

Namespaces in XML

<http://www.w3.org/TR/REC-xml-names>

- „Book“:


```
<book>
  <title> ... </title>
  <authors> ... </authors>
  <chapter> ... </chapter>
</book>
```
- „Person“


```
<person>
  <name> ... </name>
  <title> ... </title>
  <cv>
    <birthdate> ...
  </birthdate>
  ...
</cv>
</person>
```
- XHTML:


```
<html>
  <head>
    <title> ...</title>
  </head>
  <body>...</body>
</html>
```
- Fazit:
 - Die Namen von Elementen (und Attributen) von XML-Dokumenten vermischer Herkunft kollidieren leicht!
 - Es besteht Handlungsbedarf bei Mischung dieser Elemente sowie bei Verwechslungsgefahr

- Kein Problem, solange
 - jedes Dokument seine eigene DTD besitzt
 - diese DTD das Dokument auch begleitet
 - von mehreren DTDs genutzte Elemente und Attribute nicht kollidieren
 - Grenzen erreicht, wenn
 - mehrere DTD-Autoren kooperieren sollen
 - globale Attribute benötigt werden (z.B. XLink)
 - objektorientierte Ansätze abzubilden sind
- Beispiel:** Der Autor von „book.dtd“ möchte für Element „authors“ das Element „person“ erben:
- ```
<!ENTITY % person SYSTEM "person.ent">
%person;
<!ELEMENT book (title, authors, chapter+)>
<!ELEMENT title #PCDATA>
<!ELEMENT authors (person, affiliation)+ >
```
- Problem:** Kollision zwischen lokaler Deklaration von „title“ und „title“ aus person.ent!

- 1) Präfix-Vergabe für Elemente und Attribute
- Konvention: Doppelpunkt „:“ als Trennzeichen
  - Kompatibel mit XML 1.0-Regeln und DTDs  
Formal: *QNames* statt *Names*
  - Kompakt: Präfixwerte sind i.d.R. kurz
  - Flexibel: XML-Autoren können Präfixwerte frei vergeben
  - Dadurch: Schaffung disjunkter Namensräume!
- Beispiel:**
- ```
<bk:book>
  <bk:title> ... </bk:title>
  <bk:authors> <nm:person> <nm:name> ...
    <nm:title> ... </nm:title>
  </nm:name></nm:person></bk:authors>
  <bk:chapter> ... </bk:chapter>
</bk:book>
```

Der Lösungsansatz von XML namespaces

2) Folgeproblem: Nun sind Präfix-Kollisionen möglich!

Lösung:

- Identifikation jedes Präfix mit einer weltweit eindeutigen „ID“

3) Restproblem: Was sind geeignete „IDs“?

Kandidaten:

- SGML's Formal Public Identifiers (FPI)
Beispiel: "-//W3C//DTD HTML 4.0 Transitional//EN"
- Internet domain names (IDN)
Beispiel: fh-wiesbaden.de
- Universal Resource Identifiers (URI), bestehend aus:
 - Universal Resource Locators (URL)
`http://www.informatik.fh-wiesbaden.de/~werntges/namespaces/sample1`
 - Universal Resource Names (URN)
`urn:IDN fh-wiesbaden.de:fbi-werntges-sample2`

Der Lösungsansatz von XML namespaces

• Vor- und Nachteile der Alternativen

- FPI
Eingeführt zum Zweck der Ressourcenverwaltung, nicht nur zur reinen Identifikation
Offizielle Registrierung ist nicht verbreitet,
Globale, intensive Nutzung nicht registrierter FPI birgt Kollisionsgefahr
- IDN
Teuer, umständlich und zeitaufwändig in der Anschaffung
Es gibt viel mehr Bedarf an IDs als IDNs
- URL
Eingeführt zum Zweck der Ressourcenverwaltung, nicht nur zur reinen Identifikation
Weit verbreitet, leicht zu verstehen und zu verwenden
- URN
Konzeptionell genau zum Zweck der Identifikation eingeführt
Noch wenig verbreitet / wenig bekannt

Der Lösungsansatz von XML namespaces

• Die pragmatische Lösung: „Virtuelle“ URL

- Man verwendet die URL-Notation zur Vergabe von IDs
- Globale Eindeutigkeit geregelt durch
 - a) globale Sicherung durch Nutzung der IDN-Verwaltung (DNS, ICANN)
 - b) lokale Sicherung der Eindeutigkeit durch IDN-Besitzer.
- Generell sind alle URI - also auch URN - verwendbar

• ACHTUNG:

- URL, die im Kontext von XML Namespace verwendet werden, sind zunächst reine Namen. Im Gegensatz zu normalen URL befinden sich dahinter keine (z.B. per Browser ladbaren) Dokumente!
- Allerdings hindert niemand die Urheber von Namespace URL daran, tatsächlich Dokumente unter diesen URL bereitzustellen.

Der Lösungsansatz von XML namespaces

4) Einbettung der Präfix-Zuordnungen in XML 1.0

- per Konvention sowie
- per Einführung des globalen Attributs / Präfix-Wertes „xmlns:“

Beispiel (vollständig, wohlgeformt, XML namespace-konform):

```
<bk:book
  xmlns:bk="http://www.mybooks.net/ids2002"
  xmlns:nm="http://www.other-authors.org/names2002">
  <bk:title> ... </bk:title>
  <bk:authors> <nm:person> <nm:name> ...
    <nm:title> ... </nm:title>
  </nm:name></nm:person></bk:authors>
  <bk:chapter> ... </bk:chapter>
</bk:book>
```

XML namespaces: Scoping

- **Vererbung** der Attribute „xmlns:“ bzw. „xmlns:prefix“:
 - Die Wirkung von *namespace*-Deklarationen in einem Element vererbt sich an alle Unter-Elemente.
 - Eine Deklaration in einem Unter-Element überschreibt die ererbte und vererbt sich wiederum an dessen Unter-Elemente.
 - Analogie zu „xml:lang“ und xml:space“
 - Die Vererbung erfolgt einfach aufgrund der Elementschachtelung im Dokument - eine DTD ist dazu nicht notwendig.
- **Beispiel 1:**
 - Siehe Bsp. zu Punkt (4)
 - Präfix-Werte „bk“ und „nm“ werden im *root*-Element „book“ deklariert, aber auch in den Unter-Elementen verwendet.

XML namespaces: Scoping

- **Beispiel 2: Lokale scope-Änderung**

```
<bk:book xmlns:bk="http://www.mybooks.net/ids2002">
  <bk:title> ... </bk:title>
  <bk:authors><bk:person <!-- hier scope-Wechsel von bk -->
    xmlns:bk="http://www.other-authors.org/names2002">
  <bk:name> ...
    <bk:title> ... </bk:title>
  </bk:name></bk:person> <!-- hier scope-Wechsel von bk -->
</bk:authors>
<bk:chapter> ... </bk:chapter>
</bk:book>
```
- **Vorsicht:**
 - Zulässig, aber verwirrend.
 - Kein empfohlener Stil!

Der default namespace

- Diese Konvention zur weiteren Vereinfachung des *markup* vermeidet die Verwendung von Präfixwerten, typischerweise bei den am häufigsten verwendeten Elementen.

Beispiel

```
<book
  xmlns="http://www.mybooks.net/ids2002"
  xmlns:nm="http://www.other-authors.org/names2002"
>
  <title> ... </title>
  <authors> <nm:person> <nm:name> ... <!-- hier Präfix! -->
    <nm:title> ... </nm:title> <!-- hier Präfix! -->
  </nm:name></nm:person></authors> <!-- hier Präfix! -->
  <chapter> ... </chapter>
</book>
```

default namespace und scoping

- Auch der *default namespace* vererbt sich und kann lokal überschrieben werden. Im folgenden Beispiel werden so alle Präfixes vermieden.

Beispiel

```
<book xmlns="http://www.mybooks.net/ids2002">
  <title> ... </title>
  <authors>
    <person <!-- Neuer Namespace-Kontext -->
      xmlns="http://www.other-authors.org/names2002" >
    <name> ... <title> ... </title>
  </name></person>
  </authors> <!-- Wieder alter Namespace-Kontext -->
  <chapter> ... </chapter>
</book>
```

- **Vorsicht** - auch dies ist leicht verwirrend!

Leerer *default namespace*

- Als *default namespace* kann auch der leere String vergeben werden:

Beispiel

```
<book xmlns="http://www.mybooks.net/ids2002">
  <title> ... </title>
  <authors>
    <person xmlns="" > <!-- Leerer Namespace-Kontext -->
    <name> ... <title> ... </title>
  </name></person>
  </authors>      <!-- Wieder alter Namespace-Kontext -->
  <chapter> ... </chapter>
</book>
```

- Vorsicht - einfach, aber verwirrend und auch kollisionsgefährdet!

XML Prozessoren und *namespaces*

- Unterscheide XML-Prozessoren mit und ohne *namespace*-Unterstützung!
- Verhalten ohne Unterstützung:
 - Präfixwerte werden einfach als Namensteile behandelt
 - Default*-Regelungen und *scope*-Wechsel wirken nicht
- Verhalten mit Unterstützung:
 - QNames*/Präfixwerte werden intern expandiert in „*fully-qualified names*“ und dann erst verarbeitet, Defaultregeln werden dabei beachtet
 - Verschiedene Präfixwerte, die auf denselben URI verweisen, wählen denselben Namensraum aus.
- (Hypothetische) *fully-qualified names*:
 - `<bk:title>` wird zu `<{http://www.mybooks.net/ids2002}title>`,
 - `</nm:title>` wird zu `</{http://www.other-authors.org/names2002}title>`
 - `<title>` wird zu `<{}title>` (Im Kontext des leeren Namensraums)
 - Bemerkung: Die {...} sind keine gültigen Teile von XML 1.0 *Names* - sie erläutern hier nur die Expansion.

Attribute und *namespaces*

- Generell gilt:
 - Attribute gehören nicht direkt in das *Namespace*-Konzept. Sie sind ihren Elementen zugeordnet und dadurch indirekt einem Namensraum.
- Allerdings lassen die Spezifikationen eine Grauzone zu:
 - Beispiel: Sind folgende beiden Fragmente gleichbedeutend?
 - `<a:name id="25">`
 - `<a:name a:id="25">`
 - Leider bleibt die Entscheidung den Anwendungen überlassen!**
 - Während die meisten Anwendungen hier nicht unterscheiden, tut dies XSLT sehr wohl.
- Ausnahme: **Globale Attribute**
 - Bestimmte „globale Attribute“ lassen sich „importieren“ & nutzen
 - Ihr Namensraum wird dann explizit angegeben und unterscheidet sich i.d.R. von dem der lokalen Attribute des Elements.

Globale Attribute und *namespaces*

- Beispiel: XLink
 - Die *XML Linking Language* (XLink) verwendet globale Attribute, die per *namespace*-Deklaration angemeldet werden und dann an „beliebigen“ Stellen das Anlegen von *links* gestatten.

```
<mydoc xmlns:xlink="http://www.w3.org/1999/xlink">
  ...
  <citation
    xlink:type="simple"
    xlink:href="http://www.uw.ca/paper_on_xxx.xml">
    Biemanns (1997)
  </citation>
  ...
</mydoc>
```

DTD und namespaces

- Hintergrund
 - XML 1.0 und DTD gab es vor der *namespace*-Konvention.
 - Eine Unterscheidung *Names* - *QNames* kennt die DTD nicht.
 - Aus DTD-Sicht sind Präfixwerte und der Doppelpunkt einfach Teile der Element- bzw. Attributnamen - und müssen somit explizit deklariert werden.
- Vorgehen bei Element-Deklarationen
 - Die Elemente werden so deklariert, wie sie im XML-Quelltext erscheinen - incl. Präfix sofern vorhanden:

```
<!ELEMENT myns:myelement
  (#PCDATA, (myns:sub1, myns:sub2)*)>
<!ELEMENT myns:sub1 (...)>  <!-- usw. -->
```

DTD und namespaces

- Vorgehen bei Attributdeklarationen
 - Normale Attribute werden ohne Präfix verwendet und demnach auch ohne Präfix deklariert.
 - Verweise auf ihre Elementnamen enthalten ggf. ein Präfix.
 - Die Verwendung der Attribute „**xmlns**“ und „**xmlns:prefix**“ ist zu **deklarieren** - wie gewohnt.
 - **Konvention:** *Namespace*-URI sollten dabei per Attribut-*Default* zugewiesen werden, und zwar **#FIXED**:

```
<!ATTLIST myns:myelement
  xmlns:myns CDATA #FIXED "http://www.mydomain.org/ns">
<!ATTLIST math      <!-- DTD und default namespace / scoping! -->
  xmlns CDATA #FIXED "http://www.w3.org/TR/REC-MathML/">
```
 - Diese Konvention sollte man so bindend wie einen Standard betrachten. Einige Produkte, z.B. IE5, fordern dies bereits!
 - Der doppelte Pflegeaufwand garantiert sowohl die DTD-Validierung als auch die Verträglichkeit mit *namespace*-kompatiblen Produkten.

Stilfragen bei der Präfixvergabe

- **Vermeiden:**
 - Re-Deklarationen von Präfixwerten
 - Nutzung des leeren Namensraums
 - Präfix vor Attributen
 - Ausnahme: Gezielt für Globale Attribute
- **Anstreben:**
 - Kurze Präfixwerte - *markup* wird überschaubarer
 - Einhaltung verbreiteter Präfix-Konventionen
 - html, xlink, xsd, xsi, xsl, fo, ...
 - Regel: gleicher Namensraum - gleiches Präfix
 - Sparsamer Gebrauch von *default*-Regeln
 - DTD-Konformität (daraus folgern bereits andere Ziele)
 - Wiederverwendbarkeit in „fremdem“ Kontext

Nachwort zu XML namespaces

- Das Konzept ist durchaus umstritten
 - Markup wird unübersichtlicher, DTDs werden komplizierter.
 - *Defaults* und *Scope*-Änderungen können verwirren.
 - Die Empfehlungen könnten an einigen Stellen stärker einschränken.
- Gründe für die inzwischen weite Verbreitung
 - Haupteinsatzgebiet: Integration von Daten aus verschiedenen Quellen.
 - Beispiel: XHTML + SVG + MathML
 - Solange meist Anwendungen - und nicht Menschen - in direktem Kontakt mit diesem *markup* kommen, stört's nicht.
 - DTD-Validierung entfällt dabei oft bzw. wird ersetzt durch Schema-basierte Ansätze - die *namespaces* verwenden.
 - Das Konzept passt insbesondere hervorragend zu XSL(T). Steuernder *markup* lässt sich dank Präfixregelung leicht von zu erzeugendem *markup* unterscheiden.



Beispiel: XML namespaces für XHTML + SVG



```
<?xml version="1.0" encoding="iso-8859-1" ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html
  xmlns="http://www.w3.org/1999/xhtml" xml:lang="de"
  xmlns:svg="http://www.w3.org/2000/svg">
  <head>
    <title>Text z.B. für den Fensterbalken</title>
  </head>
  <body>
    <p>
      SVG Quellcode, im XHTML-Quellcode eingebettet:
    </p>
    <svg:svg width="280" height="280">
      <!-- SVG-Inhalt! -->
    </svg:svg>
  </body>
</html>
```