

Hinweis: um keinen Platz für eine Beispiel-Quelldatei verschwenden zu müssen, dient hier eine imaginäre (X)HTML-Datei als Quelle für alle Beispiele.

xPath 1.0

Einfache Pfade

xPath Pfade dienen dazu, Elemente oder Attribute (Knoten) in einer XML Datei zu adressieren. In der einfachsten Form, kann das so aussehen::

```
//p // alle (!) <p>-Tags im Dokument  
/html /head /title // Das Element <title> unter <head> unter <html>
```

Grundsätzlich werden immer *alle* Knoten ausgewählt, für die der Pfad gültig ist.

Achsen

Das Beispiel von oben könnte man auch ausführlicher schreiben als:

```
/child::html /child::head /child::title
```

Mit anderen Worten: der Pfad folgte der Nachfolgerachse („child“) bis zu den <p>-Elementen. Daneben gibt es auch noch weitere Achsen:

Bezeichnung	Beschreibung
self	Der aktuelle Knoten selbst (Kurzform: „.“)
child	Nachfahren (kann weggelassen werden)
attribute	Attribute des Elementes (Kurzform: „@“)
following	nachfolgend definierte Elemente
following-sibling	nachfolgende Elemente auf gleicher Ebene
preceding	vorher definierte Elemente
preceding-sibling	vorherige Elemente auf gleicher Ebene
ancestor	Beliebiger Vorgänger (auch mehrere Ebenen höher)
ancestor-or-self	Beliebiger Vorgänger oder auch der aktuelle Knoten selbst
descendant	Beliebiger Nachfahre (auch mehrere Ebenen tiefer)
descendant-or-self	Beliebiger Nachfahre oder auch der aktuelle Knoten („//“)

Ein Beispiel für die Verwendung der „self“-Achse (hier abgekürzt mit „.“)

```
//a/@name[.="toc"] // Alle 'name'-Attribute in <a>, die "toc" sind
```

Mehrere zu findende Werte können mittels Pipe-Symbol („|“) getrennt werden:

```
/html /body /p | /html /body /h1 // alle <p>, und alle <h1>
```

Der Stern („*“) kann als Wildcard für beliebige Namen verwendet werden.

```
/html /body /* // alle Elemente unter /html /body /  
/html /body /p /@* // alle Attribute aller <p>
```

Bedingungen

Um nicht nur bestimmte Knoten zu selektieren, können Bedingungen angegeben werden:

```
//p[2]           // das zweite <p>-Element im Dokument
//a[@name]      // alle <a>-Elemente mit ,name'-Attribut
```

Daneben können auch komplexere Abfragen integriert werden, z.B.:

```
/html /body/p[positi on()=2] // das <p> an zweiter Position
/html /body/p[positi on()<3] // die ersten zwei <p>
/html /body/p[la st()]       // das <p> an letzter Position
/html /body/p[la st()-1]    // das vorletzte <p>
/html /body/p[@cl ass="footer"] // alle <p>, die ein Attribut „class“ mit
                               Inhalt „footer“ haben
```

Einige weitere wichtige Funktionen:

count (nodeset)	Liefert die Anzahl der Elemente in dem Nodeset
concat (s1, s2, sn*)	Verbindet die Parameter zu einem neuen String
starts-wi th (s1, s2)	Liefert ‚true‘, wenn der String s1 mit s2 anfängt
contai ns (s1, s2)	Liefert ‚true‘, der String s1 irgendwo s2 enthält
i d (name)	Liefert das Element mit der eindeutigen ID

Für Vergleichsoperationen werden Operatoren im C-Stil verwendet, also =, !=, <, <=, > und >=. Achtung, erscheint der XPath-Ausdruck innerhalb eines anderen XML-Dokumentes, muss zumindest das Kleiner-als-Zeichen als Entität („<“) kodiert werden.

Mehrere Bedingungen können mittels „or“ bzw. „and“ verbunden werden:

```
//p[@cl ass="a" or @cl ass="b"] // alle <p> mit Klasse „a“ oder „b“
```

Dagegen ist die *Negation* eine Funktion, die den zu negierenden Ausdruck umschließt:

```
//p[not(@cl ass="footer")]
```

Ebenso sind „true()“ und „fal se()“ Funktionen, welche die entsprechenden Wahrheitswerte zurückgeben. Sie müssen daher immer mit Klammern geschrieben werden.

Kontext

Innerhalb eines XPath-Pfades wird der Kontext immer auf den jeweils aktuellen Pfadbestandteil gesetzt. Dadurch bezieht sich z.B. ein Pfad in einer Bedingung auf das jeweils davor stehenden Pfadelement (siehe Beispiele oben).

Um den Kontext (temporär) zurück zu setzen, muss der Teilausdruck mit einem oder zwei Schrägstrichen („/“ bzw. „//“) oder beginnen.

```
//a[@href=//li nk/@href] // Alle <a>, mit identischem ,href‘ wie ein <li nk>
```

Umgekehrt kann man den „descendant-or-sel f“-Operator („//“) explizit vom aktuellen Knoten anfangen lassen, indem man den „sel f“-Operator („.“) davor stellt:

```
//ul [contai ns(./a/@href, "#")] //, alle <ul>, die einen lokalen Li nk enthal ten
```

xPath 2.0

Schleifen

Zu den in Version 2.0 hinzu gekommenen Funktionalitäten gehört die Möglichkeit, Schleifen zu definieren:

```
for $a in //ul/li return $a/text()
```

Durch die Schleifenkonstruktion ändert sich der Kontext nicht, stattdessen dient die Variable (hier \$a) als Referenz auf den aktuellen Knoten

Die Schleife gibt eine Sequenz zurück, auf die wiederum weitere Funktionen angewandt werden können.

Bedingungen

Die Abfrage von Bedingungen kann nun auch ganz „klassisch“ mittels „if...then“ erfolgen. Dies macht wahrscheinlich am meisten Sinn in Verbindung mit einer „for“-Schleife:

```
for $a in //ul/li return
  (if (contains($a/a/text(), "XHTML")) then "OK" else "Nope")
```

Auch „if“ ändert nicht den Kontext für die darin enthaltenen XPath-Ausdrücke.

some und every

Die Befehle „some“ und „every“ testen ob eine Bedingung für die Nodes in einer Sequenz gilt. Dabei gibt „every“ nur dann ‚true‘ zurück, wenn die Bedingung für *alle* Nodes gilt, bei „some“ genügt es, wenn mindestens ein Node die Bedingung erfüllt (Existenztest):

```
every $a in //ul/li satisfies $a/@class="toc"
some $a in //head/meta satisfies $a/@name="description"
```

Dabei können die Ausdrücke auch mehrere Bestandteile haben:

```
some $a in (1, 2, 3), $b in (4, 5, 6) satisfies $a * $b = 15
```

Mengenoperationen

Mittels der Operationen „union“, „intersect“ und „except“ können die Sequenzen wie Mengen behandelt werden:

```
//ul/li union //ol/li
//p[@align="left"] intersect //p[@class="wichtig"]
//ol/li[@class="toc"] except //ol/li[@id="toc211"]
```

Dabei schließt „except“ nur Elemente aus, die wirklich identisch sind, d.h. für die eine „is“-Beziehung besteht.