# *XML Information Set, XML Path Language (XPath)*

http://www.w3.org/TR/xml-infoset
http://www.w3.org/TR/xpath

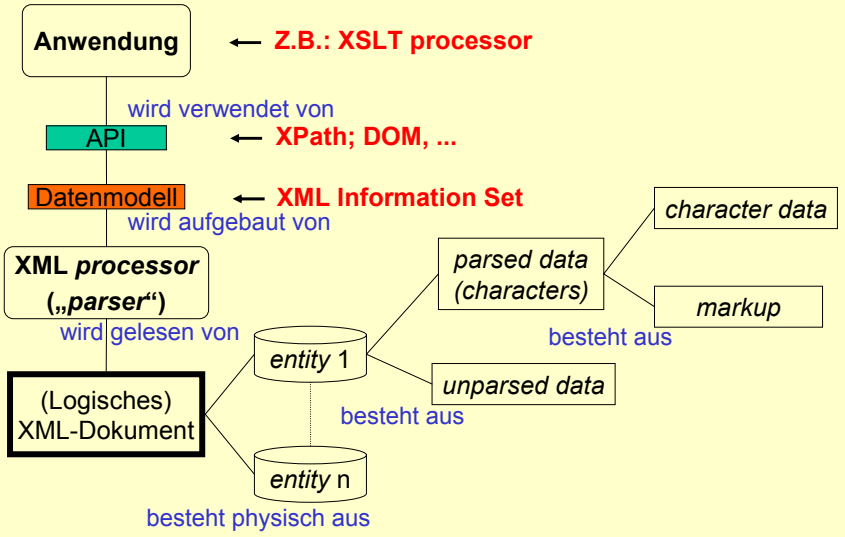




# XML Information Set

Übersicht
Begriffsbildung, Beispiel
*Item Properties*
Beispiel, Was nicht enthalten ist

# Einordnung: XML Infoset, XPath






# XML Infoset

- **Was ist XML Information Set?**
  - Die Spezifikation abstrakter Datenstrukturen, die den Anwendungen zugänglichen Inhalt von wohlgeformten (nicht unbedingt auch „gültigen“) XML-Dokumenten beschreiben.
  - Eine formale Beschreibung des „Outputs“ von XML-Prozessoren
  - Eine abstrakte Sicht, unabhängig von spezifischen APIs
  - Wie bei W3C üblich: Eine Empfehlung
- **Was ist ein XML Infoset?**
  - Meistens: Eine baumartige Repräsentation eines XML-Dokuments.
  - Auch: Eine künstlich generierte Datenstruktur ohne zugrundeliegendes XML-Dokument, die den Spezifikationen von XML Information Set genügt.
- **Was ist XML Infoset nicht?**
  - Erschöpfend (im Sinne aller Informationen eines XML-Dokuments)
  - Eine Minimalanforderung an den Output von XML-Prozessoren.

## XML Infoset

- **Begriffsbildung**
  - ***Information set*** (Informationssatz)
    - Entspricht einem Datenbaum (*tree*)
    - Enthält genau ein *document item* und weitere *information items*.
  - ***Information item*** (Ein Element des Infosatzes)
    - Entspricht einem Knoten (*node*) in diesem Baum
    - Enthält „Eigenschaften“ (*properties*), z.B. Listen weiterer *info items.*
  - Die Begriffe *tree* und *node* wurden vermieden, um zu betonen, dass Zugriffe auf den *information set* auch über andere Schnittstellen als baumartige erfolgen können, z.B. ereignisgesteuerte (*event-based*) oder anfrageartige (*query-based*).
- **Vorsicht:**
  - Elemente des Infosatzes lassen sich **nicht** 1:1 identifizieren mit Knoten des DOM oder Knoten bzw. Teilbäumen von XPath (obwohl dies auf weiten Strecken sehr wohl möglich ist).



## XML Infoset

- **Unterscheide folgende Werte von Eigenschaften voneinander:**
  - *„unbekannt“ (unknown)*
  - *„kein Wert“ (no value)*
  - Leere Zeichenkette / Liste / Menge
- Bemerkungen
  - Einem Attributwert kann die leere Zeichenkette zugewiesen werden. Dies ist ein anderer Inhalt als das Ausbleiben jeglicher Zuweisung!
  - In anderen Notationen werden die Schlüsselwörter „null“ oder „nil“ verwendet. Dies vermeidet die XML Infoset-Spezifikation, um Verwechslungen vorzubeugen, jedoch besteht konzeptionell eine enge Verwandtschaft.

## XML Infoset: *Information items*

- **Übersicht: Typen von *Information Items***
  - *Document*
  - *Element*
  - *Attribute*
  - *Unexpanded entity reference*
  - *Character*
  - *Comment*
  - *Processing instruction*
  - *Notation*
  - *Unparsed entity reference*
  - *Namespace*



## XML Infoset: Beispiel

- **Beispiel:** Anschreiben!

```
<?xml version="1.0"?>
<msg:message doc:date="19990421"
    xmlns:doc="http://doc.example.org/namespaces/doc"
    xmlns:msg="http://message.example.org/">
  Phone home!
</msg:message>
```

erzeugt folgendes XML Infoset:

- Ein `document` *information item*
- Ein `element` *info item*
- Ein `attribute` *info item*
- Drei `namespace` *info items*
- Zwei `attribute` *info items,* für die 2 *namespace*-Attribute
- Elf `character` *info items,* für Zeichenkette „`Phone home!`“

## XML Infoset: *Item Properties*

- **Document Information Item**
  - ***children***
    - Eine geordnete Liste, in Dokumentenreihenfolge
    - Optional ein *document type declaration info item*
    - Genau ein *element info item*
    - Ein *PI info item* pro Vorkommen außerhalb des Dokumentenelements (Vorkommen innerhalb der DTD sind ausgenommen).
  - ***document element***
    - Das *element info item*, entspricht dem Dok.element.
  - ***notations***
    - Eine (ungeordnete) Menge von *notation info items*, je eins pro NOTATION-Deklaration in der DTD.



## XML Infoset: *Item Properties*

- **Document Information Item** (Forts.)
  - ***unparsed entities***
    - Eine (ungeordnete) Menge von *unparsed entity info items*, je eins pro nicht vom Parser aufgelöster ENTITY-Deklaration in der DTD.
  - ***base URI***
    - *Base URI* der *document entity*.
  - ***character encoding scheme***
    - Der Name des *encoding*-Schemas, das dem *document entity* zugrunde liegt. Stammt aus der XML-Deklaration
  - ***standalone***
    - Der Wert der *standalone*-Deklaration aus der XML-Deklaration, entweder „*yes*“ oder „*no*“. Optional, „kein Wert“ wenn fehlt.

## XML Infoset: *Item Properties*

- **Document Information Item** (Forts.)
  - ***version***
    - Der Wert der *version*-Anweisung aus der XML-Deklaration. Optional, „kein Wert“ wenn fehlt.
  - ***all declarations processed***
    - Ein boolescher Wert, der nicht aus dem XML-Dokument selbst stammt, sondern anzeigt, ob der XML-Prozessor alle Deklarationen verarbeitet hat. Zwingend ist dies nur validierenden Parsern vorgeschrieben.



## XML Infoset

- Document info item **D1**:
  - children = ( **E1** )
  - document element = **E1**
  - notations = (kein Wert)
  - unparsed entities = (kein Wert)
  - base URI = „file://…“ // irgendeine passende Quelle
  - character encoding scheme = „UTF-8“ // oder (kein Wert) ?
  - standalone = (kein Wert)
  - version = „1.0“
  - all declarations processed = true

## XML Infoset: *Item Properties*

- **Element Information Item**
  - ***namespace name***
    - Der Name des Namensraums dieses Elements. Optional.
  - ***local name***
    - Der Name des Elementtyps, ohne *namespace*-Präfix und Doppelpunkt.
  - ***prefix***
    - Das *namespace*-Präfix des Elementtyp-Namens. Optional.
    - **Bem.**: Anwendungen, die *namespaces* unterstützen, sollten die Namensraum-Namen statt der Prafixwerte verwenden.



## XML Infoset: *Item Properties*

- **Element Information Item (Forts.)**
  - ***children***
    - Eine (eventuell leere) geordnete Liste, in Dokumentenreihenfolge
    - Enthaltene *Info items* sind vom Typ:
      - *element*
      - *processing instruction*
      - *unexpanded entity reference*
      - *character*
      - *comment*
  - ***attributes***
    - Eine (ungeordnete, ggf. leere) Menge von *attribute info items*, die entweder aus dem Element oder per *default*-Deklaration aus der DTD stammen.

## XML Infoset: *Item Properties*

- **Element Information Item (Forts.)**
  - ***namespace attributes***
    - Eine (ungeordnete, ggf. leere) Menge von *attribute info items*, je eins pro *namespace*-Deklaration aus dem Element oder per *default*-Deklaration aus der DTD.
    - Per Definition lautet das *namespace* URI aller *namespace*-Attribute „`http://www.w3.org/2000/xmlns`“, auch die ohne Präfix („`xmlns=…`“)
  - base URI
    - *Base URI* des Elements.
  - parent
    - Der *document* bzw. *element info item*, dessen *children*-Liste den vorliegenden Eintrag enthält.



## XML Infoset: *Item Properties*

- **Element Information Item (Forts.)**
  - ***in-scope namespaces***
    - Eine (ungeordnete) Menge von *namespace info items*, je eins für jeden Namensraum, der auf dieses Element wirkt.
    - Sie enthält immer einen Eintrag mit Präfix „`xml`“, der dem Namensraum „`http://www.w3.org/1998/namespace`“ implizit zugeordnet ist.
    - Sie enthält nie einen Eintrag mit Präfix „`xmlns`“, denn eine Anwendung kann nie auf ein Element oder Attribut mit diesem Präfix stoßen.
    - Sie enthält je einen Eintrag, der dem aus der Liste zu „*namespace attributes*“ stammt - mit folgender Ausnahme: Einträge zu Deklarationen der Form „`xmlns:''`“, welchen den *default*-Namensraum „ent-deklarieren“.
    - **Bemerkung:** Zur Auflösung von Präfixwerten sollte diese Menge Vorrang haben vor der Verwendung von „*namespace attributes*“, da letztere Probleme erzeugen können bei künstlich generierten Infosets.

## XML Infoset

- Element info item **E1**:
  - namespace name = „http://message.example.org/“
  - local name = „message“
  - prefix = „msg“
  - children = ( **C1**, **C2**, …, **C11** )
  - attributes = { **A1** }
  - namespace attributes = { **AN1**, **AN2** }
  - in-scope namespaces = { **N3**, **N1**, **N2** }
  - base URI = „file://…“ // irgendeine passende Quelle
  - parent = **D1**



## XML Infoset: *Item Properties*

- **Attribute Information Item**
  - ***namespace name***
    - Der Name des Namensraums dieses Attributs, sofern vorhanden. Sonst „ohne Wert“.
  - ***local name***
    - Der Name des Attributtyps, ohne *namespace*-Präfix und Doppelpunkt.
  - ***prefix***
    - Das *namespace*-Präfix des Elementtyp-Namens, sonst „ohne Wert“.
    - **Bem.**: Anwendungen, die *namespaces* unterstützen, sollten die Namensraum-Namen statt der Prafixwerte verwenden.
  - ***normalized value***
    - Der gemäß XML 1.0 normierte Attributwert.

## XML Infoset: *Item Properties*

- **Attribute Information Item (Forts.)**
  - ***specified***
    - Ein *flag*, das anzeigt, ob das Attribut tatsächlich im Element spezifiziert wurde (und nicht per default-Deklaration der DTD gefüllt wurde).
  - ***attribute type***
    - Gültige Werte sind ID, IDREF, ENTITY, ENTITIES, NMTOKEN, NMTOKENS, NOTATION, CDATA und ENUMERATION.
    - „ohne Wert“, falls das Attribut nicht deklariert wurde,
    - „unbekannt“, falls es eine ungelesene Deklaration gab.



## XML Infoset: *Item Properties*

- **Attribute Information Item (Forts.)**
  - ***references***
    - „ohne Wert“, falls ID, NMTOKEN, NMTOKENS, CDATA oder ENUMERATION.
    - „unbekannt“, falls der Attributtyp „unbekannt“ ist
    - sonst (falls IDREF, IDREFS, ENTITY, ENTITIES, NOTATION):
      Eine geordnete Liste der *info items* vom Typ
      - *element, unparsed entity* oder *notation*,

      in der Reihenfolge ihres Erscheinens im Attributwert, bzw. „ohne Wert“ oder „unbekannt“ im Fall von Inkonsistenzen.
  - ***owner element***
    - Der *element info item*, aus dem das Attribut stammt.

## XML Infoset

- Attribute info item **A1**:
  - namespace name = „http://doc.example.org/namespaces/doc“
  - local name = „date“
  - prefix = „doc“
  - normalized value = „19990421“
  - specified = „yes“ // „1“, true, ... ?
  - attribute type = (kein Wert)
  - references = (kein Wert) // ?
  - owner element = **E1**



## XML Infoset

- Attribute info item **AN1**:
  - namespace name = „http://www.w3.org/2000/xmlns“
  - local name = „doc“
  - prefix = „xmlns“ // ?
  - normalized value = „http://doc.example.org/namespaces/doc“
  - specified = „yes“ // „1“, true, ... ?
  - attribute type = (kein Wert)
  - references = (kein Wert) // ?
  - owner element = **E1** // ?

# XML Infoset

- Attribute info item **AN2**:
  - namespace name = „http://www.w3.org/2000/xmlns“
  - local name = „msg“
  - prefix = „xmlns“ // ?
  - normalized value = „ http://message.example.org/“
  - specified = „yes“ // „1“, true, ... ?
  - attribute type = (kein Wert)
  - references = (kein Wert) // ?
  - owner element = **E1** // ?



# XML Infoset: *Item Properties*

- **Processing Instruction Information Item**
  - ***target***
    - Eine Zeichenkette, die den „*target*“-Teil der PI enthält (ein XML *name*).
  - ***content***
    - Eine Zeichenkette, die den Inhaltsteil der PI enthält, ohne den *target*-Teil und führende *whitespace*-Zeichen. Ggf. die leere Zeichenkette.
  - ***base URI***
    - Die base URI der PI.
  - ***notation***
    - Falls verwendet: Der *notation info item* aus der PI. Sonst: „kein Wert“ bzw. „unbekannt“, je nachdem ob eine Deklaration fehlt oder eventuell nicht gelesen werden konnte.
  - ***parent***
    - Der *info item*, in dessen *children*-Liste dieser *PI info item* erscheint. Mögliche Typen sind: *document*, *element*, *document type definition*.

# XML Infoset: *Item Properties*

- **Unexpanded Entity Reference Information Item**
  - ***name***
    - Eine Zeichenkette, die den „*target*“-Teil der PI enthält (ein XML *name*).
  - ***system identifier***
    - Eine Zeichenkette mit dem angegebenen URI, ohne zusätzliche *escape*-Zeichen. Je nach Sachlage auch „ohne Wert“ oder „unbekannt“.
  - ***public identifier***
    - Eine Zeichenkette mit dem angegebenen *public identifier* in normierter Darstellung. Je nach Sachlage auch „ohne Wert“ oder „unbekannt“.
  - ***declaration base URI***
    - *Base* URI des *entity*, in dem die nicht aufgelöste *entity*-Referenz erscheint.
  - ***parent***
    - Der *element info item*, in dessen *children*-Liste dieser *info item* erscheint.

**Bemerkung:**
Validierende XML-Prozessoren erzeugen keine solchen *info items*.



# XML Infoset: *Item Properties*

- **Character Information Item**
  - ***character code***
    - Der ISO 10646 / Unicode-Wert des Zeichens.
  - ***element content whitespace***
    - Eine boolesche Variable, „*true*“ falls das dargestellte Zeichen vom Typ *whitespace* ist, „*false*“ wenn nicht.
    - Falls die Deklaration des zugrundeliegenden Elements nicht existiert bzw. nicht gelesen wurde, „ohne Wert“ bzw. „unbekannt“.
    - Validierende XML-Prozessoren müssen diese Information stets liefern.
  - ***parent***
    - Der *element info item*, in dessen *children*-Liste dieser *info item* erscheint.

## Bemerkungen

- Vorsicht - erheblicher *overhead* bei großen Dokumenten mit viel Freitext innerhalb von Elementen.
- Anwendungen steht es frei, die Zeichen wieder zu verketten.
- Gerade hier gehen XPath und XSLT andere Wege!

## XML Infoset

- Character info item **C1**:
  - character code = „P“
  - element content whitespace = false
  - parent = **E1**
- …
- Character info item **C6**:
  - character code = „ “
  - element content whitespace = true
  - parent = **E1**
- …
- Character info item **C11**:
  - character code = „!“
  - element content whitespace = false
  - parent = **E1**



## XML Infoset: *Item Properties*

- **Comment Information Item**
  - ***content***
    - Die Zeichenkette mit dem Kommentarinhalt.
  - ***parent***
    - Der *document* oder *element info item*, in dessen *children*-Liste dieser *info item* erscheint.
- Bemerkungen
  - Kommentare innerhalb von DTDs gelangen nicht in den Infoset!

## XML Infoset: *Item Properties*

- **Document Type Declaration Information Item**
  - ***system identifier***
    - Eine Zeichenkette mit dem in der DOCTYPE-Deklaration angegebenen URI des externen *Subsets*, ohne zusätzliche *escape*-Zeichen. Je nach Sachlage auch „ohne Wert“ oder „unbekannt“.
  - ***public identifier***
    - Eine Zeichenkette mit dem angegebenen *public identifier* in normierter Darstellung. Je nach Sachlage auch „ohne Wert“ oder „unbekannt“.
  - ***children***
    - Eine geordnete Liste der *PI info items* in der Reihenfolge, wie die entsprechenden PI in der DTD erscheinen.
    - PI im internen Subset erscheinen als erste.
  - ***parent***
    - Der *document info item*.

**Bemerkungen**

- *entities* und *notations* werden vom *document info item* erfasst.



## XML Infoset: *Item Properties*

- **Unparsed Entity Reference Information Item**
  - ***name***
    - Der Name des *entity*.
  - ***system identifier***
    - Eine Zeichenkette mit dem in der Deklaration angegebenen URI.
  - ***public identifier***
    - Eine Zeichenkette mit dem angegebenen *public identifier* in normierter Darstellung. Auch „ohne Wert“ oder „unbekannt“ mögl.
  - ***declaration base URI***
    - Das base URI des *entity*, in dem die *entity*-Referenz erscheint.
  - ***notation name***
    - Der Name der zugeordneten *notation*.
  - ***notation***
    - Der *notation info item*, auf den die *entity*-Referenz verweist.

## XML Infoset: *Item Properties*

- **Notation Information Item**
  - ***name***
    - Der Name der *notation*.
  - ***system identifier***
    - Eine Zeichenkette mit dem in der NOTATION-Deklaration angegebenen URI, ggf. „ohne Wert“.
  - ***public identifier***
    - Eine Zeichenkette mit dem angegebenen *public identifier* in normierter Darstellung. Je nach Sachlage auch „ohne Wert“.
  - ***declaration base URI***
    - Das base URI des *entity*, in dem die NOTATION-Deklaration erscheint.



## XML Infoset: *Item Properties*

- **Namespace Information Item**
  - ***prefix***
    - Die mit einem *namespace* zu assoziierende Zeichenkette „ohne Wert“ im Fall des *default namespace* (Element-Präfix „xmlns:“).
  - ***namespace name***
    - Der zugeordnete Name des *namespace*.
- Bemerkungen:
  - Fast alle *info items* entsprechen direkt bestimmten Objekten aus der XML 1.0-Spezifikation.
  - Dieser hier nicht: Die *namespace*-Spezifikationen finden hier von Anfang an volle Unterstützung, nicht nur im Nachhinein.

## XML Infoset

- Namespace info item **N1**:
  - prefix = „msg“
  - namespace name = „http://message.example.org/“
- Namespace info item **N2**:
  - prefix = „doc“
  - namespace name = „http://doc.example.org/namespaces/doc“
- Namespace info item **N3**:
  - prefix = „xml“
  - namespace name = „http://www.w3.org/1998/namespace“





# XML Infoset

Das Eingangsbeispiel,
nun genauer

# XML Infoset

**Das Eingangs-Beispiel nochmal:**

```
<?xml version="1.0"?>
<msg:message doc:date="19990421"
    xmlns:doc="http://doc.example.org/namespaces/doc"
    xmlns:msg="http://message.example.org/">
    Phone home!</msg:message>
```

**Es erzeugt folgendes XML Infoset - diesmal genauer, nur ohne Verkettungen:**

- Ein `document` *information item*
- Ein `element` *info item*
  - mit *namespace*-Eintrag „`http://message.example.org/`“
  - mit *local part* „`message`“ und Präfix „`msg`“
- (b.w.)



# XML Infoset

- Ein `attribute` *info item*
  - mit *namespace*-Eintrag „`http://doc.example.org/namespaces/doc`“
  - mit *local part* „`date`“ und Präfix „`doc`“
  - mit dem normierten Wert „`19990421`“
- Drei `namespace` *info items*
  - für die Namensräume „`http://www.w3.org/XML/1998/namespace`“, „`http://doc.example.org/namespaces/doc`“ und „`http://message.example.org/`“
- Zwei `attribute` *info items,* für die beiden *namespace*-Attribute
- Elf `character` *info items,* für die Zeichenkette „`Phone home!`“

# XML Infoset

Was ein XML Infoset nicht enthält

Also: Was eine Anwendung nie über Ihre XML-Dateien erfährt…

## XML Infoset

- Was ein XML Infoset nicht enthält
  - Folgen der Expansionen der XML-Prozessoren
    - Die Repräsentation von Zeichen (direkt, per *char ref*, per *entity ref*, per *CDATA section*)
    - Die Grenzen von INCLUDE/IGNORE-Abschnitten in der DTD
    - Die Grenzen von *parameter entities* in der DTD.
    - Überlesene Deklarationen, z.B. innerhalb von IGNORE-Abschnitten
    - Die Grenzen von *general parsed entities*.
    - Die Grenzen von CDATA sections.
  - Folgen der Normierung der XML-Prozessoren
    - Die Art der Zeilenende-Codierung.
    - Die Art der verwendeten Anführungszeichen.



## XML Infoset

  - Sonstiges
    - Die *content models* der Elementyp-Deklarationen aus der DTD.
    - Gruppierung/Anordnung von Attributen laut ATTLIST-Deklaration.
    - Der Name des Dokumenttypen.
    - *White space:*
      - Außerhalb des *document element*
      - Der dem *target name* einer PI unmittelbar folgende
      - Innerhalb von *tags*.
    - Die Darstellung eines leeren Elements (<foo/> vs. <foo></foo>)
    - Die Reihenfolge der Attribute in einem *start tag*.
    - Die Reihenfolge der Deklarationen in der DTD.
    - Kommentare in der DTD.
    - Die Position von Deklarationen, z.B. intern vs. extern vs. indirekt per *parameter entity*.
    - Die *default*-Werte von Attributen wie in der DTD deklariert.

# XML Path Language (XPath)

http://www.w3.org/TR/xpath





## Was ist XPath?

- Eine Sprache
  - zur Adressierung / Selektion von Teilen logischer XML-Dokumente
  - zur Prüfung ob ein Knoten des XML-Dokumentenbaumes bestimmte Bedingungen erfüllt (*pattern matching*)
  - zur Manipulation von Strings, Zahlen und booleschen Ausdrücken
  - mit ähnlicher Bedeutung für XML *information sets* wie SQL für relationale Datenbankmodelle/Schemata.
- Die Sprache
  - ist kompakt (verwendet nicht die XML-Syntax)
  - ist erweiterbar
- Der Name XPath
  - erklärt sich durch ihre Erweiterbarkeit und eine an Pfade in Dateisystemen erinnernde, allerdings stark verallgemeinernde Notation, z.B. `/doc/chapter[5]/section[2]`, `chapter//para`, aber auch: `para[@type="warning"]`.

## Was ist XPath?

- XPath ist die Grundlage
  - historisch von XSLT und XPointer
  - Inzwischen auch von weiteren XML-Technologien, z.B. von
    - XML Schema (*identity constraints*)
    - Schematron (alle *constraints*)
    - XQuery
- XPath bezieht sich auf logische XML-Dokumente, bestehend aus Knoten verschiedenen Typs, angeordnet in Baumform, analog zu - aber nicht identisch mit - XML Information Sets.
- XPath
  - unterstützt in Version 1.0 bereits XML-Namensräume
  - ist aber noch nicht auf XML Schema abgestimmt
  - entwickelt sich gegenwärtig zusammen mit XQuery weiter (XPath 2.0 + XQuery 1.0).



## Das XPath-Datenmodell

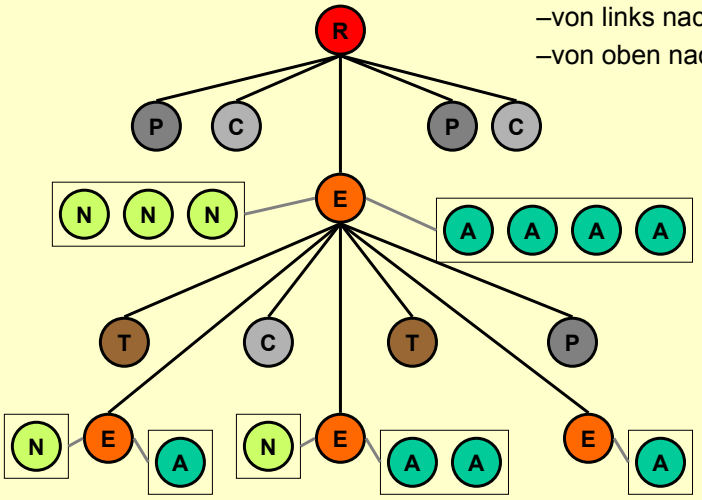
- XPath unterstellt
  - dass ein XML-Dokument als Baumstruktur vorliegt und
  - in bestimmter Weise seriell gelesen werden kann (*document order*).
- XPath definiert und verwendet 7 Knotentypen:
  - *R: root node* (Dokumentwurzelknoten)
    - genau einer pro Dokument
  - *E: element nodes* (Element-Knoten)
  - *T: text nodes* (Textknoten)
  - *A: attribute nodes* (Attribut-Knoten)
  - *N: namespace nodes* (Namensraum-Knoten)
  - *P: processing instruction nodes* (PI-Knoten)
  - *C: comment nodes* (Kommentarknoten)
- Nur die Typen R und E können verkettet werden.
  - Bem.: Die anderen sollte man also als „Blätterknoten“ bezeichnen.
  - Typen N und A sind ihren E-Knoten assoziiert („separate“ Zweige).

# Das XPath-Datenmodell

**Lesefolge**

–von links nach rechts,

–von oben nach unten






# Das XPath-Datenmodell

- Knoteneigenschaften
  - Für jeden Knoten(typ) läßt sich ein Stringwert ermitteln.
  - Viele Knoten besitzen einen expandierten Namen, bestehend aus
    - *namespace URI* *null* oder ein String
    - *local part* ein String
- Knotenreihenfolge (Dokumentenreihenfolge)
  - Erster Knoten ist stets der *root node* R., nicht zu verwechseln mit dem ersten Elementknoten, dem des *document element!*
  - Elementknoten erscheinen in der Reihenfolge ihrer *start tags*, also Eltern- vor Kind-Knoten.
  - Einem Elementknoten E folgen ggf. seine *namespace nodes*, dann seine *attribute nodes*, dann seine *child nodes*.
  - Die Reihenfolge der *namespace* und *attribute nodes* ist unbestimmt und damit implementierungsabhängig.

# XPath und XML Infoset

- XPath ist „nur“ am Inhalt von Dokumentinstanzen interessiert
  - in möglichst „bequemer“ Form
  - Validierbarkeit interessiert nicht, Wohlgeformtheit reicht.
- Im Vergleich zu XML Infoset wurden daher Vereinfachungen vorgenommen.
  - Es gibt keine Entsprechung zu:
    - *Document type info items*
    - *Unparsed entity reference info items*
    - *Notation info items*
  - Die Dokumententopologie wurde zu einem Baum vereinfacht
  - Einige *info item*-Eigenschaften werden ignoriert
    - Beispiel: Angaben aus der XML-Deklaration
  - Folgen von *character info items* werden zu *text nodes* zusammengefasst, *whitespace*-Angaben gehen verloren.
- Vorsicht: XPath unterstellte eine inoffizielle Infoset-Spezifikation!



# XPath und XML Infoset

| Knoten | Stringwert | Expandierter Name |
|---|---|---|
| • R | Stringwert von E | - |
| • E | Konkatenierung der Stringwerte aller enthaltenen T | *ii: namespace URI, local name* (ii = *information item*) |
| • A | ii: *normalized value* | *ii: namespace URI, local name* |
| • T | Konkatenierung aller *character values* aufeinanderfolgender *character info items* | - |
| • N | *ii: namespace URI* | *null, ii: prefix* |
| • P | *ii: content* | *null, ii: target* |
| • C | *ii: content* | - |

Bem.: Mozilla zeigt ohne CSS offenbar den Stringwert von R an!

## XPath-Ausdrücke

- XPath-Ausdrücke sind die zentralen Objekte der Sprache. Sie werden ausgewertet und liefern ein Objekt zurück, etwa:
  /mydoc/selection liefert alle „selection“-Elemente von „mydoc“
- Es gibt 4 derartige Objekttypen:
  - Knotenmenge (*node set*) - der mit Abstand wichtigste Fall!
  - Boolescher Ausdruck (*true* oder *false*), Zahl (*floating-point*) und String (UCS)
- Auswertungen finden stets in einem Kontext statt:
  - dem Kontext-Knoten (per *default* ist das zunächst R)
  - der Kontext-Position und -größe (zwei positive ganze Zahlen)
  - ein Satz Variablen mit ihren aktuellen Inhalten
  - einem Satz Funktionen (*core functions* + *extensions*)
  - einem Satz Namensraum-Deklarationen
- Der Kontext wird vom XPath-Verwender definiert, z.B. von XSLT



## XPath-Ausdrücke

Rückgabetyp „Knotenmenge“:

- Der XPath-Ausdruck soll alle Knoten des Dokumentbaumes zurückliefern,
  - auf die der übergebene Ausdruck passt
  - und zwar im aktuellen Kontext.
- Wichtigster Spezialfall eines Ausdrucks, der Knotenmengen erzeugt: *location path* (etwa: „Suchpfad“).
- Unterscheide relative und absolute Suchpfade:
  - Relative: section/para, ./customer
  - Absolute: /mydoc/section/para, /book/title

(in Analogie zu Dateisystem-Pfaden)

# XPath-Ausdrücke

Rückgabetyp „Knotenmenge“ (Forts.):

- Suchpfade setzten sich aus Suchschritten (*location steps*) zusammen, diese bestehen wiederum aus drei Teilen:
  - einer „Achse“ (*axis*)
  - einem Knotentest (*node test*)
  - optionalen Prädikaten (*predicates*)
- Beispiel:

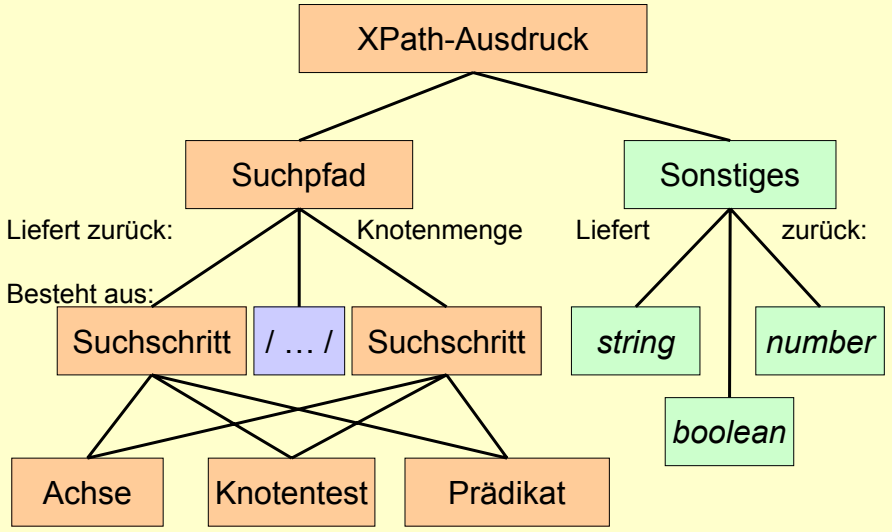
```
child::para[position()=1]
```

  Ausgehend vom Kontextknoten, das erste Kind-Element namens „para“.
  Hinweis auf Kurzschreibweise: `para[1]`

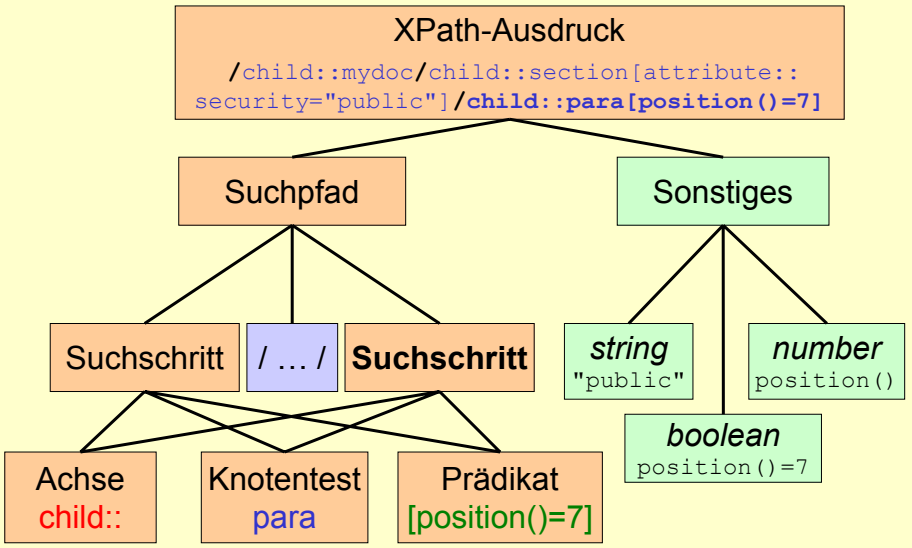
- Suchschritte werden durch „/“ getrennt.



# XPath-Ausdrücke



# XPath-Ausdrücke: Beispiel






# Bildung der Knotenmenge

- Prinzip der Teilmengenbildung:
  - Die Auswertung erfolgt immer von links nach rechts.
  - Suchschritt n+1 arbeitet mit der Knotenmenge $K_n$ von Suchschritt n, ergibt also höchstens eine Teilmenge $K_{n+1} \subseteq K_n$

    `/ step1 / step2 / …`
  - Innerhalb eines Suchschritts reduziert ein Prädikat die Knotenmenge des Knotentests weiter (Filter):

    `/ … / step[expression] / …`
  - Mehrere Prädikate hintereinander erzeugen ebenfalls verschachtelte Teilmengen. Reihenfolge beachten!

    `… /step2[expr1][expr2][expr3]/ step3 / …`
- Komplexe Ausdrücke:
  - Ausdrücke in Prädikaten können ihrerseits ganze Suchpfade enthalten!

# XPath-Ausdrücke

- Beispiele für einen Suchschritt:

| | |
|---|---|
| `child::para` | Alle para-Kindelemente des Kontextknotens K |
| `child::*` | Alle Kind-Elementknoten des Kontextknotens |
| `child::text()` | Alle Kind-Knoten vom Texttxp von K. |
| `child::node()` | Alle Kind-Knoten von K, unabhängig vom Typ |
| `descendant::*` | Alle Abkömmlinge des Kontextknotens K |
| `/` | *document root* (R) |
| `child::para[position()=1]` | Der erste Kind-Elementknoten namens „para“ des Kontextknotens. |



# XPath-Achsen

- XPath-Ausdrücke suchen ihre Knoten entlang sogenannter „Achsen“.
- Es gibt 13 Achsen.
  Die wichtigsten lassen sich in intuitiver Weise abkürzen:
  - **self** Nur der Kontextknoten K selbst
  - **child** Default-Achse, daher darf der Achsenteil child:: eines Suchpfades einfach fehlen.
  - **descendant** Alle Abkömmlinge von K
  - **descendant-or-self** K und alle Abkömmlinge
  - **parent** Der Elternknoten von K
  - **ancestor** Alle „Ahnen“-Knoten von K
  - **ancestor-or-self** K und alle „Ahnen“-Knoten

## XPath-Achsen

- Weitere Achsen:
  - **following-sibling** Alle nachfolgende „Geschwister“-Knoten von K. Leer falls K vom Typ A oder N.
  - **following** Alle nachfolgenden Knoten, außer Kindknoten, Attribut- sowie Namensraumknoten.
  - **preceding-sibling** Alle vorangehenden „Geschwister“-Knoten von K. Leer falls K vom Typ A oder N.
  - **preceding** Alle vorangehende Knoten, außer Ahnenknoten, Attribut- sowie Namensraumknoten.
  - **attribute** Abgekürzt einfach durch Präfix „@“ child::para[attribute::type="warning"] entspricht para[@type="warning"].
  - **namespace** Alle Namensraumknoten des Kontextknotens. Leer falls K selbst vom Typ A oder N



## XPath-Knotentests

- Jede Achse besitzt einen Hauptknotentyp (*principal node type*)
  - Fall A: *attribute*
  - Fall N: *namespace*
  - sonst: *element*
- Namespace-Behandlung:
  - Besteht ein Knotentest aus einem *QName,* ist er genau dann wahr,
    - wenn der aktuelle Knoten vom Hauptknotentyp der Achse ist
    - und sein Name (*expanded-name*) mit dem expandierten (!) *QName* übereinstimmt.
- Erfüllt kein Knoten entlang der aktuellen Achse den Test, wird die leere Menge (von Knoten) zurückgegeben.

## XPath-Knotentests

- Eingebaute Knotentests:

| | |
|---|---|
| * | alle Knoten des Hauptknotentyps der Achse |
| text() | alle Knotentypen T der Achse |
| comment() | alle Knotentypen C der Achse |
| processing-instruction() | alle Knotentypen P der Achse |
| node() | alle Knoten der Achse. |



## XPath-Prädikate

- Mit Achse und Knotentest eines XPath-Ausdrucks wurde bereits eine Knotenmenge gebildet. Diese kann mit einem Prädikat weiter eingeschränkt werden (**Filter**).
- Dazu wird für jeden Knoten der Knotenmenge der Ausdruck des Prädikats ermittelt. Ist er wahr, gelangt der Knoten in die neue Knotenmenge, sonst nicht.
- Dabei ist jeweils *context size* die Anzahl Knoten der (alten) Knotenmenge, *context position* die „Entfernung“ des Knotens entlang der aktuellen Achse zum Kontextknoten.
- Numerische Angaben im Ausdruck werden zu „wahr“, wenn sie mit *context position* übereinstimmen. Das erklärt folgende Kurzschreibweise:

  para[3] ist äquivalent zu para[position()=3]

## XPath-Kurzschreibweisen

| Explizite Angabe | Kurzform |
|---|---|
| child:: | (kann entfallen) |
| self::node() | . |
| parent::node() | .. |
| /descendant-or-self::node()/ | // |
| attribute:: | @ |



## XPath-Kurzschreibweisen

**Beispiele:**

../title Alle „title“-Knoten des Elternknotens von K

.//para Alle „para“-Abkömmlinge von K
(kann K selbst einschließen)

para[5][@type="warning"]
Das fünfte „para“-Kindelement von K,falls es ein Attribut „type“ mit Wert „warning“ besitzt.

para[@type="warning"][5]
Das fünfte „para“-Kindelement von K,
das ein Attribut „type“ mit Wert „warning“ besitzt.

## Operatoren in XPath-Ausdrücken

- Ausdrücke in XPath können zusammengesetzt sein, verknüpft mit Operatoren.
- Operator |
  - Vereinigungsmenge der Ergebnisse zweier Ausdrücke, die eine Knotenmenge als Ergebnis haben.
- Operatoren and, or
  - Verknüpfen Ausdrücke mit booleschem Ergebnis
- Operatoren =, !=, <, <=, >, >=
  - Unterschiedliche Vergleiche sind möglich, mit impliziten Typkonvertierungsregeln, die im Detail in der XPath-Spezifikation nachzulesen sind (Kap. 3.4)



## XPath-Funktionen

- *Vorbemerkungen:*
  - Ziel an dieser Stelle ist eine Zusammenstellung der vorhandenen Funktionen und ihre ungefähre Verwendung, damit eine rasche Orientierung bzw. ein gezieltes Nachschlagen möglich wird.
  - Hier ist nicht genug Raum für alle Einzelheiten der Spezifikationen. Lesen Sie bei Bedarf die Details in den Spezifikationen nach! (Kap. 4)
  - Auf den Mechanismus der XPath-Erweiterungen wird hier nicht eingegangen. Er erfolgt grundsätzlich auf der Ebene der XPath nutzenden Anwendung wie etwa XSLT und besteht etwa aus dem Hinzufügen weiterer Funktionen.
    - VORSICHT: Erweiterungen führen leicht zu Plattform-abhängigen Implementierungen!

# XPath-Funktionen

- *Core function library:* Funktionen auf Knotenmengen (*node set functions*)

**`number last()`**

Rückgabewert = *context size*

**`number position()`**

Rückgabewert = *context position*

**`number count(node-set)`**

Rückgabewert = Anzahl Knoten in der Knotenmenge

**`node-set id(object)`**

Liefert alle Knoten, deren ID-Attribute einem der Stringwerte des übergebenen Objekts entsprechen. Dieses Objekt kann ein einfacher String sein, eine Liste von Tokens oder auch eine Knotenmenge, wobei die Stringwerte der Knoten die Liste der Tokens ergeben.

**Bemerkung**: id() funktioniert ohne DTD nicht, denn nur aus der DTD erhält der Parser die Information, welche Attribute „ID“s sind.



# XPath-Funktionen

- *Core function library:* Funktionen auf Knotenmengen (*node set functions*)

**`string local-name(node-set?),`**

**`string namespace-uri(node-set?),`**

**`string name(node-set?)`**

Liefert *local-name* bzw. *namespace URI* bzw. QName (I.d.R. gleich *expanded name*) des *expanded-name* des ersten Knoten der übergebenen Knotenmenge (in Dokumentenreihenfolge) bzw. des Kontextknotens.

Einzelheiten ggf. in den Spezifikationen nachlesen, Kap. 4.1.

# XPath-Funktionen

- *Core function library:* String-Funktionen

**string string(object?)**

*node-set*: Stringwert des ersten Knoten in Dokumentenreihenfolge

*number*: *NaN*, *Infinity*, *-Infinity*, *0*, bzw. die Dezimaldarstellung

*boolean*: *true* bzw. *false* (als Strings)

andere: erfordert Umwandlung in *string*, kontextabhängig.

*default:* Stringwert des Kontextknotens.

**string concat(string, string, string*)**

Selbsterklärend

**boolean starts-with(string, string)**

*true* wenn das erste Argument mit dem zweiten beginnt

**boolean contains(string, string)**

*true* wenn das erste Argument das zweite enthält



# XPath-Funktionen

- *Core function library:* String-Funktionen

**string substring-before(string, string),**

**string substring-after(string, string)**

Liefert den Teilstring des ersten Arguments vor bzw. nach dem ersten Auftreten des zweiten Arguments. Beispiel:

`substring-before("1999/04/01","/")` liefert `"1999"`

**string substring(string, number, number?)**

Liefert den Teilstring des ersten Arguments ab der vom zweiten Argument angegebenen Position bis zum Ende bzw. mit der vom dritten Argument angegebenen Länge. Beispiel:

`substring("12345",2,3)` liefert `"234"`

VORSICHT: Hier wird ab 1 gezählt, nicht ab 0 wie in einigen anderen Sprachen.

# XPath-Funktionen

- *Core function library:* String-Funktionen

**number string-length(string?)**

Selbsterklärend, wirkt per default auf den Stringwert des Kontextknotens

**string normalize-space(string?)**

Normiert den Argument-String analog zur XML-Normierungsregel für die Werte von NMTOKEN-Attributtypen, wirkt per default auf den Stringwert des Kontextknotens.

**string translate(string, string, string)**

Ersetzt Zeichen des ersten Arguments bzw. entfernt sie. Kommt ein Zeichen im zweiten Argument vor, wird es ersetzt durch das an gleicher Stelle im dritten Argument stehende Zeichen. Fehlt dieses, wird das Zeichen entfernt. Analog Unix-Tool bzw. Perl-Funktion „tr“. Beispiel: `translate("abCdE","abE","AB")` liefert `"ABCd"`



# XPath-Funktionen

- *Core function library:* Boolesche Funktionen

**boolean boolean(object)**

*number*: *true*, wenn Zahl <> 0 und <> NaN.

*string*: *true*, wenn seine Länge > 0 ist.

*node-set*: *true*, wenn die Menge nicht leer ist.

andere: erfordert Umwandlung in *boolean*, kontextabhängig.

**boolean not(boolean),**

**boolean true(),**

**boolean false()**

selbsterklärend

**boolean lang(string)**

*true*, wenn der übergebene Sprachschlüssel zu dem des Attributs `xml-lang` des Kontextknotens „passt“.

## XPath-Funktionen

- *Core function library:*
  Funktionen auf numerischen Variablen

  **`number number(object?)`**

  *string*: Umwandlung wie üblich, ggf. in „NaN“

  *boolean*: Umwandlung in 1 bzw. 0 (0 entspricht *false*)

  *node-set*: Behandlung des Stringwertes der Knotenmenge wie im Fall *string.*

  andere: Erfordert kontextabhängige Typumwandlung.

  **`number sum(node-set)`**

  Summe der Umwandlungsergebnisse aller Stringwerte der Knoten in der Knotenmenge in Zahlen gemäß Funktion *number*.

  **`number floor(number),`**

  **`number ceiling(number),`**

  **`number round(number):`**

  Die üblichen Rundungsoperationen.



## XPath-Funktionen

- *Core function library:*
  Funktionen auf numerischen Variablen

  **`number floor(number),`**

  **`number ceiling(number),`**

  **`number round(number)`**

  Die üblichen Rundungsoperationen, *integer*-wertig.

- Operatoren für Zahlen
  - Zahlen entsprechen „double“ gemäß IEEE 754

    **`+, -, *`** Addition, Subtraktion, Multiplikation

    **`div, mod`** Division, Modulo
- Bemerkung

  „div“ statt „/“ vermeidet „path“-Verwechlungen.

## Variablen in XPath

- XPath selbst besitzt keine Möglichkeiten zur Definition von Variablen, aber von „außen“ vorgegebene Variablen lassen sich in XPath-Ausdrücken verwenden.
- Beispiel:

```
/mydoc//title[@myattr=$mysample]
```

  - Selektiert nur "title"-Elemente, deren Attribut "myattr" einen in der Variablen $mysample enthaltenen Wert besitzt.
- XML-Standards wie XSLT, die XPath verwenden, gestatten es, Variablen zu definieren.
- Methodische Grenzen:
  - Elementnamen o.ä. in XPath dürfen keine Variablen sein.
  - descendant::$myname // nicht zulässig
  - descendant::*[name()=$myname] // ok, ohne *namespace*



## Erweiterbarkeit von XPath

- Erfolgt indirekt über die Erweiterbarkeit der XPath verwendenden Sprachen.
- Beispiele:
  - XSLT
  - XPointer

- Ausblick
  - XPath 2.0 wird XML Schema Datentypen unterstützen und damit Vergleiche und Berechnungen etwa mit Datum/Zeit-Angaben erheblich erleichtern…
  - … und wird Mengenoperationen auf Knotenmengen wie Vereinigung, Schnittmenge, Mengendifferenz bieten.
  - XQuery wird nicht nur Knotenmengen, sondern ganze XML-Fragmente zurückgeben, z.B. gebildet aus verschiedenen Bestandteilen der XML-Quelle.
  - XPath 2.0 wird gemeinsam mit XQuery 1.0 und XSLT 2.0 entwickelt.
  - XPath 2.0 wird nicht mehr von einem einzigen Dokument beschrieben, sondern von mehreren „Baustein“-Dokumenten.





# XPath: Beispiele

Kommentierte Beispiele zur Vertiefung
des Theorie-Teils

## XPath: Beispiele

- **`/`**
  - Der root-Knoten R des Dokuments.
- **`/mydoc`**
  - Der Dokumentenelement-Knoten
- **`/mydoc/section`**
  - Alle (!) `section`-Kindelemente von `mydoc`.
- **`/child::mydoc/child::section`**
  - Dito, aber mit expliziten Achsenangaben.
- **`/mydoc/section[@security="public"]/para[7]`**
  - Das siebte Element `para` jedes `section`-Elements, dessen Attribut `security` den Wert `public` hat.
- **`/child::mydoc/child::section[attribute::security="public"]/child::para[position()=7]`**
  - Dito, aber mit expliziten Achsenangaben und ohne implizite Kurzformen.



## XPath: Beispiele

- **`/mydoc//*`**
  - Alle Nachkommen von `mydoc`, einschließlich (?!) `mydoc`
- **`/mydoc//para`**
  - Alle `para`-Nachkommen (Elementknoten) von `mydoc`, unabhängig von ihrer Tiefe im Dokumentenbaum.
- **`/mydoc//@type`**
  - Alle Attributknoten des gesamten Dokuments namens `type`.
- **`/mydoc//comment()`**
  - Alle Kommentarknoten, die von `mydoc` abstammen.
- **`/mydoc//text()`**
  - Alle Textknoten, die von `mydoc` abstammen, d.h. alle *character data* des Dokuments!

## XPath: Beispiele

- `../@*`
  - Alle Attributknoten des Elternknotens des Kontextknotens.
- `//para[footnote]/[@important]`
  - Alle `para`-Elementknoten mit `footnote` Kindelementen und `important` Attributen.
- `section[author/qualifications[@professional][@affordable]]`
  - `section` Kindelemente des Kontextknotens von Autoren mit mehreren durch Attribute benannte Qualifikationen - ein Beispiel für die Möglichkeit, auch Prädikate aus eigenen Ausdrücken mit mehreren Schritten (*steps*) aufbauen zu können.
- `id('A12345')/title | /mydoc/title`
  - Vereinigungsmenge: Alle `title`-Kindknoten des Knotens mit der angegebenen ID und alle `title`-Kindknoten des Dokumentknotens.



## XPath: Beispiele

- `//ordered-list[item[@type]/para[2]]//para`
  - Etwas zum Nachdenken: `para`-Nachfahren von `ordered-list` Knoten, die `item`-Kindknoten besitzen, welche ein `type` Attribut und mindestens 2 `para`-Kindknoten aufweisen.
- `a//b[last()-2]`
  - Der drittletzte Nachfahre `b` von `a`, in Dokumentenreihenfolge.
- `//list[@type='ordered']/item[1]/para[1]`
  - Der erste `para`-Kindknoten des ersten `item`-Kindknotens aller `list`-Knoten des Dokuments, die ein Attribut `type` mit Wert `ordered` besitzen.

# XPath in APIs

Ein Beispiel zum Zugriff auf XML-Daten mit dem Ruby-API „REXML“



## XPath: REXML-Beispiel

```
#!/usr/bin/env ruby
#
require "rexml/document"
include REXML        # Vermeidet Präfix „REXML::“

# XML-Dokument als Datenstruktur in den Speicher laden:
file = File.new( "09-bestell.xml" )
doc  = Document.new file

# XPath-Ausdrücke im Folgenden in rot.

# Ausgabe der Texte (Inhalte) aller Elemente „Beschreibung“:
XPath.each( doc, "//Beschreibung") { |element|
  puts element.text
}
```

# XPath: REXML-Beispiel

# Array aller Elemente „ArtNr“, die eine ISBN enthalten:

```
articles = XPath.match( doc,"//ArtNr[@IdentArt='ISBN']")
```

# Rollen der Handelspartner:

```
doc.elements.each
   ("//Bestellkopf/Handelspartner") { |element|
       puts element.attributes["Rolle"]
}
```

# Liste aller Belegnummern:

```
doc.elements.each
   ("/Bestellungen/Bestellung/Bestellkopf") { |element|
       puts element.elements["Belegnummer"].text
}
```