In diesem Kapitel

- Warum XML?
  - Daten und Strukturen
  - Robustheit
  - Erweiterbarkeit
  - Einfachheit
- XML Syntax
  - XML Dokumente
  - XML Applikationen
  - Elemente und Tags
  - Text
  - Attribute
  - XML Deklarationen
  - Kommentare
  - Verarbeitungs-Instruktionen
  - Entities
  - Namensräume
- Gültigkeit
  - DTD – Document Type Definition
  - Schemas
  - Schematron
- Stylesheets
  - CSS – Cascading Style Sheets
  - Assoziation von CSS und XML Dokumenten
- Zusammenfassung

1. *XML Grundlagen*

XML hat seine Wurzeln, ähnlich wie HTML, in SGML. Aber XML sollte einige der Probleme von HTML eliminieren, die eine automatische Bearbeitung von HTML Seiten erschwerten. HTML ist zu flexibel, zu locker definiert! Zielpublikum waren also primär Autoren von technischen Texten.

Die Realität sieht viel rosiger aus:

- XML hat sich zu einer universellen Basis für viele Informatik-Gebiete entwickelt
- XML basierte Metasprachen werden heute in fast allen kommerziellen Applikationen eingesetzt.
- XML wird zur Spezifikation von GUI's und Dokumentinhalten oder Datenbankinhalten und Strukturen eingesetzt.
- (Text-) Dokumente lassen sich als Objekte in XML serialsieren und problemlos übermitteln.
- XML ist universeller als jede Java Applikation.

1.1. Warum XML?

Sie kennen das Problem: jedes Programm hat sein optimales Datenformat. Excel, Word, PostScript, LaTex, PDF, Text, CSV... und oft sollten Sie Daten aus einem Format in Daten in einem anderen Format transformieren.

Ein Ausweg, robust und zuverlässig, war und ist die Verwendung reiner Text-Darstellungen für alle externen Daten. XML ist in diesem Sinne auch eine Alternative zum CSV Format.

1.1.1. Daten und Strukturen

Betrachten wir eine einfache Bestellung eines Buches über das Internet. Wir könnten einfach alle Daten als Text übermitteln. Das könnte etwa folgendermassen aussehen:

```
L101023
Josef M. Joller
Sonnenbergstrasse 73
CH-8610 USTER
TK5105.888 .N482 2002
Understanding Web Services
XML, WSDL, SOAP and UDDI
Eric Newcomer
Addison-Wesley
0-201-75081-3
EUR
45.95
1
EUR
45.95
%
7
EUR
3.20
EUR
49.15
UPS
```

Das Ganze sieht nicht gerade lesbar aus. Einfacher wäre eine Darstellung mit Hilfe eines XML Dokuments:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Bestellung>
        <Kunde KundenNummer="L101023">Josef M. Joller</Kunde>
        <Kundenadresse>Sonnenbergstrasse 73</Kundenadresse>
        <Kundenort PLTZ="CH-8610">USTER</Kundenort>
        <Artikel Artikelnummer=" TK5105.888 .N482 2002" ISBN="0-201-75081-3">Understanding Web Services –
                XML, WSDL, SOAP and UDDI</Artikel>
        <Autor>Eric Newcomer</Autor>
        <Verlag>Addison-Wesley</Verlag>
        <Preis Währung="EUR">45.95</Preis>
        <Anzahl>1</Anzahl>
        <Mehrwertsteuer Einheit="%">7</Mehrwertsteuer>
        <Mehrwertsteuer-Aufschlag Währung="EUR">3.20</Mehrwertsteuer-Aufschlag>
        <Total Währung="EUR">49.15</Total>
        <Lieferung>UPS</Lieferung>
</Bestellung>
```

Obschon wir uns über den Aufbau noch unterhalten könnten, dürfte diese Darstellung lesbarer sein als die erste. Beide Darstellungen enthalten die selbe Information.

1.1.2. Robustheit

Falls Sie in der ersten, der reinen Textdarstellung, zwei Zahlen vertauschen, wird es Ihnen nicht so einfach gelingen, den Fehler zu finden. Geschieht Ihnen der selbe Fehler mit XML, dürfte es kaum sehr lange dauern, bis Sie den Fehler gefunden und korrigiert haben.

Die XML Darstellung ist im gewissen Sinne „robuster“ als die reine Textdarstellung.
Sie könnten zudem mithilfe von DTD und / oder Schema bestimmte Beziehungen spezifizieren, die unter den einzelnen XML Angaben bestehen müssen; oder wir könnten bestimmte erlaubte Werte angeben (Währungsabkürzungen, Syntax der ISBN Nummer und vieles mehr). Damit haben wir auch eine Möglichkeit die Daten in bestimmten Grenzen zu validieren.

1.1.3. Erweiterbarkeit

Neben der Robustheit ist die Erweiterbarkeit ein weiterer Vorteil von XML. Wenn wir beispielsweise noch unseren Firmen- Discount erfassen möchten, können wir in der Textdarstellung zwei Zeilen (Prozent und Wert), oder in XML lesbaren Code hinzufügen:

```
<Rabatt Prozent="5“ Währung="EUR“>1.25<Rabatt>
```

Die Interpretation dieses Zusatzes mit einem Analyseprogramm wird kaum Probleme geben. Sollten wir aber in der Textdarstellung die Position falsch wählen, kann die neue Darstellung zu einem Absturz führen, weil eventuell Textdaten gelesen werden, wo numerische Daten erwarten werden.

Unter Umständen möchten Sie bereits formatierten Text einfügen. Hier ein Beispiel:

```
<Gruss>
        Alles Gute zum Geburtstag

        und viel Spass beim Lesen
</Gruss>
```

Die Erweiterung der obigen XML Darstellung auf eine Bestellung mit mehreren Büchern oder Artikeln, oder auch auf mehrere Kunden, dürfte Ihnen kaum Schwierigkeiten machen. Zu beachten ist lediglich, dass ein XML Dokument lediglich ein Wurzelelement besitzen darf. XML Dokumente sind immer in Form eines Baumes darstellbar.

1.1.4. Einfachheit

Das Lesen (und Schreiben) eines XML Dokuments wird wesentlich einfacher als das Lesen (und Schreiben) einer normalen Textdatei, weil es spezielle Parser gibt, die XML Tags erkennen und das Dokument entsprechend zerlegen können.

Parser können auch gleichzeitig überprüfen, ob das XML Dokument gültig oder wohlgeformt ist. Sie können mit einem Parser gezielt auf bestimmte Tags zugreifen und die darin enthaltene Information, sei es nun in Form von Attributen oder als Daten, herausholen.

1.1.5. Bemerkung

Eines der Ziele in der ursprünglichen Definition von XML war es, dass es einfach sein sollte, XML Dokumente zu lesen oder zu schreiben. Gedacht hatte man zuerst an Perl als damalige Standard-Web-Sprache (CGI-Skripts). In der Zwischenzeit stehen leistungsfähige Parser für fast alle Programmiersprachen zur Verfügung.

Der Parser sorgt sich um viele Details, er nimmt Ihnen also sehr viel Arbeit ab:

- Sie brauchen sich nicht darum zu kümmern, wie die Daten kodiert sind (ASCII, Unicode, ...).
- CR, LF und weitere Steuerzeichen spielen keine Rolle (diese werden in Unix und Windows Systemen unterschiedlich behandelt).
- Tags werden für Sie interpretiert (<Tagname>...</Tagname>).
- LSB, MSB ... unterschiedliche interne Darstellungen und damit verbundene Umwandlungen brauchen Sie nicht mehr zu belasten.

1.2. XML Syntax

Im Folgenden wollen wir die benötigten Definitionen zu XML kurz wiederholen, ohne in alle Details zu gehen.

1.2.1. XML Dokumente

Die exakte Bedeutung von XML wurde in einer W3C Spezifikation festgelegt, in Form einer BNF Notation. Neben der Syntax wurden 15 Regel für die "Wohlgeformtheit" festgelegt. Wohlgeformt ist in der Regel nicht ausreichend : es legt lediglich fest, dass Tags korrekt abgeschlossen werden , ob das Dokument lediglich ein Root besitzt, Attribute müssen in Anführungszeichen stehen, nur Unicode Zeichen sind erlaubt u.ä. . . Ein Parser kann mit einem wohlgeformten Dokument noch nicht sehr viel anfangen, Fehler können nicht lokalisiert oder korrigiert werden. Nach dem ersten Fehler bricht der Parser den Analysevorgang ab.

1.2.2. Bemerkung

Ein XML Parser benötigt ein vollständiges XML-Dokument. Es ist nicht möglich Dokumente zu parsen, welche unvollständige XML Dokumente sind. Falls man unvollständige XML-Dokumente bearbeiten möchte, dann muss ein Präprozessor geschrieben werden.

Jedes XML- Dokument kann man auch als Zeichenkette auffassen., welche bestimmten Regeln gehorchen muss.

Ein XML- Dokument ist auch ein Baum. Es besitzt genau eine Wurzel und verschiedene Sohnknoten, welche selber wieder weitere Subknoten besitzen können. Endknoten bezeichnet man als Blätter.

Ein XML Dokument kann grob gesprochen folgende fünf Knoten besitzen:

- *Root* – Wurzel : auch *Document Node* genannt.
  Dieser abstrakte Knoten repräsentiert das gesamte XML Dokument. Die Wurzel kann unterschiedliche Kinderknotentypen haben: Kommentare, Verarbeitungsanweisungen (processing instructions, PI's)
- *Element* – XML Elemente
  mit Namen, Attributliste, Namensraumliste, Kinderliste.
- *Text*
  geparste Zeichendaten zwischen zwei Tags
- *Kommentar*
  Der Inhalt eines Kommentars entspricht den Daten eines Tags. `<!-- Hallo -->`
  Kommentare können keine Kinderknoten haben.
- *Verarbeitungsanweisung* – Processing Instruction (PI)
  Eine PI besteht aus einem Target und einem Wert. PI's besitzen keine Kinderknoten.
  Beispiel: `<?xml-stylesheet type="text/css" href="antrag.css"?>`

Je nach Einsatzart können weitere Knoten definiert werden (CDATA als Beispiel). Auch Attribute und Namensräume werden zum Teil als Knoten betrachtet, sind aber eigentlich eher Teil der obigen Elemente. Innerhalb von XPath sieht die Situation aber anders aus!

1.2.3. Hinweis

Neben der Definition von XML Dokumenten definierte W3C auch noch XML Information Sets (Infosets) und das Document Model (DOM). Diese definieren ein abstraktes Modell. SAX definiert eine andere Art der Dokumentverarbeitung, bei dem ein Dokument im wesentlichen als Folge von Zeichenketten aufgefasst wird.

1.2.4. XML Anwendungen

Eine *XML Applikation* spezifiziert ein spezielles XML Vokabular, also spezielle Definitionen für spezielle Elemente und Attribute. Im Wesentlichen ist also eine XML Applikation ein konkretes Beispiel für die abstrakte XML Syntax.

Beispiel: DocBook ist eine XML Applikation
Typische Elemente einer DocBook Anwendung sind:

- `book`
- `chapter`
- `sect1`
- `...`

XML Applikationen können *Schemas* einsetzen, um definieren zu können, ob ein XML Dokument gültig oder lediglich wohlgeformt ist. Die Struktur kann mithilfe von DTD's (Document Type Definitions), W3C XML Schema, RELAX NG, Schematron, ... spezifiziert werden.

Ein konkretes Dokument wird als Instanz einer XML Applikation bezeichnet.

1.2.5. Elemente und Tags

Die Grundeinheit von XML ist das *Element*. Ein XML Dokument muss mindestens ein Element enthalten.

Elemente bestehen aus vier Bestandteilen:

- Einem Namen.
- Den Attributen des Elements.
- Dem Namensraum aus dem das Element stammt.
- Dem Inhalt des Elements.
- Dem Elementtyp (aus dem dazugehörenden XML Schema)
- Dem Wert des Elements

Der Wert eines Elements wird je nach zugrunde liegendem Modell (DOM,...) unterschiedlich definiert. In jedem Fall lässt sich der Wert eines Elements aber aus seinem Inhalt ableiten.

Syntax:

- Start-Tag	:	<*Tag-Name*>	<Name>
- Inhalt des Elements	:	*Inhalt*	Heinz
- End-Tag	:	</*Tag-Name*>	</Name>

Beispiel :
`<Quantity>12</Quantity>`
Name des Tags: Quantity; Attribute des Elements : keine; Namensraum : nicht angegeben; Inhalt : `12;` Elementtyp : nicht angegeben; Wert des Elements : 12.

Analogie: ein Element ist ein Hamburger; das Brot sind die Tags; der Inhalt sollte essbar sein.

Spezialfall:
Falls ein Element keinen Inhalt besitzt, wird es als *leeres Element* bezeichnet, und eine verkürzte Schreibweise ist erlaubt:
Syntax für leere Elemente: `<Tag-Name/>` (Kurzform für `<Tag-Name></Tag-Name>`

Ein Element kann auch ein oder mehrere Kind-Elemente besitzen. Diese sind vollständig innerhalb des Start- End-Tags des übergeordneten Elements enthalten, aber nicht in einem andern Tag.

Beispiel:
`ShipTo` Element besitzt vier Kinder: `Street, City, State` und `Zip`:

```
<ShipTo>
  <Street>Oberseestrasse 10</Street >
  <City>Rapperswil</City>
  <State>SG</State>
  <Zip>8640</Zip>
</ShipTo>
```

Neben den Tags und deren Inhalt besitzt das `ShipTo` Element noch Leerzeichen, allgemeiner Whitespaces. Da diese Leerräume keine erkennbare Rolle spielen, spricht man von *ignorierbaren Whitespaces*, bzw. gemäss Standard von "*white space in element content*". Streng formal gesehen handelt es sich um Text-Knoten.

Alle Elemente innerhalb eines Elements bezeichnet man als *Abkömmlinge* (engl. *descendants*). Die erste Ebene bezeichnet man auch als Kinder, aber nur diese Ebene.

Beispiel:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
- <Bestellung>
    <Kunde id="MM32">Media Markt</Kunde>
  - <Produkt>
      <Name>Bill Gates</Name>
      <SKU>244</SKU>
      <Quantität>12</Quantität>
      <Preis Währung="EUR">21.95</Preis>
    </Produkt>
  - <Lieferadresse>
      <Strasse>Seestrasse 12</Strasse>
      <Stadt>Erlenbach</Stadt>
      <Kanton>ZH</Kanton>
      <PLTZ>8045</PLTZ>
    </Lieferadresse>
    <Subtotal Währung="EUR">263.40</Subtotal>
    <MWSt Ansatz="7.0" Währung="EUR">18.44</MWSt>
    <Lieferart Methode="UPS" Währung="EUR">8.95</Lieferart>
    <Total Währung="EUR">290.79</Total>
  </Bestellung>
```

Das Element <Bestellung> besitzt 15 Abkömmlinge, aber lediglich 7 Kinder:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
- <Bestellung>
    <Kunde id="MM32">Media Markt</Kunde>
  + <Produkt>
  + <Lieferadresse>
    <Subtotal Währung="EUR">263.40</Subtotal>
    <MWSt Ansatz="7.0" Währung="EUR">18.44</MWSt>
    <Lieferart Methode="UPS" Währung="EUR">8.95</Lieferart>
    <Total Währung="EUR">290.79</Total>
  </Bestellung>
```

Ein Element kann auch einen gemischten Inhalt besitzen, wenn beispielsweise neben Kindern auch Textinhalte vorhanden sind.

Beispiel:

```
<ShipTo>
  LAN-Party
  <Street>Oberseestrasse</Street >
  <City>Rapperswil</City>
  <State>SG</State>
  <Zip>8640</Zip>
</ShipTo>
```

Gemischte Inhalte werden in der Praxis sehr häufig verwendet, ausser in Daten-orientierten Anwendungen (bei denen jedes Datenelement dem Inhalt eines Tags entspricht).

1.2.6. Text

XML Dokumente bestehen aus Text, also aus Sequenzen von Zeichen. Diese Zeichen können aus beliebigen Unicode Zeichen zusammengesetzt sein. Einzige Bedingung: der Parser muss diese Zeichen verarbeiten können.

1.2.7. Hinweis

Ein generelles Missverständnis bezüglich XML im Vergleich zu binären Dateiformaten ist, dass XML Dokumente mehr Speicherplatz als binäre Dateien benötigen.
Im Allgemeinen stimmt dies nicht. Vergleichen Sie ein Word Dokument mit einem XML Dokument und Sie werden erkennen, dass diese Annahme falsch ist. Der Hauptgrund für die Grösse eines Dokuments sind die vielen Formate, welche unterstützt werden müssen.
Ein weiteres Missverständnis: Unicode benötigt zwei Bytes. Dies stimmte nur zu Beginn der Unicode Definition. Insgesamt gibt es mehr Zeichen in Unicode, als mit zwei Bytes darstellbar sind.

1.2.8. Attribute

Attribute sind <Namen,Wert> Paare. Der Name eines Attributs ist ein beliebiger, gültiger XML Name. Wert eines Attributs können beliebige Zeichenketten sein, in Anführungszeichen; Trennzeichen ist das Gleichheitszeichen: `WAEHRUNG="CHF"`.

Die Reihenfolge bei mehreren Attributen spielt keine Rolle:
`<Wetterdaten Temp="30", Humid="56">Zuerich</Wetterdaten>` ist gleichwertig mit

```
<Wetterdaten Humid="56" Temp="30",>Zuerich</Wetterdaten>.
```

Attributwerte, ausser CDATA, werden normalisiert d.h. führende und abschliessende Whitespaces werden ignoriert ausser der Datentyp wird nicht angegeben. Sobald eine DTD mitgeliefert wird, werden Whitespaces ignoriert.

1.2.8.1. Hinweis

Tim Bray, einer der Autoren der XML Spezifikation für XML 1.0, hat mehrfach betont, dass die Normalisierung zu vielen Fehlern führt: "Why the $#%%!@! should attribute values be 'normalized' anyhow? This was a pure process failure: at no point during the 18-month development cycle of XML 1.0 did anyone stand up and say 'why are you doing this?' I'd bet big bucks that if someone had, the silly thing would have died a well-deserved death." [1]

1.2.9. XML Deklaration

Viele XML Dokumente starten mit einer XML Deklaration. Eine XML Deklaration besitzt ein `version` Attribut, dessen Wert `1.0` ist. Optional können weitere Attribute angegeben werden: `standalone`, `encoding`.

Beispiel:

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>
```

Das Dokument wurde in XML 1.0 geschrieben, mit ISO-8859-1 (Latin-1) Zeichensatz; es benötigt kein DTD. Die Version ist immer 1.0.

1.2.10. Kommentare in XML

XML Kommentare sind wie HTML Kommentare aufgebaut: diese beginnen mit `<!--` und enden mit `-->`.

Beispiel:

```
<!—XML ist nicht so banal wie man denken könnte! -->
```

1.2.11. XML Verarbeitungsanweisungen (Processing Instructions)

Verarbeitungsanweisungen kann man als Metainformationen ansehen, also Informationen, welche einem bestimmten Software-Paket Instruktionen für die weitere XML Dokumentverarbeitung geben.

Syntax:

```
<? XML-Name des Targets Daten ?>
```

Processing Instructions können vor, nach oder innerhalb des Root Elements stehen. Häufig stehen PI's im Prolog des Dokuments.

Beispiel:

Apache Cocoon

```
<?cocoon-process type="xinclude"?>
```

Viele PI's verwenden ein Pseudoformat Attribut:

```
<?xml-stylesheet type="text/xml" href="limited.xsl"?>
```

---

[1] *Re: Attribute normalisation and character entities*, xml-dev mailing list, January 27, 2000

1.2.12. Entities

XML Dokumente sind nicht das selbe wie *XML Dateien*. Ein XML Dokument kann as verschiedenen XML Dateien bestehen. Die einzelnen speicherbaren Einheiten eines XML Dokuments werden als *Entities* bezeichnet. Jedes XML Dokument besitzt mindestens eine Entity, die Dokument Entity, welche das Root des Dokuments enthält. Jede andere Entity besitzt einen Namen und ist darüber eindeutig identifizierbar.

Entities kann man unterschiedlichen Kategorien zuteilen:

Intern oder extern:

Der Ersatztext einer *internen Entity* wird als Zeichenkette im Dokument selber, in deren DTD beschrieben.
Eine *externe Entity* besteht aus Text, welcher aus einer oder mehrerer Dateien oder URL's gelesen wird.

Parsed oder unparsed:

Eine *geparste Entität* enthält XML Elemente. Solche Entitäten können aus einzelnen Elementen oder hanzen XML Dokumenten bestehen, sofern auch ein Root vorhanden ist. Geparste Entitäten kann man sich als irgend etwas vorstellen, welches geparst und in ein anderes Dokument eingefügt wird, wobei das ganze XML Dokument wohlgeformt bleibt.

Eine *ungeparste Entität* kann alles enthalten, auch binäre Daten. Solche Daten werden nicht in bestehende XML Dokumente eingefügt. Eine Entity Deklaration enthält in diesem Fall eine URL in der DTD. Beispiel wäre ein GIF oder andere binäre Daten, die zu einem XML Dokument gehören. Unparsed Entities werden in der Praxis selten eingesetzt.

Generell oder Parameter:

Eine *generelle Entität* wird innerhalb eines Dokuments verwendet. Sie beginnt mit `&`.
Eine *Parameter-Entität* wird innerhalb einer DTD verwendet. Sie beginnt mit `%`.

Es sind nicht alle Kombinationen möglich. Die folgenden fünf Kombinationen sind erlaubt:

Interne geparste generelle Entities:

Dabei handelt es sich um Entity Referenzen wie `&` oder `&copy`, welche vollständig innerhalb einer DTD definiert werden.
Beispiel: Definition der `copy` Entität als Text “Copyright”:

```
<!ENTITY copy "Copyright">
```

Solche Entitäten werden in Element-Inhalten und als Attributwerte verwendet.

Externe geparste generelle Entities:

In diesem Fall wird der Text aus einer Datei / URL gelesen (statt der DTD).
Beispiel: Definition der `legal` Entity als Inhalt der URL `http://www.beispiel.net/legal.xml`:

```
<!ENTITY legal SYSTEM "http://www.example.com/legal.xml">
```

Externe ungeparste generelle Entities:

Solche Entitäten beziehen sich auf nicht XML, in der Regel binäre Daten. Sie werden ähnlich wie externe geparste Entities deklariert, allerdings mit einer zusätzlichenNotation.
Beispiel:
Ungeparste generelle Entität, welche auf ein Logo verweist, bei der URL `http://www.logos.net/logo.png` vom Typus `image/png`:

```
<!NOTATION PNG SYSTEM "image/png">
<!ENTITY logo SYSTEM "http://www.logos.net/logo.png" NDATA PNG>
```

Attribute verweisen mithilfe von ENTITY oder ENTITIES auf ungeparste Entities.
Beispiel:

```
<!ELEMENT figure EMPTY>
<!ATTLIST figure logo ENTITY #REQUIRED>
```

Eine dazugehörende Instanz des `figure` Element wäre:

```
<figure source="logo"/>
```

Der Parser selber stellt die Daten der ungeparsten Entität nicht direkt zur Verfügung. Diese Daten müssen mithilfe der Standard Java I/O Klassen gelesen werden.

Interne geparste Parameter Entities:

Interne geparste Parameter Entitäten beziehen sich auf DTD's. Der Ersatztext wird durch ein String Literal in der DTD repräsentiert.
Beispiel:
Die DocBook DTD defininiert die `intermod.redecl.module` Parameter Entity als IGNORE:

```
<!ENTITY % intermod.redecl.module "IGNORE">
```

Der Parameter `%intermod.redecl.module;` kann nur in der DTD, nicht im Dokument selber verwendet werden.

Externe geparste Parameter Entities:

Externe geparste Parameter Entitäten werden lediglich in DTD's eingesetzt. Der Ersatztext steht in einer gegebenen URL. Die Referenz auf die Entität beginnt mit einem `%` Zeichen.
Beispiel:
Die DocBook DTD definiert die dbpool Parameter Entität über eine `PUBLIC ID`, welche das DTD Fragment aus der Relativ-URL lädt `dbpoolx.mod`:

```
<!ENTITY % dbpool PUBLIC
"-//OASIS//ELEMENTS DocBook XML Information Pool V4.1.2//EN"
"dbpoolx.mod">
```

1.2.13. Namensräume – Namespaces

Namensräume gehören nicht zur ursprünglichen Definition von XML 1.0. Die Namensräume wurden etwa ein Jahr nach der XML Definition eingeführt, als bereits viele XML Applikationen in der Produktion waren. Dabei werden oft unterschiedliche Dokumente zu einem einzigen zusammengefasst.

Viele XML Applikationen kommen ohne Namensräume aus, beispielsweise XML-RPC.

Die Grundidee der Namensräume ist es, jedes Element an einen *Uniform Resource Identifier* (URI) (in der Praxis einen URL) zu binden:
Falls jede Firma lediglich URI's einsetzen, die deren Domain-Name enthalten, können keine Mehrdeutigkeiten auftreten, selbst wenn Dokumente unterschiedlicher Firmen gemeinsam in einem XML Dokument eingesetzt werden.

Bemerkung:
Eine URI identifiziert eine Ressource. Allerdings wird nichts über deren Speicherort ausgesagt. URI's umfassen sowohl URL's als auch URN's (Uniform Resource Names).
Beispiel:
Wenn wir die ISBN Nummer eines Buches zu dessen Identifizierung verwenden, sagt eine `urn:isbn:1020304050` noch nichts über den Ort des Buches aus.

Es hat sich eingebürgert, als URI lediglich absolute URL's (keine relative) einzusetzen. URI's sind reine Zeichenketten, sie werden also selbst als URL's nicht überprüft.

Beispiel:
Die folgenden URL's sind physisch identisch, als Namensraum URI aber unterschiedlich:
```
http://www.oceanmammalinst.org/index.htm
http://www.oceanmammalinst.org/
http://www.oceanmammalinst.org
```

In der Regel definiert man einen Kurznamen anstelle des Namensraumes. Der Namensraum wird durch einen Präfix repräsentiert.

Beispiel:
```
<Order xmlns:xi="http://www.w3.org/2001/XInclude">
  <xi:include href="order_details.xml"/>
</Order>
```

`xi` wird als Präfix bezeichnet, mittels `xmlns:xi` oder allgemeiner `xmlns:`*`prefix`* wird der Präfix an eine URI gebunden. Der Zugriff auf einen *lokalen Namen* geschieht qualifiziert mit einem Präfix: `xi:include`, das ganze wird als *qualifizierter Name* bezeichnet.

Falls vergessen wird, einen Präfix an einen Namensraum zu binden, wird ein *namespace well-formedness error* geworfen.

Für ein Dokument kann auch ein Default-Namensraum definiert werden.

Beispiel:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Order xmlns="http://ns.global.net/Beispiele/Bestellungen/">
  <Customer id="ZH31">Schwarzwald Timing</Customer>
  <Product>
    <Name>Kuckuck Uhr</Name>
    <SKU>244</SKU>
    <Quantity>12</Quantity>
    <Price currency="EUR">35.00</Price >
    <ShipTo>
      <Street>Waldhütte</Street >
      <City>Schwarzwaldsee</City> <State>BW</State> <Zip>12345</Zip>
    </ShipTo>
  </Product>
  <Subtotal currency='EUR'>420.00</Subtotal>
  <Tax rate="7.0"
       currency='EUR'>28. </Tax>
  <Shipping  method="UPS" currency='EUR'>12.00</Shipping>
  <Total currency='EUR' >460.00</Total>
</Order>
```

In der Regel steht der Namensraum gleich am Anfang des XML Dokuments. Es gibt aber keine Vorschriften dazu, wie das folgende Beispiel zeigt.

Beispiel:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Order xmlns="http://ns.global.net/Beispiele/Bestellungen/">
  <Customer id="ZH31">Schwarzwald Timing</Customer>
  <Product>
    <Name>Kuckuck Uhr</Name>
    <SKU>244</SKU>
    <Quantity>12</Quantity>
    <Price currency="EUR">35.00</Price >
    <ShipTo xmlns="http://ns.global.net/Beispiele/Adresse/">
      <Street>Waldhütte</Street >
      <City>Schwarzwaldsee</City> <State>BW</State> <Zip>12345</Zip>
    </ShipTo>
  </Product>
  <Subtotal currency='EUR'>420.00</Subtotal>
  <Tax rate="7.0"
       currency='EUR'>28. </Tax>
  <Shipping  method="UPS" currency='EUR'>12.00</Shipping>
  <Total currency='EUR' >460.00</Total>
</Order>
```

Eher selten werden Namensräume für Attribute definiert, XLink ist ein Anwendungsbeispiel.

Beispiel:

```
    <ShipTo xmlns="http://ns.global.net/Beispiele/Adresse/"
         xmlns:xlink="http://www.w3.org/1999/xlink"
         xlink:type="simple" xlink:href="mailto:joller@bluemail.ch">
      <Recipient>Patrick Fischer</Recipient>
      <Street>Beatushöhle</Street >
      <City>Brienzer Rotstock</City> <State>BE</State> <Zip>0070</Zip>
    </ShipTo>
```

1.3. Gültigkeit

Die Gültigkeit eines XML Dokuments kann mithilfe eines Schemas oder einer DTD, Document Type Definition, festgelegt werden. Die Gültigkeit entspricht der Korrektheit eines Programms, DTD / Schema sind in diesem Fall die Syntax der Sprache.

Die *Validation* eines Dokuments entspricht der Syntaxprüfung. Diese ist für ein XML Dokument nicht zwingend erforderlich.

1.3.1. DTD's

DTD beschreiben die Struktur eines Dokuments aus der Sicht seiner Elemente, also: welche Elemente kann das Dokument enthalten, wie oft, welche Attribute und von welchem Typus.

1.3.1.1. Element Deklarationen

Jedes Element eines gültigen XML Dokuments muss in seiner DTD als *ELEMENT* deklariert sein.

Beispiel:

```
<!ELEMENT Name (#PCDATA)>
```

Das Element Name enthält lediglich `#PCDATA`, also Text, aber keine Kinderknoten.

Und hier noch ein Beispiel mit Kindknoten:

```
<!ELEMENT Bestellung (Kunde, Produkt, Subtotal, MWSt, Fracht, Total)>
```

In der Klammer steht das sogenannte *Content Model* des Elements.
Die Anzahl Elemente kann mit einer Zusatzangabe angegeben werden:

```
? : 0..1
*: 0...
+: 1...
```

Beispiel:

```
<!ELEMENT Lieferadresse (Empfänger?, Strasse+, Stadt, Staat, PLTZ)>
```

falls der Empfänger nicht bekannt, aber die Lieferadresse bekannt ist.

Die *ODER* Verknüpfung von Elementen wird durch einen senkrechten Strich angegeben.

Beispiel:

```
<!ELEMENT Adresse (Lieferadresse | Hauptsitzadresse)>
```

Falls ein Element keinen Inhalt besitzen muss, kann anstelle des Content Modells auch einfach `EMPTY` stehen.

1.3.1.2. Attribut Deklarationen

Die Attribute eines XML Elements werden in der DTD mittels Attributlisten, der `ATTLIST` *Deklaration* spezifiziert. Eine Attributliste enthält folgende Informationen:

- Das Element zu dem das Attribut gehört
- Der Name des Attributs
- Der Standardwert des Attributs

Beispiel:

```
<!ATTLIST Artikel ser_nr ID #REQUIRED>
```

Jeder Artikel muss eine Serienummer besitzen. Diese muss vom Typ ID sein.

Gemäss der DTD Spezifikation können die folgenden zehn Attributtypen verwendet werden:

`CDATA`
: Das Attribut kann ein beliebiger Text sein (Standardtyp für Attribute).

`NMTOKEN`
: Das Attribut besteht aus Text, wobei der Namens- Token auch mit einer Zahl beginnen darf.

`NMTOKENS`
: Das Attribut besteht aus einer Liste von durch Whitespaces getrennten Namens-Token.

`ID`
: Das Attribut besteht aus einem im XML Dokument eindeutigen Schlüssel vom Typ ID.

`IDREF`
: Ein XML Name, welcher als ein ID Attribut Wert eines Elements in einem Dokument verwendet wird.

`IDREFS`
: Eine durch Whitespaces getrennte Liste von XML Namen, welche als ID Attribute im Dokument verwendet werden.

`ENTITY`
: Der Name einer ungeparsten Entity, welche in einer ENTITY Deklaration in der DTD verwendet wird.

`ENTITIES`
: Eine durch Whitespaces getrennte Liste ungeparster Entities, welche in der DTD deklariert werden.

`NOTATION`
: Der Name einer Notation, welche in einer NOTATION Deklaration in der DTD steht.

`Enumeration`
: Eine Liste aller legalen Werte für das Attribut, durch einen senkrechten Strich getrennt. Jeder mögliche Wert muss ein XML Namens- Token sein.

Die meisten Parser und API's erkennen den Attribut-Typ. Mit XML Schema können sehr unterschiedliche Datentypen definiert werden; mit DTD ist die Auswahl geringer.

DTD's gestatten vier Standardwerte für Attribute:

`#REQUIRED`
: Jede Instanz der XML DTD muss für diese Attribute einen Wert besitzen.

`#IMPLIED`
: Jede Instanz der XML DTD kann, muss aber nicht einen Wert für dieses Attribut besitzen. Falls kein Wert festgelegt wird, stellt DTD keinen Standardwert zur Verfügung

`#FIXED "`*`value`*`"`
: Der Wert des Attributs ist immer der nach `#FIXED` in einfachen oder doppelten Anführungszeichen, selbst wenn das Attribut im XML Dokument nicht explizit erscheint.

"*`value`*"
: Jedes Dokument zur DTD besitzt diesen in der DTD spezifizierten Wert.

Das folgende Beispiel enthält eine vollständige DTD, mit ELEMENT und ATTLIST.

Beispiel:
Bestellung-DTD

```
<!ELEMENT Order (Customer, Product+, Subtotal, Tax, Shipping, Total)>
<!ELEMENT Customer (#PCDATA)>

<!ATTLIST Customer id ID #REQUIRED>

<!ELEMENT Product (Name, SKU, Quantity, Price, Discount?,
                   ShipTo, GiftMessage?)>
<!ELEMENT Name (#PCDATA)>
<!ELEMENT Serial-Nr (#PCDATA)>
<!ELEMENT Quantity (#PCDATA)>
<!ELEMENT Price (#PCDATA)>

<!ATTLIST Price currency (CHF | EUR | GBP) #REQUIRED>

<!ELEMENT Discount (#PCDATA)>
<!ELEMENT Address (GiftRecipient?, Street+, City, State, Zip)>
<!ELEMENT GiftRecipient (#PCDATA)>
<!ELEMENT Street (#PCDATA)>
<!ELEMENT City   (#PCDATA)>
<!ELEMENT State  (#PCDATA)>
<!ELEMENT Zip    (#PCDATA)>
<!ELEMENT GiftMemo (#PCDATA)>
<!ELEMENT Subtotal (#PCDATA)>

<!ATTLIST Subtotal currency (CHF | EUR | GBP) #REQUIRED>
<!ELEMENT Tax (#PCDATA)>
<!ATTLIST Tax currency (CHF | EUR | GBP) #REQUIRED
              rate CDATA "0.0">
<!ELEMENT Shipping (#PCDATA)>
<!ATTLIST Shipping currency (CHF | EUR | GBP) #REQUIRED
                   method   (USPS | UPS | Overnight) "UPS">
<!ELEMENT Total (#PCDATA)>
<!ATTLIST Total currency (CHF | EUR | GBP) #REQUIRED>
```

1.3.1.3. Dokument Typ Deklarationen

Gültigen XML Dokumenten sind DTD's oder XML Schemas zugeordnet. Die Zuordnung einer DTD zu einem XML Dokument geschieht auf folgende Art und Weise:

```
<!DOCTYPE Order SYSTEM "bestellung.dtd">
```

Die DTD kann entweder im XML Dokument stehen, im Prolog, aber vor dem Wurzel-Element.

Beispiel:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE Order SYSTEM "bestellung.dtd">
<Order>
  ...
```

Die DTD kann aber auch in einer separaten Datei abgespeichert werden.

Beispiel:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE Order SYSTEM "http://www.dokumente.org/xml/dtds/bestellung.dtd">
<Order>
  ...
```

Hinweis:

DTD steht für "document type *definition*", also *nicht* "document type *declaration*". Die Dokument D*eklaration* kann eine DTD enthalten oder auf eine verweisen.

DTD's werden nicht nur zur Validation eines Dokuments eingesetzt. Wie wir oben gesehen haben kann eine DTD:

- Entities definieren
- Notations definieren
- Standardwerte für Attribute zur Verfügung stellen.

1.3.2. Schemas

Die W3C XML Schema Sprache kümmert sich um einige der Defizite von DTD's:

- Schemas werden in Standard-XML spezifiziert.
- Schemas verwenden XML Namensräume.
- Schemas kennen unterschiedliche Datentypen, wie Integer u.ä.

Einzelnen DTD Konzepten entsprechen Schema Inhalte entsprechen sich grob folgendermassen:

- Elemente: in DTD `ELEMENT`, in Schema `xsd:element`
- Attributliste: in DTD `ATTLIST`, in Schema `xsd:attribute`.

Zusätzlich enthält aber ein Schema viele zusätzliche Datentypen und Subtypen.

Beispiel:

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <xsd:element name="Bestellung">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="Kunde">
          <xsd:complexType>
            <xsd:simpleContent>
              <xsd:extension base="xsd:string">
                <xsd:attribute name="id" type="xsd:ID"/>
```

```
            </xsd:extension>
          </xsd:simpleContent>
        </xsd:complexType>
       </xsd:element>
      <xsd:element name="Produkt" maxOccurs="unbounded">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="name"     type="xsd:string"/>
            <xsd:element name="sku"
             type="xsd:positiveInteger"/>
            <xsd:element name="quantitaet"
             type="xsd:positiveInteger"/>
            <xsd:element name="preis"    type="MoneyType"/>
            <xsd:element name="discount" type="xsd:decimal"
                         minOccurs="0"/>
            <xsd:element name="lieferadresse">
              <xsd:complexType>
                <xsd:sequence>
                  <xsd:element name="geschenkadresse"
                   type="xsd:string"
                   minOccurs="0" maxOccurs="unbounded"/>
                  <xsd:element name="strasse"
                   type="xsd:string"/>
                  <xsd:element name="stadt" type="xsd:string"/>
                  <xsd:element name="staat"
                   type="xsd:string"/>
                  <xsd:element name="pltz" type="xsd:string"/>
                </xsd:sequence>
              </xsd:complexType>
            </xsd:element>
            <xsd:element name="geschenktext" type="xsd:string"
                         minOccurs="0"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="subtotal" type="MoneyType"/>
      <xsd:element name="steuer">
        <xsd:complexType>
          <xsd:simpleContent>
            <xsd:extension base="MoneyType">
              <xsd:attribute name="rate" type="xsd:decimal"/>
            </xsd:extension>
          </xsd:simpleContent>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="transport">
        <xsd:complexType>
          <xsd:simpleContent>
            <xsd:extension base="MoneyType">
              <xsd:attribute name="method" type="xsd:string"/>
            </xsd:extension>
          </xsd:simpleContent>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="total" type="MoneyType"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<xsd:complexType name="MoneyType">
  <xsd:simpleContent>
    <xsd:extension base="xsd:decimal">
      <xsd:attribute name="currency" type="xsd:string"/>
    </xsd:extension>
  </xsd:simpleContent>
</xsd:complexType>

</xsd:schema>
```

Eine verbreitete Art und Weise einem XML Dokument ein XML Schema zuzuweisen ist mithilfe eines `xsi:noNamespaceSchemaLocation` Attributs im Wurzelelement. `xsi` steht dabei für `http://www.w3.org/2001/XMLSchema-instance`.

Beispiel:
```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Order xsi:noNamespaceSchemaLocation="bestellung.xsd"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  ...
```

XML Schemas sind ziemlich komplex und werden nicht sehr häufig eingesetzt. Schemas werden eingesetzt:
- zur Validierung von XML Dokumenten
- bei der Umwandlung eines XML Dokuments in einen Java Programm-Rahmen.

Sie können in XML Spy oder ähnlichen Tools zu einem XML Dokument ein XML Schema oder eine DTD generieren lassen, oder umgekehrt zu einem Schema ein XML Dokument, eine Instanz des Schemas, erstellen lassen.

1.4. Stylesheets

In der Regel verwenden wir XML als Eingabe und Ausgabe, ohne dass uns die Darstellung des XML Dokuments interessiert. Falls die Darstellung von Interesse ist, bieten sich Style-Sheets an, je ein Style-Sheet pro Ausgabeformat.

Heute kennt man zwei grundlegende Format-Beschreibungssprachen:
- Cascading Style Sheets (CSS)
- Extensible Stylesheet Language (XSL)

1.4.1. CSS

Beim CSS werden jedem Element im XML Dokument bestimmte Formate zugewiesen. CSS wird auch für HTML eingesetzt.

Beispiel:
```
<Kunde>Peterhans</Kunde>
Kunde {font-weight: bold}
```
oder
```
Order {font-family: serif; font-size: 16pt}
```

Falls beispielsweise ein Font nicht verfügbar ist, wird auf einen andern Font gewechselt oder ein System Font verwendet.

```
GiftMessage {font-family: ZapfChancery, script}
```

Falls eine Formatangabe für mehrere Elemente gelten soll, müssen diese durch Kommas getrennt, aufgeführt werden:
```
Street, Subtotal, Tax, Shipping, Total {display: block}
```

Beispiel:
```
Order {font-family: serif; font-size: 16pt;
       display: block; line-height: 20pt;
       margin-left: 0.25in
      }
ShipTo {margin-left: 0.5in; display: block }
ShipTo:before {content: "Ship to:"; margin-left: -0.25in }
Product {font-size: 12pt; display: block }
```

```
Customer {font-weight: bold; display: block }
GiftMessage {font-family: ZapfChancery, script}
Street, Subtotal, Tax, Shipping, Total {display: block}
Quantity:before {content: "Quantity: "}
SKU {display: none}
Subtotal:before {content: "Subtotal: $"}
Price:before {content: "Unit Price: $"}
Tax:before {content: "Tax: $"}
Shipping:before {content: "Shipping: $"}
Total:before {content: "Total: $"}
```

1.4.2. Style Sheets für XML Dokumente

Style-Sheets werden mithilfe einer `xml-stylesheet` Processing Instruction in ein XML Dokument eingebunden, eventuell über eine `href` Angabe.

Beispiel:

```
<?xml-stylesheet type="text/css" href="order.css"?>
```

Die folgenden Beispiele zeigen, welche Wirkung ein Style-Sheet haben kann, bei der Anzeige in einem Browser.

Beispiel:

XML-Dokument `book.html`

```
<!-- Dateiname: Book.htm -->
<HTML>
<HEAD>
   <TITLE>Buchbeschreibung</TITLE>
</HEAD>
<BODY>
   <XML ID="dsoBook" SRC="Book.xml"></XML>
   <H2>Buchbeschreibung</H2>
   <SPAN STYLE="font-style:italic">Titel: </SPAN>
   <SPAN STYLE="font-weight:bold" DATASRC="#dsoBook"
      DATAFLD="TITLE"></SPAN>
   <BR>
   <SPAN STYLE="font-style:italic">Autor: </SPAN>
   <SPAN DATASRC="#dsoBook" DATAFLD="AUTHOR"></SPAN>
   <BR>
   <SPAN STYLE="font-style:italic">Bindung: </SPAN>
   <SPAN DATASRC="#dsoBook" DATAFLD="BINDING"></SPAN>
   <BR>
   <SPAN STYLE="font-style:italic">Seitenzahl: </SPAN>
   <SPAN DATASRC="#dsoBook" DATAFLD="PAGES"></SPAN>
   <BR>
   <SPAN STYLE="font-style:italic">Preis: </SPAN>
   <SPAN DATASRC="#dsoBook" DATAFLD="PRICE"></SPAN>
</BODY></HTML>
```

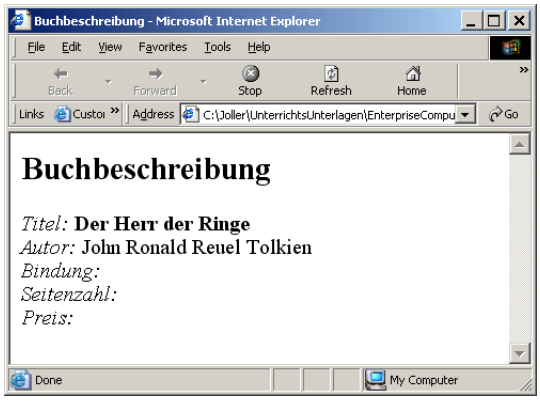
XML Datei: `book.xml`

```
<?xm version="1.0"?>

<!DOCTYPE BOOK
      [
      <!ELEMENT BOOK (TITLE, AUTHOR, COVERIMAGE)>
      <!ELEMENT TITLE (#PCDATA)>
      <!ELEMENT AUTHOR (#PCDATA)>
      <!ELEMENT COVERIMAGE EMPTY>
      <!ATTLIST COVERIMAGE Source ENTITY #REQUIRED>
      <!NOTATION GIF SYSTEM "ShowGif.exe">
      <!ENTITY ringe SYSTEM "herrDerRinge.gif" NDATA GIF>
      ]
>
<BOOK>
      <TITLE>Der Herr der Ringe</TITLE>
      <AUTHOR>John Ronald Reuel Tolkien</AUTHOR>
      <COVERIMAGE Source="ringe"/>
```

```
</BOOK>
```

Anzeigebeispiel im Internet Explorer:



1.4.3. XSL

CSS beschränkt sich darauf, vorhandene Elemente in der vorhandenen Reihenfolge zu präsentieren.

XSL kann wesentlich mehr. Elemente können neu angeordnet oder weggelassen werden. Es können auch neue Texte aus dem XSL sichtbar werden.

XSL wird normalerweise aufgeteilt in die zwei Komponenten:

- XSLT : XSL Transformations und
- XSL- FO, Formatting Objects.

Beides verwenden XML konforme Spezifikationen.

XSLT ist eine Turing-vollständige *funktionale Programmiersprache,* welche speziell entwickelt wurde, um Transformationen von einem XML Format in ein anderes beschreiben zu können (eine funktionale Programmiersprache besitzt keine Seiteneffekte, die Auswertung eines Ausdrucks [von links nach rechts oder umgekehrt] hat also keine Auswirkung auf das Endergebnis).

Ein XSLT Prozessor liest ein XML Dokument und ein XSLT Stylesheet und transformiert das Dokument gemäss den Angaben im Stylesheet und gibt dieses aus. Die Ausgabe kann auch in HTML, PDF oder irgend einem andern Format erfolgen (TeX, troff, ...).

XSL-FO ist eine (bei Apache herunterladbare) XML Applikation, welche das Layout der Elemente für's Drucken gestattet (in unterschiedlichen Formaten).

Beispiel:

```
C:\fop-0.20.5>fop -fo C:\fop-0.20.5\examples\fo\basic\readme.fo -pdf
readmeEx.pdf
[INFO] Using org.apache.xerces.parsers.SAXParser as SAX2 Parser
[INFO] FOP 0.20.5
[INFO] Using org.apache.xerces.parsers.SAXParser as SAX2 Parser
[INFO] building formatting object tree
[INFO] setting up fonts
[INFO] [1]
[INFO] [2]
[INFO] [3]
[INFO] [4]
[INFO] [5]
[INFO] [6]
```

```
[INFO] [7]
[INFO] [8]
[INFO] [9]
[INFO] [10]
[INFO] Parsing of document complete, stopping renderer
```

Eingabe:
eine fo-Datei
`-fo C:\fop-0.20.5\examples\fo\basic\readme.fo`
Ausgabe:
eine PDF Datei
`-pdf readmeEx.pdf`

Der ganze Text steht in diesem Fall im `.fo` File.

In XSLT basiert auf *Templates*:
Der XSLT Prozessor liest das XML Dokument und vergleicht die Knoten im Eingabebaum mit Templates im Stylesheet. Falls für den Knoten ein Template gefunden wird, wird der Knoten gemäss Template formatiert.

Im folgenden Beispiel wird die Formattieranweisung (`.fo`) in einer separaten Datei abgespeichert, nicht zusammen mit dem Text wie in obigem Beispiel.

Beispiel:
Zuerst das XML Dokument mit den Bestelldaten

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Order>
  <Customer id="c32">Klaus Peters</Customer>
  <Product>
    <Name>Design Patterns</Name>
    <SKU>123</SKU>
    <Quantity>1</Quantity>
    <Price currency="EUR">90</Price>
    <ShipTo>
      <Street>Seestrasse 10</Street>
      <City>Erlenbach</City>
      <State>ZH</State>
      <Zip>8123</Zip>
    </ShipTo>
  </Product>
  <Product>
    <Name>Architektur-Patterns</Name>
    <SKU>234</SKU>
    <Quantity>1</Quantity>
    <Price currency="EUR">100.00</Price>
    <Discount>.10</Discount>
    <ShipTo>
      <GiftRecipient>Peter Herrmann</GiftRecipient>
      <Street>Guldenenweg 12</Street>
      <City>Klausthal</City>
      <State>SG</State>
      <Zip>9789</Zip>
    </ShipTo>
    <GiftMessage>
     Endlich- wie versprochen!
```

```
      Viel Spass beim Lesen
      Peter und Klaus
    </GiftMessage>
  </Product>
  <Subtotal currency="EUR">180.00</Subtotal>
  <Tax rate="7.0" currency="EUR">192.60</Tax>
  <Shipping method="Post" currency="EUR">10</Shipping>
  <Total currency="EUR">202.60.00</Total>
</Order>
```

Nun das Stylesheet

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

  <!-- Try to make the output look half decent -->
  <xsl:output indent="yes"/>

  <xsl:template match="Order">
    <fo:root>

      <fo:layout-master-set>
        <fo:simple-page-master master-name="only">
          <fo:region-body margin-left="0.5in"
                          margin-top="0.5in"/>
        </fo:simple-page-master>
      </fo:layout-master-set>

      <fo:page-sequence master-reference="only">

        <fo:flow flow-name="xsl-region-body">
          <xsl:apply-templates select="Customer"/>
          <xsl:apply-templates select="ShipTo"/>
          <fo:table font-size="12pt"
                    space-before="24pt" space-after="24pt">
            <fo:table-column column-width="2in"/>
            <fo:table-column column-width="1in"/>
            <fo:table-column column-width="1in"/>
            <fo:table-column column-width="1in"/>
            <fo:table-body>
              <fo:table-row font-weight="bold">
                <fo:table-cell>
                  <fo:block>Product</fo:block>
                </fo:table-cell>
                <fo:table-cell>
                  <fo:block>Quantity</fo:block>
                </fo:table-cell>
                <fo:table-cell>
                  <fo:block>Unit Price</fo:block>
                </fo:table-cell>
                <fo:table-cell>
                  <fo:block>Subtotal</fo:block>
                </fo:table-cell>
              </fo:table-row>
              <xsl:apply-templates select="Product"/>
```

```
          </fo:table-body>
        </fo:table>
        <xsl:apply-templates select="Tax"/>
        <xsl:apply-templates select="Shipping"/>
        <xsl:apply-templates select="Total"/>
      </fo:flow>

    </fo:page-sequence>

  </fo:root>
</xsl:template>

<xsl:template match="Customer">
  <fo:block font-size="16pt" font-family="serif"
            line-height="20pt">
     Ship to:
  </fo:block>
  <fo:block font-size="16pt" font-family="serif"
            margin-left="0.5in" line-height="20pt">
     <xsl:apply-templates/>
  </fo:block>
</xsl:template>

<xsl:template match="ShipTo">
  <fo:block font-size="16pt" font-family="sans-serif"
            line-height="18pt" margin-top="20pt"
            margin-left="0.5in">
    <xsl:apply-templates select="Street"/>
  </fo:block>
  <fo:block font-size="16pt" font-family="sans-serif"
            line-height="18pt" margin-left="0.5in">
    <xsl:apply-templates select="City"/> 
    <xsl:apply-templates select="State"/> 
    <xsl:apply-templates select="Zip"/>
  </fo:block>
</xsl:template>

<xsl:template match="Product">
  <fo:table-row>
    <fo:table-cell>
      <fo:block><xsl:value-of select="Name"/></fo:block>
    </fo:table-cell>
    <fo:table-cell>
      <fo:block><xsl:value-of select="Quantity"/></fo:block>
    </fo:table-cell>
    <fo:table-cell>
      <fo:block> EUR <xsl:value-of select="Price"/></fo:block>
    </fo:table-cell>
    <fo:table-cell>
      <fo:block>
        EUR <xsl:value-of select="Price*Quantity"/>
      </fo:block>
    </fo:table-cell>
  </fo:table-row>
</xsl:template>
```

```
  <xsl:template match="Tax|Shipping|Total">
    <fo:block font-size="16pt" font-family="serif"
              line-height="20pt">
      <xsl:value-of select="name()"/>: EUR <xsl:apply-templates/>
    </fo:block>
  </xsl:template>

  <!-- want to leave this one out of the output -->
  <xsl:template match="SKU"/>

</xsl:stylesheet>
```

Mithilfe von XML Spy habe ich (XSL->XSL-FO Transformation) die Bestellung mithilfe des obigen Stylesheets in folgende Datei transformiert:

```
<?xml version="1.0"?>
<!-- produced by null -->
<AreaTree>
  <Page number="1">
    <BodyAreaContainer>
      <AreaContainer name="before-float-reference-area">
      </AreaContainer>
      <AreaContainer name="main-reference-area">
        <SpanArea>
          <AreaContainer name="normal-flow-ref.-area">
            <BlockArea start-indent="0" end-indent="0" is-
first="true" is-last="true" generated-
by="fo:block//org.apache.fop.fo.flow.Block@89cc5e">
              <DisplaySpace size="2000"/>
              <LineArea font-weight="normal">
                <WordArea font-weight="normal" red="0.0"
green="0.0" blue="0.0" width="29344">Ship</WordArea>
                <InlineSpace size="4000"/>
                <WordArea font-weight="normal" red="0.0"
green="0.0" blue="0.0" width="16896">to:</WordArea>
                <InlineSpace size="4000"/>
              </LineArea>
              <DisplaySpace size="2000"/>
            </BlockArea>
            <BlockArea start-indent="36000" end-indent="0" is-
first="true" is-last="true" generated-
by="fo:block//org.apache.fop.fo.flow.Block@c79809">
              <DisplaySpace size="2000"/>
              <LineArea font-weight="normal">
                <WordArea font-weight="normal" red="0.0"
green="0.0" blue="0.0" width="37328">Klaus</WordArea>
                <InlineSpace size="4000"/>
                <WordArea font-weight="normal" red="0.0"
green="0.0" blue="0.0" width="39104">Peters</WordArea>
              </LineArea>
              <DisplaySpace size="2000"/>
            </BlockArea>
            <DisplaySpace size="24000"/>
            <AreaContainer name="null">
              <AreaContainer name="null">
              </AreaContainer>
              <AreaContainer name="null">
```

```
                </AreaContainer>
                  <AreaContainer name="null">
                </AreaContainer>
                  <AreaContainer name="null">
                </AreaContainer>
                  <AreaContainer name="null">
                    <AreaContainer name="null">
                      <AreaContainer name="null">
                        <BlockArea start-indent="0" end-indent="0"
is-first="true" is-last="true" generated-
by="fo:block//org.apache.fop.fo.flow.Block@1ce784b">
                          <DisplaySpace size="1200"/>
                          <LineArea font-weight="bold">
                            <WordArea font-weight="bold" red="0.0"
green="0.0" blue="0.0" width="45336">Product</WordArea>
                          </LineArea>
                          <DisplaySpace size="1200"/>
                        </BlockArea>
                      </AreaContainer>
                      <AreaContainer name="null">
                        <BlockArea start-indent="0" end-indent="0"
is-first="true" is-last="true" generated-
by="fo:block//org.apache.fop.fo.flow.Block@3bc473">
                          <DisplaySpace size="1200"/>
                          <LineArea font-weight="bold">
                            <WordArea font-weight="bold" red="0.0"
green="0.0" blue="0.0" width="48672">Quantity</WordArea>
                          </LineArea>
                          <DisplaySpace size="1200"/>
                        </BlockArea>
                      </AreaContainer>
                      <AreaContainer name="null">
                        <BlockArea start-indent="0" end-indent="0"
is-first="true" is-last="true" generated-
by="fo:block//org.apache.fop.fo.flow.Block@1de9ac4">
                          <DisplaySpace size="1200"/>
                          <LineArea font-weight="bold">
                            <WordArea font-weight="bold" red="0.0"
green="0.0" blue="0.0" width="23328">Unit</WordArea>
                            <InlineSpace size="3336"/>
                            <WordArea font-weight="bold" red="0.0"
green="0.0" blue="0.0" width="29352">Price</WordArea>
                          </LineArea>
                          <DisplaySpace size="1200"/>
                        </BlockArea>
                      </AreaContainer>
                      <AreaContainer name="null">
                        <BlockArea start-indent="0" end-indent="0"
is-first="true" is-last="true" generated-
by="fo:block//org.apache.fop.fo.flow.Block@182bcde">
                          <DisplaySpace size="1200"/>
                          <LineArea font-weight="bold">
                            <WordArea font-weight="bold" red="0.0"
green="0.0" blue="0.0" width="48000">Subtotal</WordArea>
                          </LineArea>
                          <DisplaySpace size="1200"/>
```

```
                    </BlockArea>
                  </AreaContainer>
                </AreaContainer>
                <DisplaySpace size="13500"/>
                <AreaContainer name="null">
                  <AreaContainer name="null">
                    <BlockArea start-indent="0" end-indent="0"
is-first="true" is-last="true" generated-
by="fo:block//org.apache.fop.fo.flow.Block@3dc0bb">
                      <DisplaySpace size="1200"/>
                      <LineArea font-weight="normal">
                        <WordArea font-weight="normal" red="0.0"
green="0.0" blue="0.0" width="37344">Design</WordArea>
                        <InlineSpace size="3336"/>
                        <WordArea font-weight="normal" red="0.0"
green="0.0" blue="0.0" width="44688">Patterns</WordArea>
                      </LineArea>
                      <DisplaySpace size="1200"/>
                    </BlockArea>
                  </AreaContainer>
                  <AreaContainer name="null">
                    <BlockArea start-indent="0" end-indent="0"
is-first="true" is-last="true" generated-
by="fo:block//org.apache.fop.fo.flow.Block@384065">
                      <DisplaySpace size="1200"/>
                      <LineArea font-weight="normal">
                        <WordArea font-weight="normal" red="0.0"
green="0.0" blue="0.0" width="6672">1</WordArea>
                      </LineArea>
                      <DisplaySpace size="1200"/>
                    </BlockArea>
                  </AreaContainer>
                  <AreaContainer name="null">
                    <BlockArea start-indent="0" end-indent="0"
is-first="true" is-last="true" generated-
by="fo:block//org.apache.fop.fo.flow.Block@10bbf9e">
                      <DisplaySpace size="1200"/>
                      <LineArea font-weight="normal">
                        <WordArea font-weight="normal" red="0.0"
green="0.0" blue="0.0" width="25332">EUR</WordArea>
                        <InlineSpace size="3336"/>
                        <WordArea font-weight="normal" red="0.0"
green="0.0" blue="0.0" width="13344">90</WordArea>
                      </LineArea>
                      <DisplaySpace size="1200"/>
                    </BlockArea>
                  </AreaContainer>
                  <AreaContainer name="null">
                    <BlockArea start-indent="0" end-indent="0"
is-first="true" is-last="true" generated-
by="fo:block//org.apache.fop.fo.flow.Block@513d61">
                      <DisplaySpace size="1200"/>
                      <LineArea font-weight="normal">
                        <WordArea font-weight="normal" red="0.0"
green="0.0" blue="0.0" width="25332">EUR</WordArea>
                        <InlineSpace size="3336"/>
```

```
                              <WordArea font-weight="normal" red="0.0"
green="0.0" blue="0.0" width="13344">90</WordArea>
                            </LineArea>
                            <DisplaySpace size="1200"/>
                          </BlockArea>
                        </AreaContainer>
                      </AreaContainer>
                      <DisplaySpace size="13500"/>
                      <AreaContainer name="null">
                        <AreaContainer name="null">
                          <BlockArea start-indent="0" end-indent="0"
is-first="true" is-last="true" generated-
by="fo:block//org.apache.fop.fo.flow.Block@194d372">
                            <DisplaySpace size="1200"/>
                            <LineArea font-weight="normal">
                              <WordArea font-weight="normal" red="0.0"
green="0.0" blue="0.0" width="106032">Architektur-
Patterns</WordArea>
                            </LineArea>
                            <DisplaySpace size="1200"/>
                          </BlockArea>
                        </AreaContainer>
                        <AreaContainer name="null">
                          <BlockArea start-indent="0" end-indent="0"
is-first="true" is-last="true" generated-
by="fo:block//org.apache.fop.fo.flow.Block@1c5fde0">
                            <DisplaySpace size="1200"/>
                            <LineArea font-weight="normal">
                              <WordArea font-weight="normal" red="0.0"
green="0.0" blue="0.0" width="6672">1</WordArea>
                            </LineArea>
                            <DisplaySpace size="1200"/>
                          </BlockArea>
                        </AreaContainer>
                        <AreaContainer name="null">
                          <BlockArea start-indent="0" end-indent="0"
is-first="true" is-last="true" generated-
by="fo:block//org.apache.fop.fo.flow.Block@19e8329">
                            <DisplaySpace size="1200"/>
                            <LineArea font-weight="normal">
                              <WordArea font-weight="normal" red="0.0"
green="0.0" blue="0.0" width="25332">EUR</WordArea>
                              <InlineSpace size="3336"/>
                              <WordArea font-weight="normal" red="0.0"
green="0.0" blue="0.0" width="36696">100.00</WordArea>
                            </LineArea>
                            <DisplaySpace size="1200"/>
                          </BlockArea>
                        </AreaContainer>
                        <AreaContainer name="null">
                          <BlockArea start-indent="0" end-indent="0"
is-first="true" is-last="true" generated-
by="fo:block//org.apache.fop.fo.flow.Block@16c163f">
                            <DisplaySpace size="1200"/>
                            <LineArea font-weight="normal">
```

```
                              <WordArea font-weight="normal" red="0.0"
green="0.0" blue="0.0" width="25332">EUR</WordArea>
                              <InlineSpace size="3336"/>
                              <WordArea font-weight="normal" red="0.0"
green="0.0" blue="0.0" width="20016">100</WordArea>
                            </LineArea>
                            <DisplaySpace size="1200"/>
                          </BlockArea>
                        </AreaContainer>
                      </AreaContainer>
                      <DisplaySpace size="13500"/>
                    </AreaContainer>
                  </AreaContainer>
                  <DisplaySpace size="24000"/>
                  <BlockArea start-indent="0" end-indent="0" is-
first="true" is-last="true" generated-
by="fo:block//org.apache.fop.fo.flow.Block@12884e0">
                    <DisplaySpace size="2000"/>
                    <LineArea font-weight="normal">
                      <WordArea font-weight="normal" red="0.0"
green="0.0" blue="0.0" width="29328">Tax:</WordArea>
                      <InlineSpace size="4000"/>
                      <WordArea font-weight="normal" red="0.0"
green="0.0" blue="0.0" width="32000">EUR</WordArea>
                      <InlineSpace size="4000"/>
                      <WordArea font-weight="normal" red="0.0"
green="0.0" blue="0.0" width="44000">192.60</WordArea>
                    </LineArea>
                    <DisplaySpace size="2000"/>
                  </BlockArea>
                  <BlockArea start-indent="0" end-indent="0" is-
first="true" is-last="true" generated-
by="fo:block//org.apache.fop.fo.flow.Block@12bcd4b">
                    <DisplaySpace size="2000"/>
                    <LineArea font-weight="normal">
                      <WordArea font-weight="normal" red="0.0"
green="0.0" blue="0.0" width="62240">Shipping:</WordArea>
                      <InlineSpace size="4000"/>
                      <WordArea font-weight="normal" red="0.0"
green="0.0" blue="0.0" width="32000">EUR</WordArea>
                      <InlineSpace size="4000"/>
                      <WordArea font-weight="normal" red="0.0"
green="0.0" blue="0.0" width="16000">10</WordArea>
                    </LineArea>
                    <DisplaySpace size="2000"/>
                  </BlockArea>
                  <BlockArea start-indent="0" end-indent="0" is-
first="true" is-last="true" generated-
by="fo:block//org.apache.fop.fo.flow.Block@b307f0">
                    <DisplaySpace size="2000"/>
                    <LineArea font-weight="normal">
                      <WordArea font-weight="normal" red="0.0"
green="0.0" blue="0.0" width="38224">Total:</WordArea>
                      <InlineSpace size="4000"/>
                      <WordArea font-weight="normal" red="0.0"
green="0.0" blue="0.0" width="32000">EUR</WordArea>
```

```
                    <InlineSpace size="4000"/>
                    <WordArea font-weight="normal" red="0.0"
green="0.0" blue="0.0" width="64000">202.60.00</WordArea>
                  </LineArea>
                  <DisplaySpace size="2000"/>
                </BlockArea>
              </AreaContainer>
            </SpanArea>
          </AreaContainer>
          <AreaContainer name="footnote-reference-area">
        </AreaContainer>
      </BodyAreaContainer>
    </Page>
</AreaTree>
```

wobei Apache FOP verwendet wurde, in XML Spy.

Und dier die Ausgabe als PDF:

Ship to:
Klaus Peters

Product	Quantity	Unit Price	Subtotal
Design Patterns	1	EUR 90	EUR 90
Architektur-Patterns	1	EUR 100.00	EUR 100

Tax: EUR 192.60
Shipping: EUR 10
Total: EUR 202.60.00

Sieht doch ganz nett aus!
Aber vergessen Sie nicht den Aufwand, das XSL zu definieren!

1.5. Zusammenfassung

XML ist die Standard Text-basierte Markup Sprache, mit deren Hilfe fast alle Datentypen beschrieben werden können, sowohl unstrukturierter Text, als auch Daten-orientierte Informationen. Weniger geeignet für XML sind Bitmaps und allgemein Bilder oder Ton, obschon sich XML auch auf diesen Gebieten versucht.

XML Dokumente bestehen auf der logischen Ebene aus verschachtelten Elementen. Jedes Element besitzt einen Namen, ein Set von Attributen und einen Inhalt.

Der Inhalt selber kann auch XML Elemente enthalten.

Die Attribute bestehen aus Namens/Wert Paaren.

Jedes Dokument besitzt genau eine Wurzel. Deswegen besitzt jedes XML Dokument eine Baumstruktur.

Neben Elementen und Attributen kann ein XML Dokument auch Kommentare, Verarbeitungsanweisungen, eine XML Deklaration und eine Dokumenten-Typ Definition sowie Textknoten umfassen.

Syntaktische werden Elemente durch Tags definiert (Start und Abschluss). Falls kein Inhalt vorhanden ist, kann eine Kurzversion eingesetzt werden: `<MeinLeeresElement/>`

Attribute entsprechen *name*="*value*" Paaren innerhalb Start-Tags und Leeren-Element Tags, wie in `<Menge number="17"/>`.

Physisch kann ein XML Dokument in Speichereinheiten aufgeteilt werden, Entities. Diese Entities können Dateien, Datenbank-Datensätze, Speicherstrukturen oder sonst was sein. Die Dokument-Entity enthält die Dokument-Wurzel.

Geparste Entities enthalten XML Markup, welcher zum Gesamt-XML Dokument zusammengefügt werden. Geparste Entities werden mithilfe von Entity-Referenzen lokalisiert, referenziert: `&arabiata;`.

Ungeparste Entities enthalten nicht-XML Daten, möglicherweise binäre Daten, deren Typus mit dem ENTITY Type Attributen im Dokument beschrieben werden müssen.

Jedes XML Dokument muss wohlgeformt sein, also zu jedem Start-Tag einen End-Tag besitzen, die Attribute müssen in Anführungszeichen stehen und dürfen lediglich Zeichen enthalten, die gemäss XML Spezifikation zugelassen sind.

XML Dokumente mit Schema oder DTD (Document Type Definitions) oder einem W3C XML Schema kann überprüft werden (Validation) und wird damit hoffentlich zu einem gültigen XML Dokument.

Die Formatierung von XML Dokumenten geschieht entweder mit CSS (Cascading Style Sheets) oder der Extensible Stylesheet Language (XSL). CSS ist nicht XML konform. XSL besteht aus den zwei Teilen XSL-FO Seitenbeschreibungssprache und der XSLT Turing-vollständigen funktionalen Sprache. XSLT gestattet die Transformation eines XML Dokuments in XSL-FO; XSLT kann auch benutzt werden, um daraus XHTML zu generieren.