



<http://www.therealgang.de/>

Titel :	Einführung in XML (Teil 1/2)
Author :	Dr. Pascal Rheinert
Kategorie :	Programmierung-Sonstige

Teil II: XML

Wichtiger Hinweis: Das folgende Dokument basiert auf dem HTML-Lehrgang „Selfhtml“ in der Version 8.0 von Stefan Münz, auf den an sehr vielen Stellen verwiesen wird.

I. Vorbemerkungen (teilweise in intro\technologien\xml.htm)

- XML = eXtended Markup Language
- Wurde – wie HTML – von SGML abgeleitet.
- Erster Entwurf zu XML: 1996
- Februar 1998: XML als Empfehlung des W3-Konsortiums
- Das Besondere an XML: Erlaubt Definition von standardisierten Dokumenten
 - Abkehr von proprietären Dateiformaten (Word6.0 Format, Word97, Word2000, Exel 4.0, Excel 5.0)
 - XML definiert Schnittstellen zum Austausch von Daten
 - Datenaustausch zwischen beliebigen Applikationen wird möglich
 - Voraussetzung für fairen Wettbewerb im Software-Markt
- Im Unterschied zu HTML können neue Elemente (tags) definiert werden
 - XML ist lediglich ein Schema, keine Ausprägung (wie HTML)
 - Eigene Ausprägungen („Sprachen“) von XML müssen (individuell) definiert werden
 - Art der Darstellung (im Browser) ist daher nicht definiert
- Darstellung der Inhalte von XML-Dateien im Browser ist über StyleSheet-Definitionen möglich (XSL - Extensible Stylesheet Language)
- XSLT „übersetzt“ XML in andere Auszeichnungssprachen (z.B. HTML)
- Von XML abgeleitete Sprachen werden als „XML-Derivate“ bezeichnet (Beispiel: OpenTrans, BME-cat)

II. Einführung in XML (xml/intro.htm)

- Aufgabe von XML: Bereitstellung eines Regelwerks zur Definition eigener Sprachen.
- Beispiele für mögliche XML-Derivate (siehe „selfhtml“)
Interessant: Völlig unterschiedliche Einsatzgebiete; „eigene Sprachkonstrukte“
Wichtig: Macht relativ wenig Sinn, wenn außer dem „Erfinder“ es keiner versteht
„Erfinder“: Definiert Elemente, Attribute und mögliche Werte.
- Definition der o.g. Regeln erfolgt in eigenen Datei(en) → DTD (Document Type Definition)
→ Trennung von Inhalten und Regeln
Vorteile:
Regeln können für beliebig viele Dokumente verwendet werden;
Regelwerk kann (und sollte) von vielen Partnern verwendet werden;
Möglichkeit der Definition von Standards
Nachteil (Gefahr):
Jeder muss die Regeln genau kennen, sonst: Mißverständnis bzw. Fehler
→ Validierung der XML-Daten erforderlich
- Beispiel für DTD und einer darauf aufbauenden XML-Datei: in selfhtml
- DTD definieren Elemente (und Unterelemente), Attribute sowie zulässige Werte, nicht jedoch die Darstellung im Browser
- Hierzu dienen XSL und CSS
- XSL: Von Syntax her an XML angelehnt und umfangreicher als CSS (z.B. bedingte Formatierung)
- Beispiele von Formatierungen mit XSL bzw. CSS: in selfhtml
- XML-Parser: Analysiert die XML-Datei (validierend (über die entsprechende DTD-Datei) oder nicht-validierend) und gibt die Daten an die dahinterliegende Applikation weiter.
- Web-Browser sind erst seit neuerem in der Lage, XML-Dateien zu parsen und (über eine XSL- oder CSS-Datei) entsprechend darzustellen (Netscape ab 6.0, IE ab 5.0)
→ Client-Seitig gibt es heute noch einige Probleme bei Anwendung von XML als HTML-vergleichbare Web-Sprache
Abhilfe:
XSLT (XSL-Transformation): Können XML-Dateien serverseitig in HTML übersetzen
→ der Browser beim Client hat keine Probleme
- XML-Standardsprachen:
 - SVG (Scalable Vector Graphics) → Beispiel: in selfhtml
 - MathML (Mathematical Markup Language) → Beispiel: in selfhtml
 - ... → in selfhtml

III. Regeln für XML-Dateien (xml\regeln\)

- **XML-Deklaration und Verarbeitungsanweisungen (xmldeklaration.htm):**

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>
<!-- Daten -->
<?xml-stylesheet type="text/css" href="styles.css"?>
```

- Version, encoding, standalone (befindet sich DTD innerhalb der Datei?)
- Zusätzlich: Verarbeitungsanweisungen

- **Dokumenttyp-Deklaration (dokumenttypdeklaration.htm):**

- Stellt Bezug her zwischen vorliegender XML-Datei und DTD-Datei

```
<?xml version="1.0"?>
<!DOCTYPE Gruss [
  <!ELEMENT Gruss (#PCDATA)>
]>
<Gruss>Hallo Jupiter!</Gruss>
```

- oben: interne Deklaration
- Externe Deklaration: <!DOCTYPE <Typ> SYSTEM „http:....“>
- Endung der DTD-Dateien: „.dtd“ (empfohlen)

- **Regeln für Tags, Attribute, Wertzuweisungen und Kommentare (tagsattrwerte.htm):**

- **WICHTIG:** XML unterscheidet zwischen Groß- und Kleinschreibung!!
→ Wenn in DTD „links“ deklariert wurde, darf in XML hierfür nicht „Links“ stehen!
- Wertzuweisungen stehen in Anführungszeichen.
- Das Anführungszeichen darf nur als " das Apostroph als ' auftauchen.
- Tags stehen immer paarweise (<bereich> </bereich>)
- Ausnahme: leere Tags (<bereich />) oder (<bereich></bereich>)
- Selbst definierte Elementnamen dürfen nicht mit „xml“ beginnen und keine Leerzeichen, bzw „=“ enthalten
- Kommentare wie in HTML (<!-- Kommentar, der hier endet -->)

- **Wohlgeformtheit, Gültigkeit und Vollständigkeit einer XML-Datei (begriffe.htm):**

- Wohlgeformtheit: XML-Datei hält Regeln von XML korrekt ein. (Beispiel: in selfhtml)
 - XML-Syntax wird eingehalten, ob der Inhalt aber verstanden bzw. einer DTD-Datei entspricht ist nicht sicher.
- Gültigkeit und Vollständigkeit:

```
<?xml version="1.0"?>
<!DOCTYPE quelle [
  <!ELEMENT quelle (adresse, beschreibung)>
  <!ELEMENT adresse (#PCDATA)>
  <!ELEMENT beschreibung (#PCDATA)>
]>
<quelle>
  <adresse>http://www.villeroy-boch.de/</adresse>
  <beschreibung>Alles Produkte dieses Unternehmens</beschreibung>
</quelle>
```

- **Baumstruktur und Knoten einer XML-Datei (baumstruktur.htm):**

- In XML-Datei gibt es Elemente, Attribute mit Wertzuweisungen, sowie dem Inhalt der Elemente, die wiederum aus Elementen bestehen können oder aus Text.
- Aufbau einer XML-Datei vergleichbar mit den Ästen und Blättern eines Baumes.
- XML-Datei kann in baumartiger Struktur dargestellt werden.
- Beispiel: in selfhtml
- Knoten = Bestandteile des Baumes; Knoten-Sets = Beschreibung des Wegs (Auflistung von Knoten (Ästen)) meist vom Wurzelknoten bis zum betrachteten Element.

- **() XML-Namensräume (namensraeume.htm):**

- Verweis auf Namensräume (etwa Unterverzeichnisse einer URL-Adresse)
- Ansatz zur Lösung der Problematik „Ist ein wohlgeformtes Dokument auch gültig“.
- Wichtig v.a. bei Mehrfachbezeichnungen (Nicht-Eindeutigkeit) von Elementen in *einem* XML-Dokument.

- **Zeichen, Zeichensätze und nicht interpretierte Abschnitte (zeichen.htm):**

- Die Zeichen „<“, „>“, „=“, „““, „’“, „&“ müssen „umschrieben“ werden (Grund: Zeichen haben spezielle Aufgaben in XML – wie auch in HTML)
- Nicht-interpretierte Abschnitte: Werden von dem Parser nicht analysiert (z.B. <![CDATA[<Element>dieses Element wird nur als Zeichenfolge ausgegeben</Element>]]>).
- Namenskonventionen: Endung „.xml“, „.dtd“

IV. Dokumenttyp-Definitionen (xml\dtd\)

- **Allgemeines zu DTDs (allgemeines.htm):**
 - Aufgabe der DTD: Vorlage zur Validierung der xml-Dateien durch Parser
 - Ist nicht in jedem Fall erforderlich (nicht jede XML-Datei muss „gültig“ sein).
 - **Eigene oder vorhandene DTD verwenden?**
 - Nach Möglichkeit keinen eigenen „Standard“ über eigene DTD definieren.
 - XML als „Datenaustausch-Sprache“ macht (nur) Sinn, wenn sich möglichst viele Partner auf einen Standard einigen.
 - In dieser Hinsicht gibt es viele Bestrebungen (OpenTrans zum Austausch von Geschäftsdokumenten, BME-cat als Standard für Produkt-Kataloge).
 - → Informieren vor (Neu-)Definieren.
 - DTD können modular aufgebaut werden (vgl. mit Vererbung in Objekt-Orientieren Programmiersprachen)
 - **Datei-interne oder DTD in separater Datei verwenden?**
 - Separat ist (fast) immer besser.
 - **Namenskonzept für eine DTD**
 - Sinnvoll: nur Kleinbuchstaben, englische Begriffe, aussagekräftige Begriffe, kurze Begriffe (ähnliche Problematik wie bei einer Programmiersprache; zu bedenken hierbei aber: falls große Datenmengen → lange Begriffe führen zu enormen Dateigrößen)
 - **Datenanalyse als Vorbereitung zum Entwurf einer DTD**
 - Vergleich mit der Entwicklung eines Programms.
→ zu Beginn: Exakte Anforderungs-Analyse.
 - Wichtig!!! → siehe selfhtml
- **Bearbeitungsregeln für DTDs (bearbeitungsregeln.htm):**
 - XML- und DTD-Dokumente sind menschenlesbar und daher mit beliebigem Editor editierbar.
 - Es gibt spezielle Editoren, die den Erstellungsprozess vereinfachen.
 - Beispiele:
 - `<!ELEMENT text (#PCDATA) >`
 - `<!ELEMENT nummer (#PCDATA) >`
 - Regeln für Namen (keine Ziffer am Anfang, etc.) → selfhtml
 - Kommentare (wie in HTML)
 - Bedingte Abschnitte häufig zu Testzwecken (`<![IGNORE[<!ELEMENT)`)

- **Elemente und Verschachtelungsregeln (elemente.htm):**

- Definition von Elementtypen über `<!ELEMENT name (inhalt)>`.
- Diese Elementtypen werden in der xml-Datei als Vorlage für Elemente verwendet.
- (Vergleichbar mit OOP: Objekt – Instanz eines Objekts).
- Beispiel: Durchwahlnummer in selfhtml.
- Elementtypen mit Elementinhalt (verschachtelte Elemente) definieren.
- Anzahl der Wiederholungen eines (Unter-)elements definieren:
 - `<!ELEMENT test (genau-einmal)>`
 - `<!ELEMENT test (mindestens-einmal)+>`
 - `<!ELEMENT test (beliebig)*>`
- Beispiel: Kochrezept in selfhtml.
- **Alternative und optionale Elementtypen für Elementinhalt definieren**
 - `<!ELEMENT test (optional?, (alternative1 | alternative2), muss1, muss2)!>`
- **Elemente mit gemischtem Inhalt definieren**
 - `<!ELEMENT text (#PCDATA | drohend | lachend | fragend | zynisch)*>`
- **Elemente mit beliebigem Inhalt definieren**
 - `<!ELEMENT anytext ANY>`
- **Elemente mit leerem Inhalt definieren**
 - `<!ELEMENT void EMPTY>` <-- z.B. `
` in HTML -->
- Parameter Entity (entsprechen Variablen bzw. Makros)
- **Zusammenhängendes Beispiel: ein Buch → selfhtml**

- **Attribute und Wertzuweisungen (attribute.htm):**

- Attribute dienen der Spezifikation von Elementen (vergleichbar mit HTML: `<p align=justify>sdfsdf</p>`)
- Attribute sind v.a dann sinnvoll, wenn die Wertzuweisung sich auf die Auswahl aus einer definierten Menge beschränkt.
- Häufig sind Spezifikationen auch über Unter-Elemente lösbar (Beispiel in selfhtml).
- Schema zum Definieren von Attributen:

```
<!ELEMENT Elementname (Inhalt)>
<!ATTLIST Elementname
  Attributname_1 Inhalt [#REQUIRED|#IMPLIED|#FIXED
  "Wert"|Defaultwert]
  Attributname_n Inhalt [#REQUIRED|#IMPLIED|#FIXED
  "Wert"|Defaultwert]
>
```

- **REQUIRED:** Muss-Attribut, **IMPLIED:** Kann-Attribut **FIXED:** Nur eine Zuweisung möglich.

- DTD für ein Auto:

```
<!ELEMENT autos (auto)*>
<!ELEMENT auto EMPTY>
<!ATTLIST auto
  typ CDATA #REQUIRED
  bj CDATA #REQUIRED
  km CDATA #REQUIRED
  ps CDATA #REQUIRED
  vb CDATA #REQUIRED
>
```

- XML-Datei für ein konkretes Auto:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE autos SYSTEM "autos.dtd">
<autos>
<auto typ="AUDI 80" bj="1992" km="125000" ps="90" vb="12500 DM" />
</autos>
```

- Attribute mit festen alternativen Werten:

```
<!ELEMENT hotels (hotel)*>
<!ELEMENT hotel (#PCDATA)>
<!ATTLIST hotel
  name CDATA #REQUIRED
  klasse (I|II|III|IV|V) #REQUIRED
  einzelzimmer (ja|nein) #IMPLIED
  doppelzimmer (ja|nein) "ja"
>
```

- #FIXED: Eindeutige Zuweisung ((sprache (de|en|it) #FIXED „en“)): Mehrsprachigkeit ist zwar vorgesehen, bisher gibt es aber nur englisch!
- Attribute mit Bezeichnerwerten:
- Attribute mit ID-Wert: **ID**
 - <!ATTLIST test wert ID #REQUIRED ...>
 - Wertzuweisung zu wert muss eindeutig sein (häufige Anforderung von Datenbanken), wird vom Parser entsprechend überprüft. Zusätzlich muss Wert den Regeln für Namen entsprechen (ID als Zahl ist nicht zulässig)
- Attribute mit ID-Referenzwert: **IDREF**
 - <!ATTLIST test referenz IDREF #IMPLIED ...>
 - Verweis auf ein Element mit dem eindeutigen (ID-) Wert referenz.
 - Ermöglicht Aufbau von baumartigen Strukturen.
 - Beispiel in selfhtml (attribute.htm#mit_id_referenzwert)
- Attribute mit Entity-Wert (z.B. für externe Dateireferenzen): **ENTITY**
 - <!ATTLIST test quelle ENTITY #REQUIRED ...>
 - Nicht Zeichenfolge des zugewiesenen Werts interessiert, sondern die damit verbundene Datei (= Verweis auf eine vorher definierte Entity).

- Attribute mit Namenwert: **NMTOKEN**
 - `<!ATTLIST test quelle NMTOKEN #REQUIRED ...>`
 - Entspricht einer ID, die nicht eindeutig sein muss. Zugewiesene Wert darf mit Ziffer beginnen.
- Beispiel: Personenbeschreibung → Selfhtml
- **Entities für Textbausteine und Umschreibungen (entities.htm):**
 - Entities stehen für Makros bzw. Konstanten, die die Verwendung von DTDs flexibler gestalten.
 - Schema zur Definition von Entities:

```
<!ENTITY [%] Name [SYSTEM|PUBLIC] "Wert" [zusätzliche Angaben] >
```

- Wichtig: Definition der Entities muss vor deren ersten Verwendung erfolgen
→ Sinnvoll: Zu Beginn einer DTD-Datei.
- Entity für Textbaustein (baustein.dtd):

```
<!ELEMENT baustein (#PCDATA)>  
<!ENTITY mfg "mit freundlichen Gr&#252;&#223;en" >
```

- Verwendung dieses Textbausteins (baustein.xml):

```
<?xml version="1.0" encoding="ISO-8859-1" ?>  
<!DOCTYPE baustein SYSTEM "baustein.dtd">  
<baustein>  
ich verbleibe &mfg;  
</baustein>
```

- Entities für die Benennung von Zeichen → selfhtml
- Verschachtelte Entities → selfhtml
- Entities für externe Ressourcen

```
<!ELEMENT news (newsdaten)*>  
<!ENTITY datenquelle SYSTEM "news.txt" >  
<!ELEMENT newsdaten EMPTY>  
<!ATTLIST newsdaten  
  quelle          ENTITY          #REQUIRED  
>  
-----  
<?xml version="1.0" encoding="ISO-8859-1" ?>  
<!DOCTYPE news SYSTEM "news.dtd">  
<news>  
<newsdaten quelle="datenquelle" />  
</news>
```

- Die Zuweisung: `quelle = „datenquelle“` setzt nicht die Zeichenfolge ein, sondern die vorher definierte Datei „news.txt“.

- **Parameter Entities für komplexere DTDs**
 - In der Definition von Parameter-Entities steht „%“
 - Sinnvoll, um die Menge der möglichen Werte an einer Stelle ändern / erweitern zu können.
 - Beispiel: `<!ENTITY % artikel "artikelnummer, artikelname, artikelmenge">`
 - Beispiel: `<!ENTITY % zusatz "beschreibung | warenkategorie">`
 - Verwendung der so definierten Parameter-Entities: `(%artikel;)`
 - → siehe selfhtml
- **Modulare DTDs mit Hilfe von Entities**
 - Aufbau von modularen DTDs bei großen Projekten sinnvoll
 - logisch zusammengehörige Definitionen in eine dtd-Datei
 - Trennung in mehrere dtd-Dateien (vgl. mit Programmierung in C)
 - Einbinden von solchen modularen DTD über:
`<!ENTITY % beispiel SYSTEM "beispiel.dtd" >`
`%beispiel;`
 - `%beispiel;` fügt den Inhalt der Datei „beispiel.dtd“ an die entsprechende Stelle der einbindenden dtd-Datei ein.
 - Eine XML-Datei muss nur die übergeordnete DTD-Datei einbinden.
- **Notations für referenzierte Daten (notations.htm)**
 - Notations sind Verarbeitungshinweise für externe Dateien.
 - Schema zur Definition von Notations:

```
<!NOTATION Datentyp [SYSTEM|PUBLIC] "Verarbeitungshinweis" >
```

- SYSTEM: Hinweis (Pfad zu einer lokal installierten Anwendung)
- PUBLIC: public identifier