universität wien

Faculty of Computer Science
Workflow Systems and Technology Group

XPath – XML Path Language

Juergen Mangler
University of Vienna

XPath

XML Path Language (XPath) Version 1.0
http://www.w3.org/TR/xpath

Wird verwendet um Teile eines XML Dokuments zu adressieren, z.B. in XSLT, XPointer, Schema und in XML Anwendungen

Verwendet kompakte nicht-XML Syntax um auch in URIs und XML Attributen verwendet werden zu können

Operiert auf der logischen Struktur des XML Dokuments

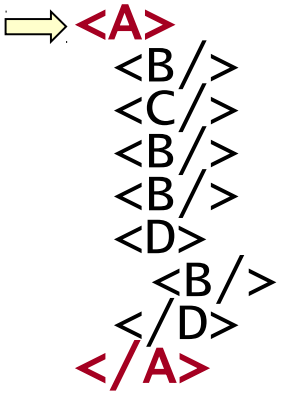
Der Name kommt von der Verwendung einer pfadähnlichen Notation (wie etwa in Dateisystemen) für die hierarchische Struktur eines XML Dokuments

Modelliert ein XML Dokument als einen Baum aus Knoten. Es gibt unterschiedliche Knotenarten, wie Elementknoten, Attributknoten, Textknoten, Kommentarknoten, etc.

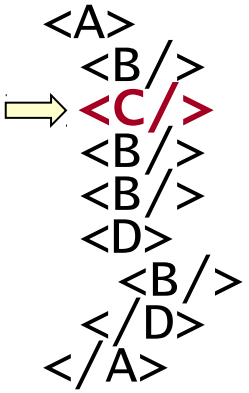
Adressierung: /

XPath Syntax ist ähnlich wie Adressierung in Dateisystemen. Wenn der Pfad mit einem Slash / startet, dann ist dies ein absoluter Pfad (beginnend von der Wurzel). Die Navigation erfolgt schrittweise.

/A
Das Wurzelelement



/A/C
Alle Elemente C, die Kinder des Wurzelelements A sind



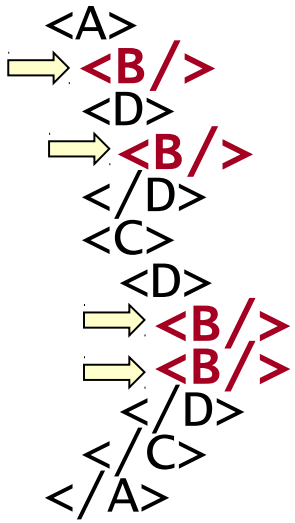
Schrittweises adressieren indem Pfad durch Slashes getrennt wird

Anweisung: *"Gehe zu Wurzelelement A, dann zu allen Kindern C"*

Adressierung: //

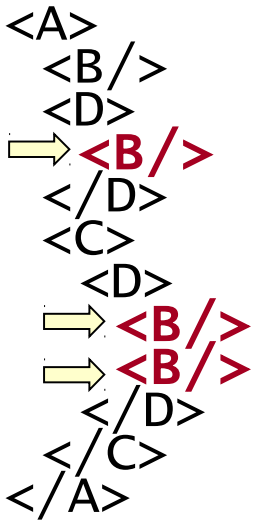
Wenn der Pfad mit // startet, dann werden **alle** Knoten im Dokument, für die der Pfad nach dem // zutrifft, selektiert

//B
Alle Elemente B



*Ergebnis eines solchen XPath Ausdruckes ist eine Knoten**menge**!*

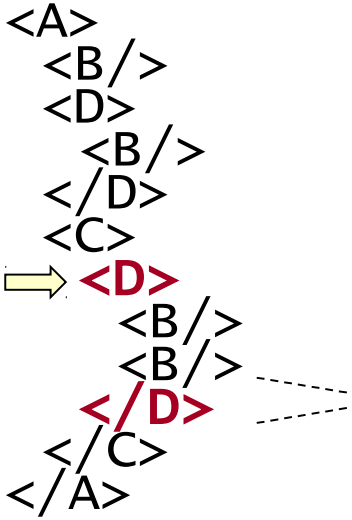
//D/B
Alle Elemente B die Kinder eines Elements D sind



Adressierung: *

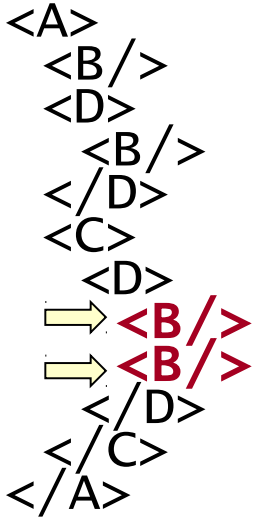
Ein Stern * selektiert alle Knoten auf der Ebene des (bisherigen) Pfades

/A/C/*
Alle Elemente auf der Ebene unterhalb der Hierarchie /A/C



/A/C//* würde auch diese beiden B's selektieren

/*/*/*/B
Alle Elemente B in der dritten Ebene unterhalb des Wurzelknotens

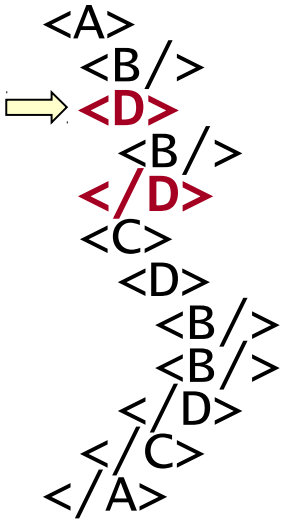


Prädikate: Position

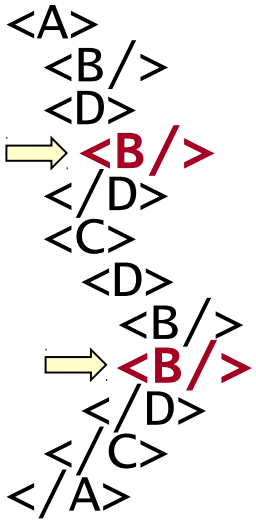
Ein Ausdruck in eckigen Klammern gibt eine Bedingung (Prädikat) an, auf die die Ergebnisknotenmenge des bisherigen Pfades getestet werden soll

- Eine Zahl (bzw. ein Ausdruck der zu einer Zahl evaluiert) bezieht sich auf die Position eines Knotens in einem best. Pfad
- Funktion last() liefert die Position des letzten Knotens in einem best. Pfad

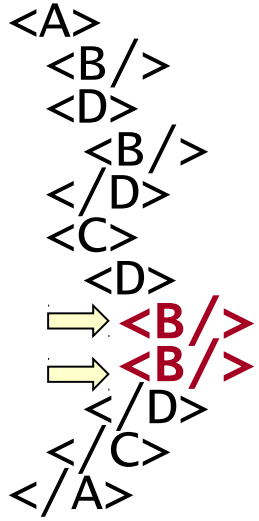
`/A/*[2]`
Das zweite Kind von A



`//B[last()]`
Jedes letzte B



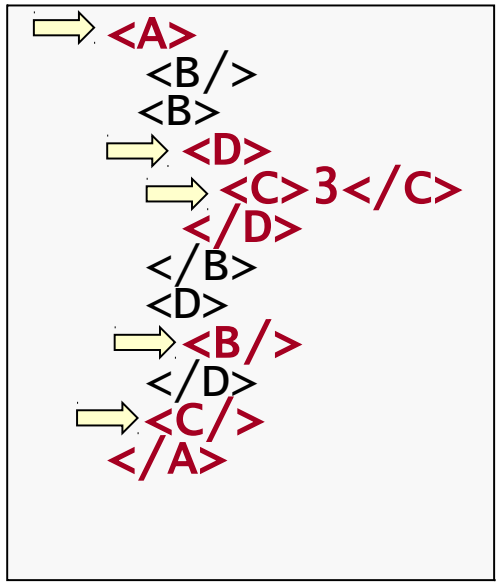
`//D[last()]/*`
Alle Kinder eines letzten D



Prädikate: Logische Audrücke

Prädikat kann auch ein logischer Ausdruck sein

- Verknüpfung möglich mit **or** (logisches oder) bzw. **and** (logisches und)
- Vergleichsoperationen: = (gleich), **!**= (ungleich), >=, >, <, <=
- Als Operanden können ganze XPathes benutzt werden
- Auch Schachtelung möglich



`/A[B = 'Hugo']`
Alle A, die B Kinder mit Text "Hugo" haben

`//*[D[C > 25] or position()=last()]`
Alle Knoten die ein Kind D haben (die wiederum ein Kind C mit Wert grösser 25 haben), oder Elemente an letzter Position

`/A[B/C != 'Hugo']`
Alle A, mit Kindern B die Kinder C haben mit Text ungleich "Hugo"

`//D[B and not(C)]`
Alle D, die B Kinder, aber keine C Kinder haben

Auf Attribute kann man mit @ zugreifen

`//@*`
Alle Attribute irgendwo im Dokument

`//@matnr`
Alle Attribute matnr

`//student[@matnr]`
Alle Studenten die ein Attribut matnr haben

`//student/@name`
Alle Attribute name von Studenten

`//student[@*]`
Alle Studenten die irgendein Attribut haben

`//student[not(@*)]`
Alle Studenten, die gar kein Attribut haben

`//student[@name or @matnr='9506264']`
Alle Studenten die ein Attribut name haben oder Matrikelnummer 9506264

```
<students>
  <student matnr="9506264"/>
  <student matnr="0002843"/>
  <student name="Max"/>
  <student/>
</students>
```

Komplexere Ausdrücke in Prädikaten

In Prädikaten können als Operanden ganze XPathes verwendet werden

Beispiel: Spionagedatenbank
Info: `//detail/@pers` referenziert `//person/@id`

```
<spionagedaten>
  <personen>
    <person id="1" name="Juan Carlos"/>
    <person id="2" name="Osama bin Laden"/>
    <person id="3" name="James J. Bulger"/>
  </personen>
  <details>
    <detail pers="1" typ="fahrzeug">Mercedes</detail>
    <detail pers="3" typ="konfession">röm.kath.</detail>
    <detail pers="2" typ="waffe">StG 77</detail>
    <detail pers="3" typ="verheiratet">ja</detail>
  </details>
</spionagedaten>
```

Wir wollen nun z.B. wissen ob James J. Bulger verheiratet ist:

```
//detail[ @typ = 'verheiratet' and
          @pers = (//person[@name='James J. Bulger']/@id) ]
```

Kontextknoten und Achsen

universität wien

Kontextknoten bezeichnet den "aktuellen" Knoten

XPath Ausdrücke werden immer auf einen Kontextknoten angewandt (Dokument oder bestimmter Knoten innerhalb des Dokuments)

Für relative Pfade gilt der Kontextknoten als Ausgangspunkt

In Prädikaten kann man sich jeweils auf Kontextknoten beziehen

Implizit: `//*[name()='A']` ... Hier wird die Funktion name() auf den jeweiligen Kontext (=Pfad vor der eckigen Klammer) angewandt

Explizit: mit einem Punkt bezieht man sich auf den Kontextknoten:
`//*[.='Hugo']`

`//*[B='Hugo']` ist beispielsweise äquivalent zu `//*[./B='Hugo']`

In XPath "navigiert" man entlang von Achsen

"Richtungen" die aus dem Kontextknoten hinausführen

Knotenmengen relativ zum Kontextknoten

Navigation: . und ..

Mit einem Punkt bezieht man sich auf den Kontextknoten (s. vorige Folie)

Mit zwei Punkten navigiert man eine Ebene nach "oben"

Beispiele für eine Kundenkartei. Relevanter DTD Ausschnitt:

```
<!ELEMENT kunden (kunde*)>
<!ELEMENT kunde (adresse+)>
<!ATTLIST kunde name CDATA #REQUIRED titel CDATA #IMPLIED>
<!ATTLIST adresse str CDATA #REQUIRED ort CDATA #REQUIRED plz CDATA #REQUIRED>
...
```

`/kunden/kunde/adresse[../@name='Tom Turbo']`

Hier wird das Attribut name eines Kunden getestet! Ist also äquivalent zu **`/kunden/kunde[@name='Tom Turbo']/adresse`**

`//adresse[@ort='wien']/..`

Alle Kunden die in Wien wohnen

Achsen

Es gibt einige vordefinierte Achsen, die man in XPath verwenden kann um in Relation zu einem Knoten zu navigieren. Verwendung z.B.:

`//B/ancestor::C` ... Alle Vorfahren C von allen B

`//C/descendant::*/D` ... Alle Kinder D von Nachfahren von C

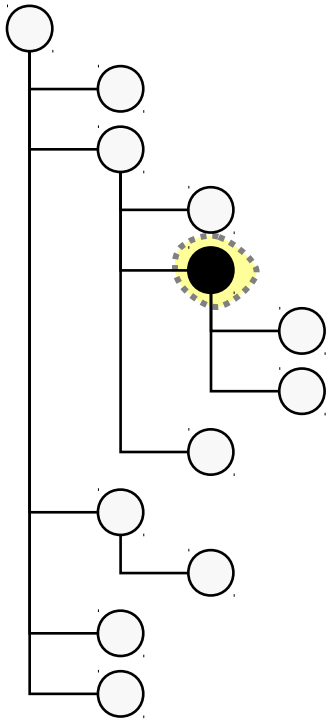
Folgende Achsen sind in XPath definiert:

ancestor (Vorfahren)
ancestor-or-self (Vorfahren oder selbst)
child (Kinder)
descendant (Nachfahren)
descendant-or-self (Nachfahren oder selbst)
following (Folgende)
following-sibling (Folgende Geschwister)
parent (Eltern)
preceding (Vorangehende)
preceding-sibling (Vorangehende Geschwister)
self (Selbst)
attribute
Namespace

Achsen: Überblick

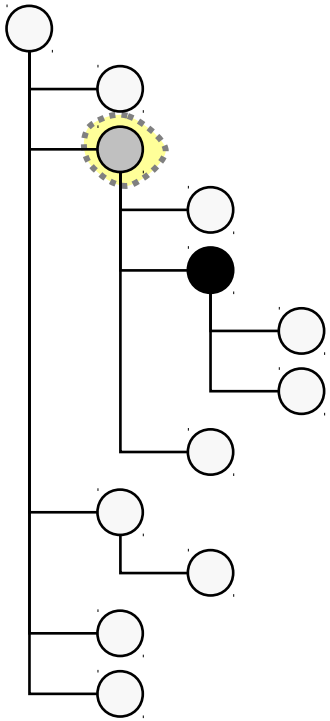
...Kontextknoten

self
(Kontextknoten)



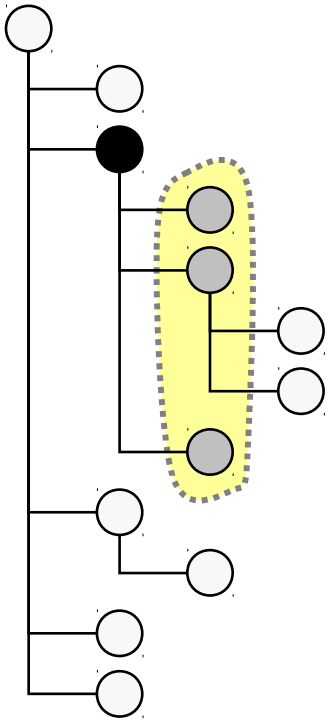
`self::node()`

parent
(Eltern)



`parent::node()`

child
(Kinder)

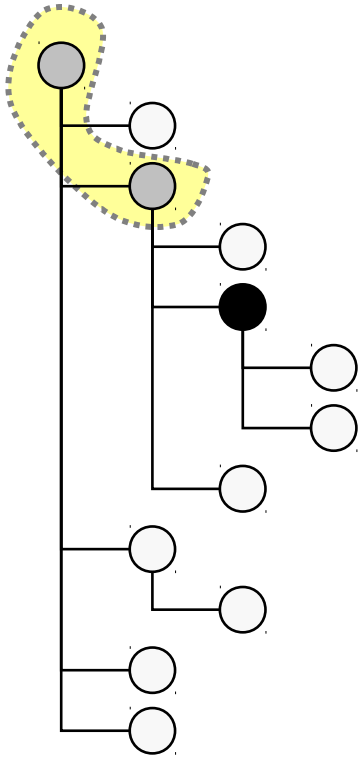


`child::*`

Achsen: Überblick

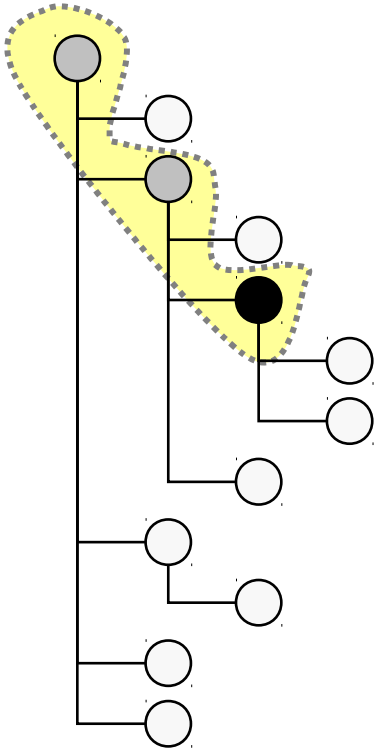
...Kontextknoten

ancestor
(Vorfahren)



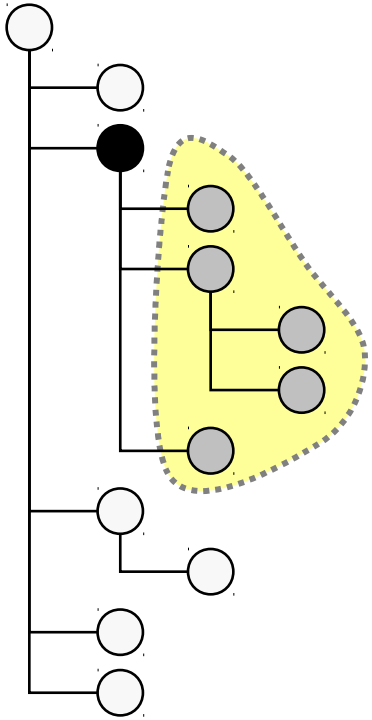
`ancestor::*`

ancestor-or-self
(Vorfahren oder selbst)



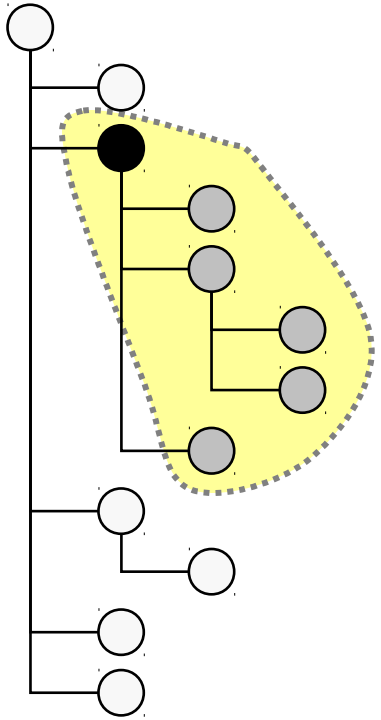
`ancestor-or-self::*`

descendant
(Nachfahren)



`descendant::*`

descendant-or-self
(Nachfahren oder selbst)

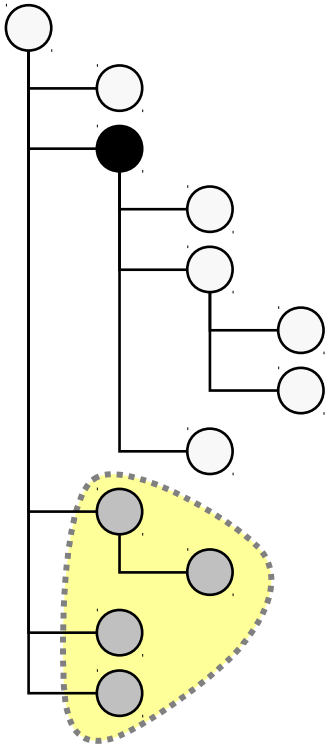


`descendant-or-self::*`

Achsen: Überblick

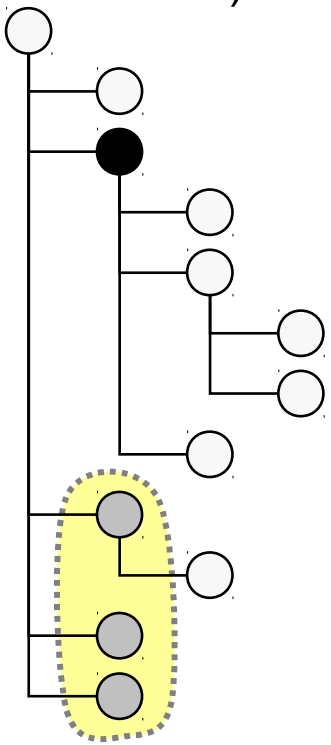
...Kontextknoten

following
(Folgende)



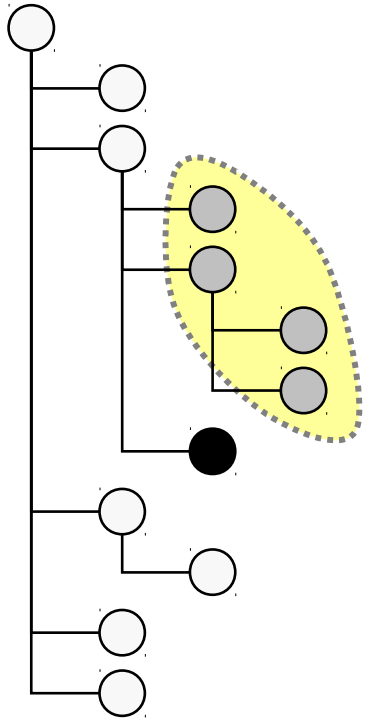
`following::*`

following-sibling
(Folgende Geschwister)



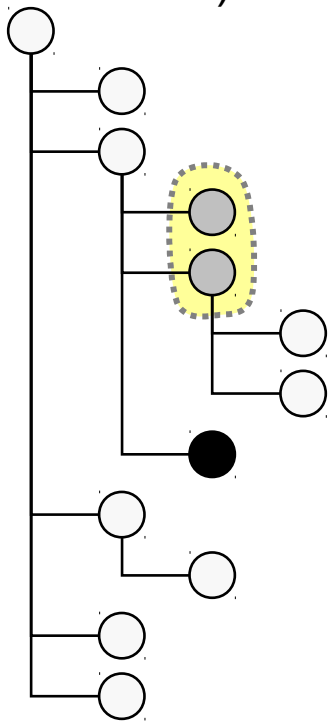
`following-sibling::*`

preceding
(Vorangehende)



`preceding::*`

preceding-sibling
(Vorangehende Geschwister)



`preceding-sibling::*`

Achsen: Kurzformen

(**child** ist die Standardachse, die nicht angegeben werden muss!)

`/A`
ist die Kurzform von
`/child::A`

`//A`
ist die Kurzform von
`/descendant-or-self::node()/child::A`

node() testet auf Knoten

`.`
ist die Kurzform von
`self::node()`

`..`
ist die Kurzform von
`parent::node()`

`@a`
ist die Kurzform von
`attribute::a`

Kombinationsbeispiele für Achsen-Kurzformen

`/A//*/.`

ist die Kurzform von

`/child::A/descendant-or-self::node()/*/self::node()`

`//A[@x='4']/..`

ist die Kurzform von

`/descendant-or-self::node()/child::A[attribute::x='4']/parent::node()`

`//D/B/following-sibling::C/attribute::x`

Alle Attribute x von einem C, das ein folgendes Geschwister von einem B ist, das wiederum Kind irgendeines D ist

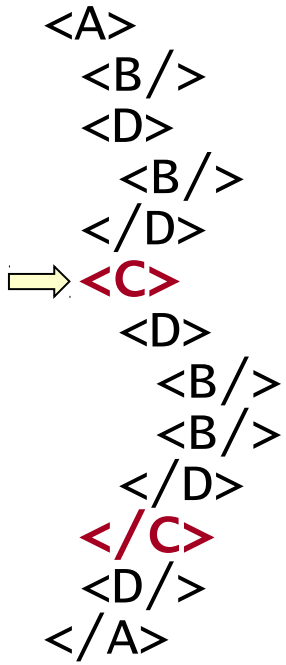
Achsen: Reihenfolge

Achtung: Bei den **preceding** und **preceding-sibling** Achsen ist sind die Positionen der Knoten in *umgekehrter* Dokumentreihenfolge zu sehen

Beispiel:

```
/A/*[4]/preceding-sibling::*[1]
```

Das erste vorangehende Geschwister des vierten Kindes des Wurzelelements



Anders bei dieser Klammerung:

```
(/A/*[4]/preceding-sibling::*)[1]
```

Das erste von den vorangehenden Geschwistern des vierten Kindes des Wurzelelements

Vereinigung von Ergebnismengen

Mehrere XPathes können mit | kombiniert werden

Dadurch werden die Ergebnismengen der einzelnen XPathes *vereinigt* (Knoten werden nicht doppelt ins Ergebnis aufgenommen, auch wenn sie durch mehrere Teilausdrücke ausgewählt werden!)

(1) (2)

```
/A/C//* | //B[1]
```

<A>
(2) <B/>
<D x="5">
(2) <B/>
</D>
<C y="6">
(1) <D>
(1+2) <B/>
(1+2) <B/>
</D>
</C>
</A>

(1) (2) (3)

```
//B | //@*/.. | //*[2]
```

<A>
(1) <B/>
(2+3) <D x="5">
(1) <B/>
</D>
(2) <C y="6">
<D>
(1) <B/>
(1+2+3) <B
z="7" />
</D>
</C>
</A>

Funktionen für Knoten(-mengen)

Einige eingebaute Funktionen für Knoten(-mengen):

last() : *number*
Anzahl der Knoten im aktuellen Kontext
`//*[last()]`

position() : *number*
Position des Knoten
`//B[position()=3]`

count(*node-set*) : *number*
Anzahl der Knoten im node-set
`//*[count(B)=2]` ... Alle Elemente die zwei B Kinder haben
Weitere Bsp.: `//B[count(C/D)>5]` oder `//*[count(*)=0]`

id(*object*) : *node-set*
object kann eine Knotenmenge (node-set) oder String sein
`id("a1")/*` ... Alle Kinder eines Elements mit id "a1"

name(*node-set?*) : *string*
Wenn ein node-set übergeben wird: der Name des ersten Knoten in der Menge; wird nichts übergeben: den Namen des Kontextknoten
`//*[name()='C']` oder `name(/A/B/*)`

Funktionen für Wahrheitswerte

Einige eingebaute Funktionen für Wahrheitswerte:

boolean(*object*) : *boolean*

Eine Boolean Repräsentation des Objekts. Liefert **true** bei:

- Zahl die ungleich 0 ist
- Knotenmenge die nicht-leer ist
- String der mindestens ein Zeichen enthält

`//A[boolean(@x)]` ... Alle A die ein Attribut x haben; Kurzform: `//A[@x]`

not(*boolean*) : *boolean*

Liefert Negation des Arguments

`//A[not(@x)]` ... Alle A ohne Attribut x; Langform: `//A[not(boolean(@x))]`

Funktionen für Text

Einige eingebaute Funktionen für Text:

string(*node-set?*) : *string*
Liefert die Stringrepräsentation des ersten Knoten im Argument bzw. des Kontextknoten falls kein Knoten übergeben wird
`//@*[string()='Hugo']`

concat(*string*, *string*+) : *string*
Verknüpfung aller übergebenen Strings
`//student[concat('a',string(@matnr))='a0593856']`

starts-with(*string*, *string*) : *boolean*
Beginnt der erste übergebene String mit dem zweiten?
`starts-with("Hugo", "H")` ergibt `true`

contains(*string*, *string*) : *boolean*
Beinhaltet der erste String den zweiten?
`//student[contains(@name,"ich")]` ... Alle Studenten deren Name ein "ich" beinhaltet

substring(*string*, *number*, *number*) : *string*
Extrahiert Teilstring ab einer bestimmten Position (2. Argument) für eine bestimmte Länge (3. Argument)
`substring("123456", 3, 2)` liefert "34"