres Element kann Attribute haben:
 <middle status="unknown"></middle> oder
 <middle status="unknown"/>



XML-Syntax

Grundbausteine: XML-Attribute

```
<name id="1232345" nickname="Shiny John" >  
  <first>John</first>  
  <last>Doe</last>  
</name>
```

- **Attribut: Name-Wert-Paar**
 - name="wert" oder name='wert' aber ~~name="wert"~~
- **Attribut-Wert:**
 - immer PCDATA: keine Kind-Elemente, kein < und &
 - "we"rt" und 'we'rt' ebenfalls nicht erlaubt
 - Normalisierung: u.a. Zeilenumbruch →

- **Beachte: Reihenfolge der Attribute belanglos**

Attribut statt Element

- Jedes Attribut auch als Kind-Element darstellbar:

```
<name id="12345">
  <first>John</first>
  <middle>Fitzgerald</middle>
  <last>Doe</last>
</name>
```



```
<name>
  <id>12345</id>
  <first>John</first>
  <middle>Fitzgerald</middle>
  <last>Doe</last>
</name>
```

id als Attribut

id als Kind-Element

Element statt Attribut

- Jedes Kind-Element mit **unstrukturiertem Inhalt** auch als Attribut darstellbar:

```
<name>  
  <id>12345</id>  
  <first>John</first>  
  <middle>Fitzgerald</middle>  
  <last>Doe</last>  
</name>
```

**id, first, middle und last
als Kind-Elemente**



```
<name id="12345"  
  first="John"  
  middle="Fitzgerald"  
  last="Doe" />
```

**id, first, middle und last
als Attribute**

Resultat: leeres Element

Attribut oder Element?

- Attribut kann nur einfachen String (PCDATA) als Wert haben, Element kann beliebig strukturiert sein
- `<![CDATA[...]]>` in Element-Inhalten erlaubt, nicht aber in Attribut-Werten
- Reihenfolge der Attribute belanglos, diejenige von Elementen nicht
- einheitliche Darstellung mit Elementen eleganter, Darstellung mit Attributen kompakter

Fazit: Attribute für einfache, unstrukturierte Zusatzinformationen (Metadaten) geeignet.

```
<name creation-date="21.05.2003">  
  <first>John</first>  
  <middle>Fitzgerald Johansen</middle>  
  <last>Doe</last>  
</name>
```

- Erstellungsdatum creation-date ist Zusatzinformation
 - falls noch andere Attribute vorhanden: Reihenfolge egal
- ⇒ Repräsentation als Attribut

```
<name creation-date="21.05.2003">  
  ...  
</name>
```

- Nachteil: Datum "21.05.2003" unstrukturiert

```
<xs:element name="creation-date" type="xs:date"/>
```

```
<creation-date>2002-09-24</creation-date>
```

Reservierte Attribute

■ xml:lang

- Sprache des Inhalts
- Beispiel: `<p xml:lang="de">Übung 1</p>`

■ xml:space

- Leerräume im Inhalt
- Beispiel: `<p xml:space="[preserve/default]">Übung 1</p>`

■ xml:id

- Elementbezeichner (dokumentweit eindeutig)
- Beispiel: `<p xml:id="Abschnitt_1">Ein Absatz</p>`

■ xml:base

- Basis-URL (für relative Links)
- Beispiel: `<ul xml:base="http://www.ag-nbi.de/lehre/10/">
 XML-Technologien
`



XML-Syntax

Grundbausteine: XML-Deklaration

```
<?xml version="1.0" encoding="UTF-8"?>
<name id="1232345">
  <first>John</first>
  <middle>Fitzgerald Johansen</middle>
  <last>Doe</last>
</name>
```

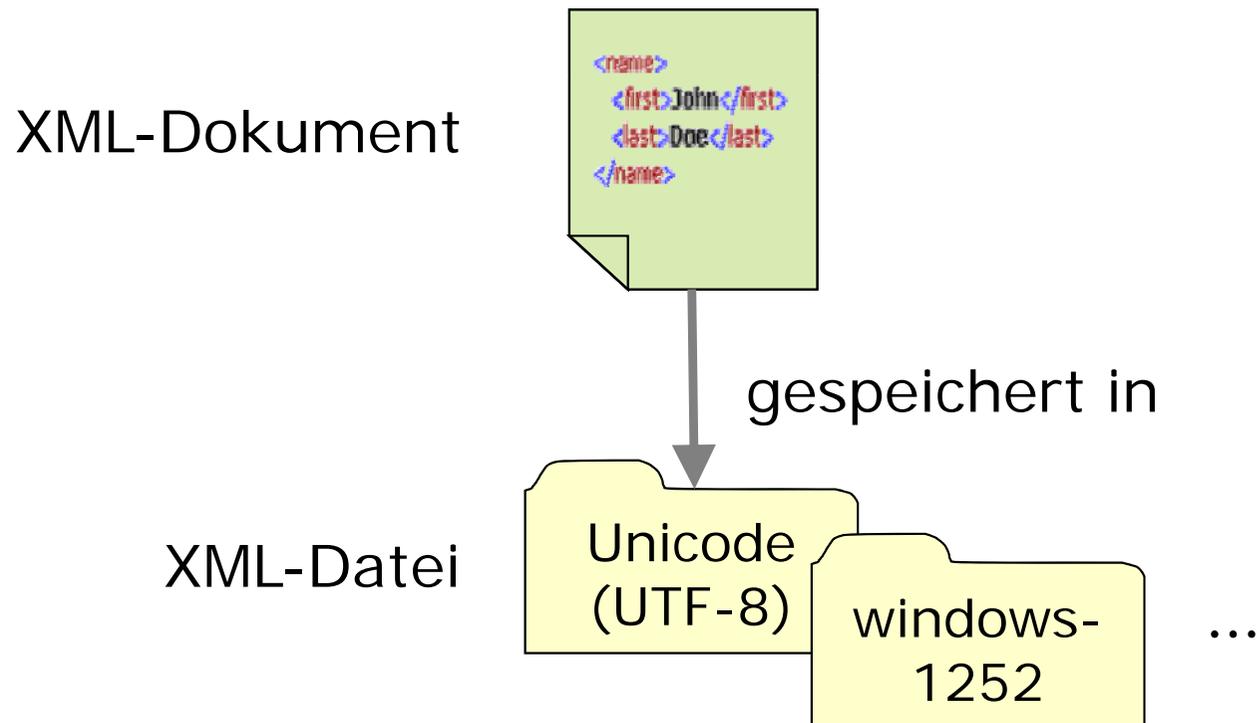
- enthält Informationen für Parser: z.B. verwendete XML-Version und Kodierung
- wenn vorhanden, dann immer am Anfang der Datei
- in XML 1.0 optional, in XML 1.1 obligatorisch

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

- **Attribut: version**
 - verwendete XML-Version: "1.0" oder "1.1"
 - obligatorisch
- **Attribut: encoding**
 - Kodierung der XML-Datei
 - Optional
- **Attribut: standalone**
 - Gibt an, ob es eine zugehörige DTD oder ein XML-Schema gibt ("no") oder nicht ("yes").

Beachte: immer in dieser Reihenfolge!

XML-Deklaration: Kodierung



XML-Deklaration: Kodierung

- XML-Parser
 - müssen intern mit Unicode (UTF-8 oder UTF-16) arbeiten
- Unicode
 - kann alle nationalen Zeichen darstellen: insgesamt ca. 65.000 Zeichen
- encoding-Attribut
 - Zeichenkodierung der XML-Datei
 - Fehlt das Attribut, dann wird Kodierung in Unicode angenommen.

Beachte: XML-Parser müssen nur Unicode verarbeiten können!



XML-Syntax

Andere Grundbausteine

Weitere Grundbausteine von XML

- **Kommentare**
 - `<!-- Kommentar -->`
 - `<!-- Kommen-
tar -->`
 - `--` in Kommentaren nicht erlaubt
- **Prozessorinstruktionen**
 - Beispiel: `<?mysql SELECT * FROM PO?>`
 - werden ungeparst an die Anwendung weitergegeben
 - selten benutzt



XML-Syntax

Wohlgeformte XML-Dokumente

1. Jedes Anfangs-Tag muss ein zugehöriges Ende-Tag haben.
2. Elemente dürfen sich nicht überlappen.
3. XML-Dokumente haben genau ein Wurzel-Element.
4. Element-Namen müssen bestimmten Namenskonventionen entsprechen.
5. XML beachtet grundsätzlich Groß- und Kleinschreibung.
6. XML belässt White Space im Text.
7. Ein Element darf niemals zwei Attribute mit dem selben Namen haben.

Regel 1: Anfangs- und Ende-Tags

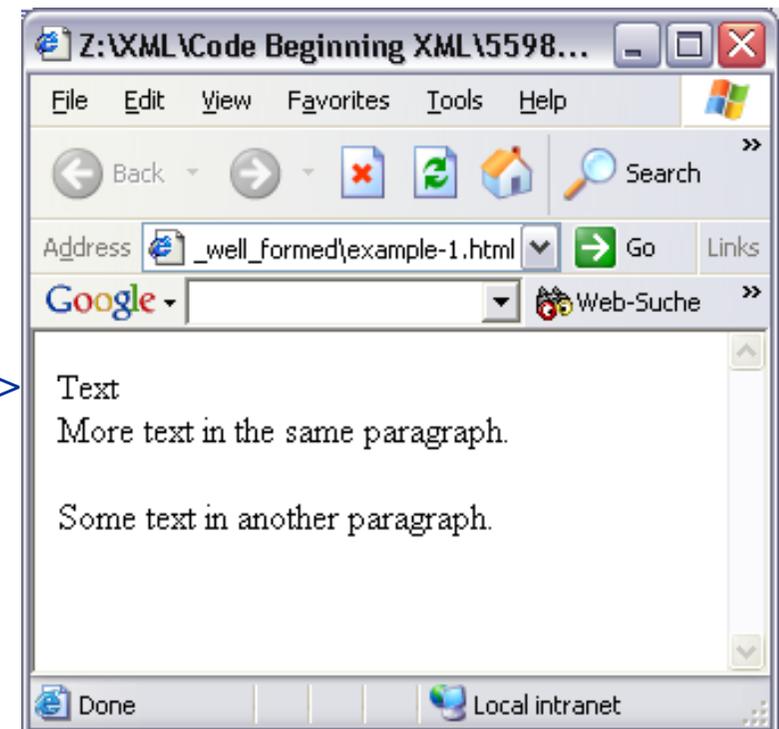
Jedes Anfangs-Tag muss zugehöriges Ende-Tag haben.

- 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?

⇒ HTML mehrdeutig



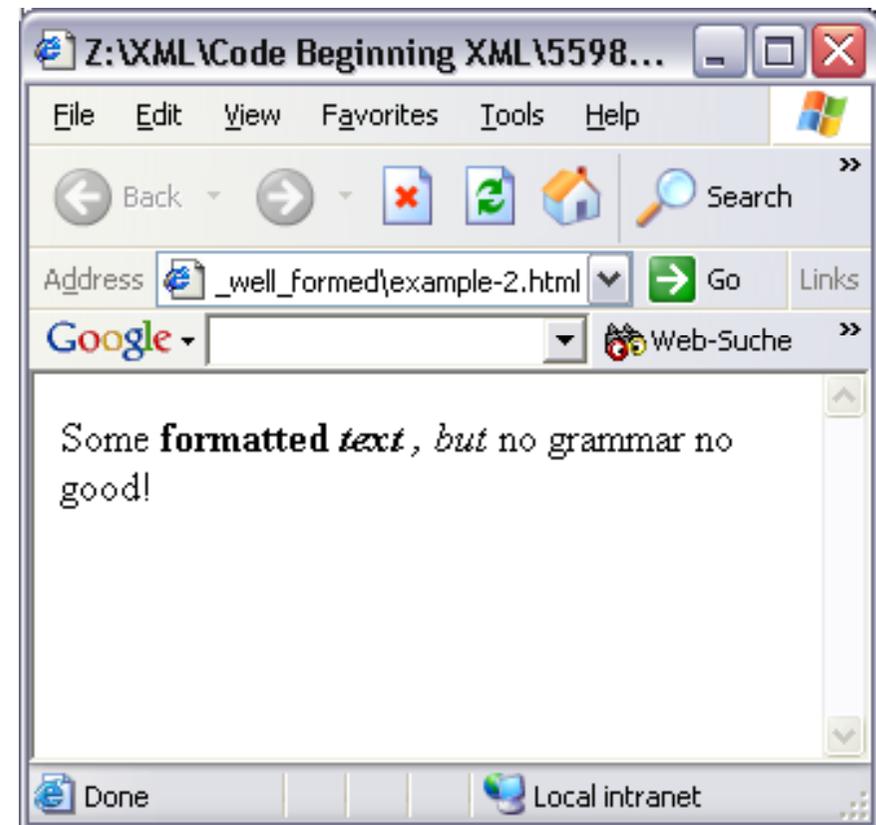
Regel 2: Elemente dürfen sich nicht überlappen

Elemente dürfen sich nicht überlappen.

- In HTML gilt diese Regel nicht:

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

⇒ HTML unstrukturiert



Regel 3: Wurzel-Element

Jedes XML-Dokument hat genau ein Wurzel-Element.

- Also z.B. statt zweier Wurzel-Elemente

```
<?xml version="1.0"?>
<name>John</name>
<name>Jane</name>
```

- zusätzliches Eltern-Element einführen:

```
<?xml version="1.0"?>
<names>
  <name>John</name>
  <name>Jane</name>
</names>
```

Regel 4: Namenskonventionen

Element- und Attribut-Namen:

- beginnen entweder mit Buchstaben oder `_` aber nie mit Zahlen:
z.B. `first`, `First` oder `_First`
- nach erstem Zeichen zusätzlich Zahlen sowie `-` und `.` erlaubt:
z.B. `_1st-name` oder `_1st.name`
- enthalten keine Leerzeichen
- enthalten keinen Doppelpunkt
- beginnen nicht mit `xml`, unabhängig davon, ob die einzelnen Buchstaben groß- oder kleingeschrieben sind

Namenskonvention: Beispiele

- `<résumé>` ✓
- `<xml-tag>` nicht korrekt: beginnt mit „xml“
- `<123>` nicht korrekt: beginnt mit Zahl
- `<fun=xml>` nicht korrekt: enthält „=“
erlaubt wären: `_`, `-` und `.`
- `<first name>` nicht korrekt: enthält Leerzeichen

Regel 5: Groß- und Kleinschreibung

XML beachtet Groß- und Kleinschreibung.

- Im Gegensatz zu HTML unterscheidet XML also z.B. zwischen `<P>` und `<p>`.

Dennoch möglichst nicht gleichzeitig
`<First>` und `<first>` verwenden!

Hinweis: eine Schreibweise im gesamten
Dokument verwenden z.B. `<FirstName>`

Regel 6: White Space

XML belässt White Space im Text.

- Beispiel:

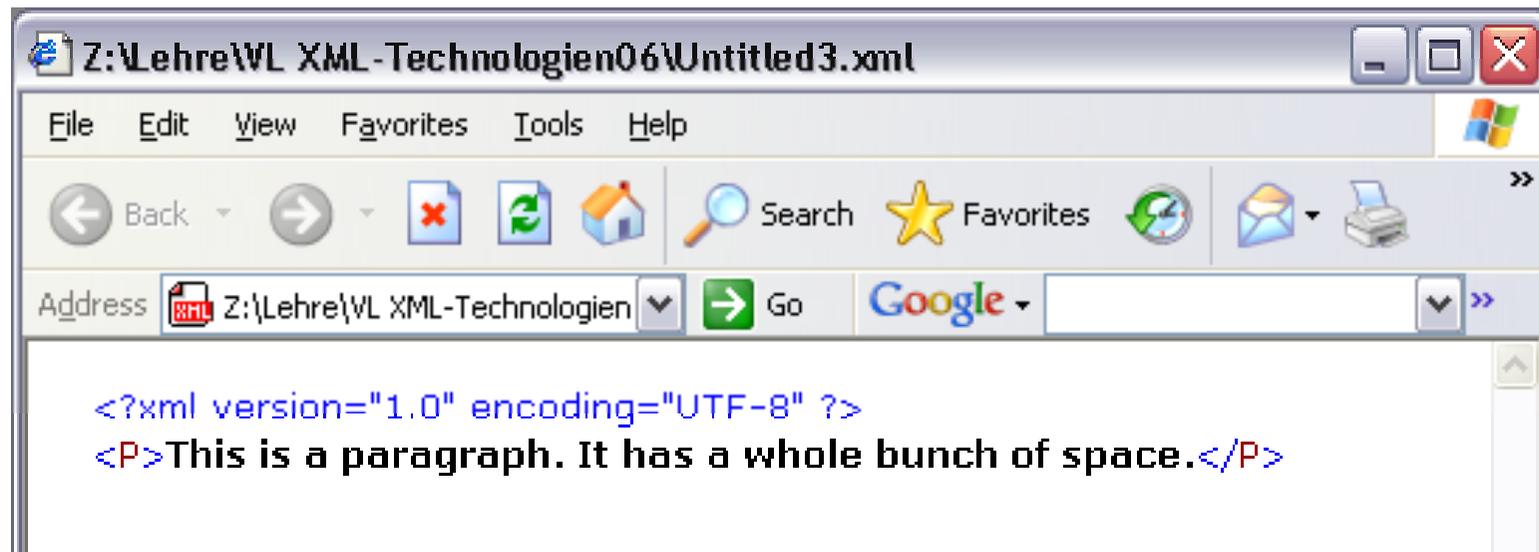
```
<?xml version="1.0" encoding="UTF-8" >  
<P>This is a paragraph.           It has a whole bunch  
  of space.</P>
```

Inhalt des P-Elementes:

```
This is a paragraph.           It has a whole bunch  
  of space.
```

Regel 6: White Space

- Beachte: Von Browsern wird White Space allerdings nicht angezeigt:



- Grund:
 - XML-Dokumente werden zur Darstellung im Browser in HTML umgewandelt
 - HTML reduziert White Space auf ein Leerzeichen

White what?

- Rückblick Folie 33:

- xml:space

- Leerräume im Inhalt

- Beispiel: `<p xml:space="[preserve/default]">Übung 1</p>`

- Rückblick Folie 49:

- XML belässt White Space im Text.

- Warum das Attribut?

default

- This value **allows the application to handle white space as necessary**. Not including an xml:space attribute produces the same result as using the default value.

preserve

- This value instructs the application to **maintain white space as is, suggesting that it might have meaning**.

XML Information Set (Infoset)

- beschreibt das XML-Datenmodell unabhängig von der konkreten XML-Syntax
- allerdings ziemlich umständlich:

“Rather than saying “‘foo’ has a ‘quantity’ of ‘3’” you have to say “the element information item with the [local name] property ‘foo’ has an attribute information item in its [attributes] property with the [local name] property ‘quantity’ and the [normalized value] property ‘3’.” (Nottingham, 2004)

⇒ <http://www.w3.org/TR/xml-infoset/>

- meistbenutzter XML-Editor: XMLSpy von Altova
 - steht in den PC-Pools zur Verfügung
 - Home Edition leider nicht mehr kostenlos verfügbar (bis zur Version 2006 war das so)
 - Enterprise Edition als vierwöchige Testlizenz kostenlos
- ➔ www.xmlspy.com



XML-Editoren

- oXygen
<http://www.oxygenxml.com/>
- Serna – Open Source XML Editor:
<http://www.syntext.com/downloads/serna-free/>
- EditiX Lite:
<http://free.editix.com/>
- Mylin Plug-In in Eclipse (sehr rudimentär)



Namensräume



```
<course>
  <title>Semantic Web</title>
  <lecturers>
    <name>
      <title>Prof. Dr.-Ing.</title>
      <first>Robert</first>
      <last>Tolksdorf</last>
    </name>
  </lecturers>
  <date>12/11/2004</date>
  <abstract>...</abstract>
</course>
```

- **Namenskonflikt:** gleicher Name, aber unterschiedliche Bedeutung
- z.B. Titel einer Veranstaltung vs. Titel einer Person
- in einem Dokument unterschiedliche Vokabularien

Auflösung durch Präfixe

```
<course:course>
  <course:title>Semantic Web</course:title>
  <course:lecturers>
    <pers:name>
      <pers:title>Prof. Dr.-Ing.</pers:title>
      <pers:first>Robert</pers:first>
      <pers:last>Tolksdorf</pers:last>
    </pers:name>
  </course:lecturers>
  <course:date>12/11/2004</course:date>
  <course:abstract>...</course:abstract>
</course:course>
```

- Präfixe geben Kontext an:
Aus welchem Bereich stammt der Name
 - z.B. pers:title vs. course:title
- ähnliches Vorgehen in Programmiersprachen:
 - z.B. java.applet.Applet

{
course:course
course:title course:abstract
course:lecturers
course:date
}

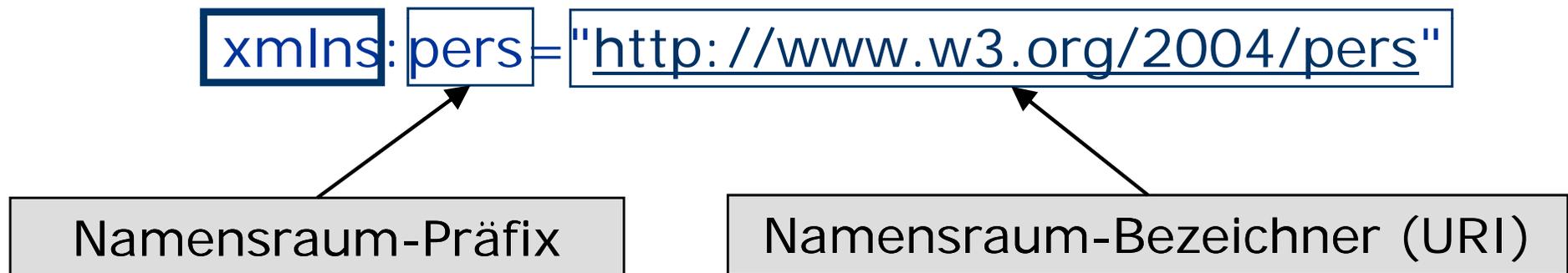
{
pers:name
pers:title pers:first
pers:last
}

Namensraum (namespace):

- alle Bezeichner mit identischen Anwendungskontext
- Namensräume müssen eindeutig identifizierbar sein.

Namensräume in XML

- WWW: Namensräume müssen global eindeutig sein.
- In XML wird Namensraum mit URI identifiziert.
- Zuerst wird Präfix bestimmter Namensraum zugeordnet, z.B.:



- Anschließend kann das Namensraum-Präfix einem Namen vorangestellt werden: z.B. `pers:title`
- Beachte: Wahl des Präfixes egal!

```
<course:course xmlns:course="http://www.w3.org/2004/course">
  <course:title>Semantic Web</course:title>
  <course:lecturers>
    <pers:name xmlns:pers="http://www.w3.org/2004/pers">
      <pers:title>Prof. Dr.-Ing.</pers:title>
      <pers:first>Robert</pers:first>
      <pers:last>Tolksdorf</pers:last>
    </pers:name>
  </course:lecturers>
  <course:date>12/11/2004</course:date>
  <course:abstract>...</course:abstract>
</course:course>
```

Standard-Namensraum

- **xmlns="URI"** statt xmlns:prefix="URI"
- Namensraum-Präfix kann weggelassen werden.
- Standard-Namensraum gilt für das Element, wo er definiert ist.
- Kind-Elemente erben Standard-Namensraum von ihrem Eltern-Element.
- Ausnahme: Standard-Namensraum wird überschrieben
- Beachte: Standardnamensräume gelten nicht für Attribute

```
<course:course xmlns:course="http://www.w3.org/2004/course">
  <course:title>Semantic Web</course:title>
  <course:lecturers>
    <pers:name xmlns:pers="http://www.w3.org/2004/pers">
      <pers:title>Prof. Dr.-Ing.</pers:title>
      <pers:first>Robert</pers:first>
      <pers:last>Tolksdorf</pers:last>
    </pers:name>
  </course:lecturers>
  <course:date>12/11/2004</course:date>
  <course:abstract>...</course:abstract>
</course:course>
```

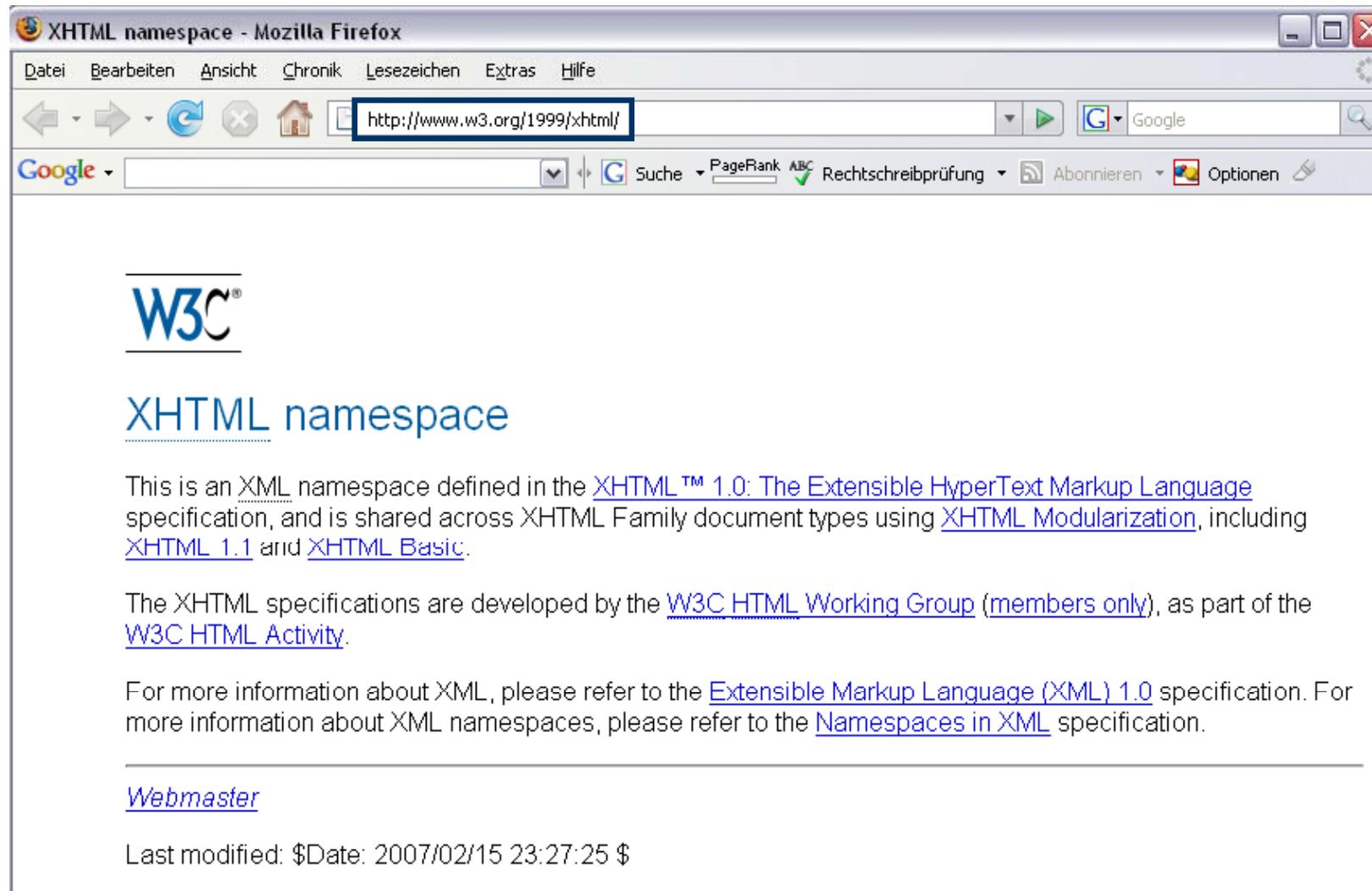
```
<course xmlns="http://www.w3.org/2004/course" >
  <title>Semantic Web</title>
  <lecturers>
    <pers:name xmlns:pers="http://www.w3.org/2004/pers" >
      <pers:title>Prof. Dr.-Ing.</pers:title>
      <pers:first>Robert</pers:first>
      <pers:last>Tolksdorf</pers:last>
    </pers:name>
  </lecturers>
  <date>12/11/2004</date>
  <abstract>...</abstract>
</course>
```

```
<course xmlns="http://www.w3.org/2004/course">
  <title>Semantic Web</title>
  <lecturers>
    <name xmlns:pers="http://www.w3.org/2004/pers">
      <title>Prof. Dr.-Ing.</title>
      <first>Robert</first>
      <last>Tolksdorf</last>
    </name>
  </lecturers>
  <date>12/11/2004</date>
  <abstract>...</abstract>
</course>
```

Standard-Namensräume: Gültigkeit

```
<course xmlns="http://www.w3.org/2004/course" >
  <title>Semantic Web</title>
  <lecturers>
    <name xmlns="http://www.w3.org/2004/pers" >
      <title>Prof. Dr.-Ing.</title>
      <first>Robert</first>
      <last>Tolksdorf</last>
    </name>
  </lecturers>
  <date>12/11/2004</date>
  <abstract>...</abstract>
</course>
```

- Beispiel: <http://www.w3.org/1999/xhtml> bezeichnet den Namensraum für XHTML



http://blog.ag-nbi.de/category/news/



Protokoll



Resource Identifier

Über das angegebene **Protokoll** ist eine **Interaktion** mit einer **Repräsentation** der identifizierten **Informationsressource** über ein Netzwerk möglich.

urn:isbn:4-123-23545-6



protokollunabhängig

Identifizier



- URI kann (muss aber nicht) Beschreibung des Namensraumes enthalten
- URI muss nicht einmal existieren
- nur bei existierenden URIs Eindeutigkeit sichergestellt

```
<Book xmlns="http://www.books-ns.org" >  
  ...  
</Book >
```

- <http://www.book-ns.org> muss nicht existieren
 - keine Fehlermeldung, keine Warnung von XML-Parser oder XML-Editor
 - dennoch Eindeutigkeit nicht sichergestellt:
jemand anderes kann gleiche URL für anderen Namensraum verwenden
- ⇒ neue Namensräume nur mit URIs bezeichnen, die man selbst besitzt

Qualified vs. Unqualified

- Element- oder Attribut-Name heißt namensraumeingeschränkt (qualified), wenn er einem Namensraum zugeordnet ist.
- Einschränkung vom Element-Namensraum:
 1. Standard-Namensraum festlegen
 2. Namensraum-Präfix voranstellen
- Einschränkung vom Attribut-Namensraum:
 1. Namensraum-Präfix voranstellen

Was bedeutet `<p>...</p>`?

- **HTML:**
 - Bedeutung festgelegt (p = Absatz)
- **XML:**
 - Bedeutung offen
 - kann aber mit **Namensraum** festgelegt werden
 - Beispiel: p stammt aus dem Namensraum für XHTML.

xhtml Abkürzung für Namensraum

`<xhtml:p xmlns:xhtml="http://www.w3.org/1999/xhtml">...</xhtml:p>`

Namensraum:
u.a. p = Absatz

Und das war es schon!

- Syntax wohlgeformter XML-Dokumente vorgestellt
- XML-Syntax sehr einfach
- XML beliebig erweiterbar

Wie geht es weiter?

- ☑ XML-Syntax
- ☑ Namensräume

- Definition von XML-Sprachen mit DTDs und XML-Schema