

Auf einen Blick

Einleitung	17
1 XSLT und XPath – der Start	23
2 XSLT und XPath – Runde zwei	107
3 XSLT und XPath – Runde drei	171
4 Referenz XPath – diese Sprache	237
5 Referenz XPath-Funktionen	319
6 Referenz XSLT-Elemente	617
7 Referenz Saxon 7	909
8 XSLT-Entwicklungsumgebungen	927
A Lösungen	973
B W3C-Online-Ressourcen	1001
C Glossar	1009
Index	1043

Inhalt

Einleitung	17
1 XSLT und XPath – der Start	23
1.1 Die Themenverortung	25
1.2 Eine erste Transformation	27
1.2.1 Drei beteiligte Dokumente	27
1.2.2 Kopierte Inhalte: Literal Result Elements	29
1.2.3 Erster Einsatz von XSLT-Instruktionen	29
1.3 Einteilung der Elemente von XSLT	32
1.4 Das Wurzelement <code>xsl:stylesheet</code>	32
1.4.1 Das Wurzelement und seine Namensräume	33
1.4.2 Der Sinn eines Namensraums	33
1.5 Die Grundstruktur des Stylesheets: <code>xsl:template</code>	35
1.5.1 Regeln für den Aufruf des Templates	36
1.5.2 Das Vergleichsmuster – ein XPath-Pattern	36
1.6 Literal Result Elements unter der Lupe	37
1.6.1 Namensräume und Literal Result Elements	38
1.6.2 Default-Namensraum für Literal Result Elements	38
1.6.3 Namensraum mit Präfix für Literal Result Elements	39
1.6.4 Textlitterale in Literal Result Elements	40
1.6.5 Attribute in Literal Result Elements	40
1.7 Die Instruktionen im Templaterumpf	41
1.7.1 Exemplarisch: Die Instruktion <code>xsl:value-of</code>	42
1.7.2 XPath-Ausdruck vs. XPath-Pattern	43
1.8 Das XML-Dokument als Baum	43
1.8.1 Warum Baumstrukturen?	43
1.8.2 Eltern und Kinder im Baumdiagramm	44
1.8.3 Knotengattungen im Dokumentbaum	45
1.8.4 Dokumentknoten (document node)	46
1.8.5 Elementknoten (element node)	47
1.8.6 Textknoten (text node)	47
1.8.7 Der Attributknoten (attribute node)	48
1.8.8 Der Baum als Abbild eines Infosets oder PSVI	48
1.9 Eine Transformation Schritt für Schritt	50
1.9.1 Aktuell: Der Current Node im Quelldokument	50
1.9.2 Der fiktive Ergebnisbaum	51

1.10	Start mit XPath: Pfadausdrücke	53
1.10.1	Gewusst wo: Vom Auffinden der Inhalte	53
1.10.2	Bezugspunkt: Der Pfadausdruck im Kontext	55
1.10.3	Der Pfadausdruck – relativ und absolut	55
1.10.4	Einfache und zusammengesetzte Pfadausdrücke	56
1.10.5	Gefunden: Das Ergebnis des Pfadausdrucks	56
1.11	Von Achsen und Nodetests	58
1.11.1	Eine Achse für die Bewegungsrichtung	58
1.11.2	Ein Knotentest für die Auswahl	59
1.11.3	Eine kleine Achsenparade	59
1.11.4	Die zwei Schreibweisen für Achsenbezeichner	60
1.12	Pfadausdrücke – Beispiele	61
1.12.1	Pfadausdrücke auf der Child-Achse	62
1.12.2	Ein Platzhalter für Bezeichner – der Wildcardbezeichner	62
1.12.3	KindTests auf den Elementachsen	63
1.12.4	Übersprunghandlung: Der Ausdruck »//«	64
1.12.5	Auswahl des Kontextes – die Self-Achse	64
1.12.6	Der Schritt zum Elternelement – die Parent-Achse	65
1.13	Von Stringwerten und Attributwerten	66
1.13.1	Der Stringwert eines Elementknotens	66
1.13.2	Verwendung der Attributachse	67
1.13.3	Pfadausdrücke für Attributknoten:	68
1.14	Stylesheets mit mehreren Templates	69
1.14.1	Zur Verarbeitung: Eine XML-Adresskartei	70
1.14.2	Havarie: Die Grenzen des Stringwerts	71
1.14.3	Auslagern: Eine Templaterregel für die Adressen	72
1.14.4	Aktivist: Die Instruktion <code>xsl:apply-templates</code>	72
1.14.5	Weiter geht's: Die Fortführung der Aktivierungskette	74
1.15	Ein Vergleich – Ausdruck vs. Instruktion	75
1.15.1	Variante 1: Der XSLT-orientierte Ansatz	75
1.15.2	Das Problem mit den Whitespacenodes	75
1.15.3	Schutzwall für Leerzeichen: Die Instruktion <code>xsl:text</code>	77
1.15.4	Variante 2: Der XPath-orientierte Ansatz	78
1.15.5	Nebenschauplatz: Konstruktion von Sequenzen	79
1.15.6	Nichts drin: Die leere Sequenz	80
1.15.7	Konkretes: Literale in einer Sequenz	81
1.15.8	Von Singletons und Knotendubletten	82
1.15.9	Das separator-Attribut von <code>xsl:value-of</code>	82
1.15.10	Das Ziel: Die Sequenz nach Maß	83
1.15.11	Showdown: Vergleich zwischen beiden Varianten	84
1.16	Struktur steuert Verarbeitung	86
1.16.1	Eine Templaterregel pro Element – beinahe ...	86
1.16.2	Die Verarbeitung im freien Fluss	89
1.16.3	Defaulttemplates	90
1.16.4	Defaultregel für Element- und Dokumentknoten	91
1.16.5	Defaultregel für Text- und Attributknoten	91
1.16.6	Defaultregel für Kommentar- und PI-Knoten	91
1.16.7	Defaultregel für Namensraumknoten	92
1.16.8	Der strukturgesteuerte Ablauf der Verarbeitung	92

1.17	Mehr Kontrolle über das Ergebnis	94
1.17.1	Die Deklaration <code>xsl:output</code>	95
1.17.2	Weg mit dem Leerraum: <code>xsl:strip-space</code>	98
1.17.3	Reservate für Leerraum: <code>xsl:preserve-space</code>	99
1.18	Zusammenfassung und Aufgaben	101
1.18.1	Testfragen	101
1.18.2	Aufgaben	102

2 XSLT und XPath – Runde zwei 107

2.1	XPath – Filtern von Sequenzen	109
2.1.1	Predicates – Filterbedingungen für Location-Steps	110
2.1.2	Die Filterung einer Sequenz	111
2.1.3	Logische Verknüpfung von Bedingungen mit AND und OR	114
2.1.4	XPath-Funktionen in Predicate-Ausdrücken	115
2.2	Bedingungen mit XSLT und XPath	119
2.2.1	Entscheidungen mit XSLT: <code>xsl:if</code>	119
2.2.2	Mehr Entscheidungen in XSLT: <code>xsl:choose</code>	121
2.2.3	Alternative in XPath: <code>if ... then ... else</code>	123
2.3	Nummerieren, Zählen, Summieren	125
2.3.1	Einfache Nummerierung	126
2.3.2	<code>xsl:number</code>	126
2.3.3	Zählen, Summieren und Runden	135
2.4	Sortierung – <code>xsl:sort</code>	138
2.4.1	Sortierung von Sequenzen – <code>xsl:sort</code>	138
2.4.2	Nummerierung sortierter Sequenzen	141
2.5	Schleifen	144
2.5.1	Schleifen in XSLT: <code>xsl:for-each</code>	144
2.5.2	Alternative in XPath: <code>for ... in ... return</code>	149
2.6	Gruppierung mit <code>xsl:for-each-group</code>	150
2.7	Attributwerte zur Laufzeit: Attributwert-Templates	153
2.8	Erzeugung von Knoten	159
2.8.1	Elementknoten erzeugen mit <code>xsl:element</code>	160
2.8.2	<code>xsl:attribute</code>	162
2.8.3	Eine Sammlung von Attributdeklarationen – <code>xsl:attribute-set</code>	163
2.9	Kopieren von Knoten: <code>xsl:copy</code> und <code>xsl:copy-of</code>	164
2.9.1	Kopieren mit <code>xsl:copy</code> – die flache Kopie	164
2.9.2	Kopieren mit <code>xsl:copy-of</code> – die tiefe Kopie	166
2.10	Zusammenfassung und Aufgaben	167
2.10.1	Testfragen	168
2.10.2	Aufgaben	169

3 XSLT und XPath – Runde drei 171

3.1	Variable und Parameter	173
3.1.1	Variable in XSLT – xsl:variable und xsl:param	174
3.1.2	Globale Variablen und Parameter	176
3.1.3	Variablenwerte sind temporäre Bäume	179
3.1.4	xsl:call-template, name	181
3.1.5	xsl:with-param	183
3.1.6	Tunnelparameter	185
3.1.7	Durchtunneln von Defaultregeln	190
3.2	Templates und Templatemodes	191
3.2.1	Verschieden gestimmte Templaterregeln – das mode-Attribut	191
3.3	Stringverarbeitung	193
3.3.1	Rekursiver Templateaufruf vs. regulärer Ausdruck	194
3.4	Stylesheetfunktionen	197
3.5	Externe Quellen – Dokumente und Entities	201
3.5.1	Dokumente einbinden mit document()	202
3.5.2	Die Funktion unparsed-entity-uri()	203
3.6	Mehrere Ergebnisdokumente	206
3.6.1	Erzeugung eines HTML-Framesets mit xsl:result-document	206
3.7	Modulare Stylesheets	212
3.7.1	Importpräzedenz – Konfliktbewältigung bei Importen	212
3.7.2	xsl:apply-imports	216
3.7.3	Parameter und Variable in importierten Stylesheets	218
3.8	Schlüssel und Identifier	218
3.8.1	Laufzeitidentifier für Elemente – generate-id()	219
3.8.2	Echte Identifier auslesen mit der Funktion id()	222
3.8.3	Keys – Listen von Schlüsseln zur Laufzeit	223
3.8.4	Externe keys – Schlüssel in anderen Dokumenten	226
3.8.5	Gegenüberstellung: Keys – ID	229
3.9	Zusammenfassung und Aufgaben	231
3.9.1	Testfragen	231
3.9.2	Aufgaben	233
3.10	Anstelle eines Nachworts	235

4 Referenz XPath – die Sprache 237

4.1	XPath als Pfadbeschreibungssprache	239
4.1.1	XSLT als Host-Language für XPath	239
4.1.2	Das XML-Dokument als Baum	240
4.2	XPath-Ausdrücke	241
4.2.1	Der XPath-Ausdruck als Grundeinheit	241
4.2.2	Die Sequenz als Ergebnis eines Ausdrucks	242

4.3	XPath-Ausdrücke: Auswertungskontext	244
4.3.1	Statischer Kontext	244
4.3.2	Dynamischer Kontext	246
4.4	XPath-Ausdrücke: Operatoren und Keywords	248
4.4.1	Präzedenz der XPath-Operatoren	249
4.4.2	Arithmetische Ausdrücke	250
4.4.3	Vergleichsausdrücke	252
4.4.4	Logische Verknüpfungen von Ausdrücken	257
4.4.5	Operatoren für Sequenztypen	258
4.4.6	Bedingte Ausdrücke	261
4.4.7	Quantifizierende Ausdrücke	261
4.4.8	Schleifenausdrücke mit for	264
4.4.9	Verknüpfung von Sequenzen	267
4.4.10	XPath 2.0-Kommentare	268
4.5	Pfadausdrücke: Grundlagen	270
4.5.1	Pfadausdruck in XPath 1.0 und XPath 2.0	270
4.5.2	Pfadausdrücke allgemein betrachtet	270
4.5.3	Relative und absolute Pfadausdrücke	271
4.5.4	Der Doppelslash im Pfadausdruck	271
4.6	Pfadausdrücke: Achsen	272
4.6.1	Self-Achse	275
4.6.2	Child-Achse	276
4.6.3	Descendant-Achse	277
4.6.4	Descendant-Or-Self-Achse	278
4.6.5	Following-Sibling-Achse	279
4.6.6	Following-Achse	280
4.6.7	Attribute-Achse	281
4.6.8	Namespace-Achse	283
4.6.9	Parent-Achse	284
4.6.10	Ancestor-Achse	285
4.6.11	Ancestor-Or-Self-Achse	286
4.6.12	Preceding-Sibling-Achse	287
4.6.13	Preceding-Achse	288
4.7	Pfadausdrücke: Location-Steps	289
4.7.1	Knotentests	289
4.7.2	Knotentest auf den Bezeichner: NameTest	290
4.7.3	Knotentest auf die Knotenart: KindTest	291
4.7.4	Der Achsensschritt	296
4.7.5	Die Predicates (Filterbedingungen)	302
4.7.6	Zusammenstellung der Ergebnissequenz	305
4.7.7	Innerer und äußerer Fokus in Pfad- und Predicate-Ausdrücken	306
4.8	Sequenzausdrücke	307
4.8.1	Erzeugung von Sequenzen	307
4.8.2	Sequenzenbeschreibung: Sequenztypen	309
4.9	Vergleich von XPath 1.0 und XPath 2.0	311
4.9.1	Gemeinsamkeiten und Unterschiede	311
4.9.2	Abwärtskompatibilität XPath 2.0 zu XPath 1.0	313

5 Referenz XPath-Funktionen 319

5.1	XPath- und XSLT-Funktionen allgemein	321
5.1.1	Formale Einteilung in XPath- und XSLT Funktionen	321
5.1.2	Erweiterungsfunktionen	321
5.1.3	Stylesheetfunktionen	322
5.1.4	Namensräume für Funktionen	323
5.1.5	Aufruf einer XPath/XSLT-Funktion	323
5.2	Funktionen in XPath 1.0 und XSLT 1.0	324
5.3	Funktionen in XPath 2.0 und XSLT 2.0	327
5.4	Alphabetisch geordnete Übersicht der Funktionen	334
5.4.1	fn:abs	335
5.4.2	fn:adjust-date-to-timezone	337
5.4.3	fn:adjust-dateTime-to-timezone	339
5.4.4	fn:adjust-time-to-timezone	342
5.4.5	fn:avg	345
5.4.6	fn:base-uri	347
5.4.7	fn:boolean	348
5.4.8	fn:ceiling	350
5.4.9	fn:codepoints-to-string	352
5.4.10	fn:collection	353
5.4.11	fn:compare	354
5.4.12	fn:concat	357
5.4.13	fn:contains	361
5.4.14	fn:count	364
5.4.15	current	365
5.4.16	fn:current-date	369
5.4.17	fn:current-dateTime	370
5.4.18	current-group	371
5.4.19	current-grouping-key	372
5.4.20	fn:current-time	373
5.4.21	fn:data	374
5.4.22	fn:day-from-date	375
5.4.23	fn:day-from-dateTime	376
5.4.24	fn:days-from-dayTimeDuration	377
5.4.25	fn:deep-equal	379
5.4.26	fn:default-collation	382
5.4.27	fn:distinct-values	383
5.4.28	fn:doc	386
5.4.29	document	388
5.4.30	fn:document-uri	392
5.4.31	element-available	393
5.4.32	fn:empty	395
5.4.33	fn:ends-with	396
5.4.34	fn:error	398
5.4.35	fn:escape-uri	399
5.4.36	fn:exactly-one	403
5.4.37	fn:exists	405
5.4.38	fn:expanded-QName	406

5.4.39	fn:false	407
5.4.40	fn:floor	408
5.4.41	format-date	410
5.4.42	format-dateTime	424
5.4.43	format-number	429
5.4.44	format-time	433
5.4.45	function-available	438
5.4.46	generate-id	441
5.4.47	fn:hours-from-dateTime	443
5.4.48	fn:hours-from-dayTimeDuration	445
5.4.49	fn:hours-from-time	446
5.4.50	fn:id	448
5.4.51	fn:idref	452
5.4.52	fn:implicit-timezone	455
5.4.53	fn:index-of	456
5.4.54	fn:in-scope-prefixes	458
5.4.55	fn:insert-before	459
5.4.56	key	461
5.4.57	fn:lang	463
5.4.58	fn:last	466
5.4.59	fn:local-name	467
5.4.60	fn:local-name-from-QName	469
5.4.61	fn:lower-case	469
5.4.62	fn:matches	472
5.4.63	fn:max	477
5.4.64	fn:min	480
5.4.65	fn:minutes-from-dateTime	483
5.4.66	fn:minutes-from-dayTimeDuration	484
5.4.67	fn:minutes-from-time	486
5.4.68	fn:month-from-date	487
5.4.69	fn:month-from-dateTime	488
5.4.70	fn:months-from-year/MonthDuration	489
5.4.71	fn:name	491
5.4.72	fn:namespace-uri	494
5.4.73	fn:namespace-uri-from-QName	497
5.4.74	fn:namespace-uri-for-prefix	497
5.4.75	fn:node-name	498
5.4.76	fn:normalize-space	499
5.4.77	fn:normalize-unicode	501
5.4.78	fn:not	504
5.4.79	fn:number	506
5.4.80	fn:one-or-more	509
5.4.81	fn:position	510
5.4.82	regex-group	511
5.4.83	fn:remove	513
5.4.84	fn:replace	514
5.4.85	fn:resolve-QName	519
5.4.86	fn:resolve-uri	522
5.4.87	fn:reverse	524
5.4.88	fn:root	525
5.4.89	fn:round	526
5.4.90	fn:round-half-to-even	528
5.4.91	fn:seconds-from-dateTime	531

5.4.92	fn:seconds-from-dayTimeDuration	532
5.4.93	fn:seconds-from-time	534
5.4.94	fn:starts-with	535
5.4.95	fn:string	538
5.4.96	fn:string-join	542
5.4.97	fn:string-length	545
5.4.98	fn:string-to-codepoints	547
5.4.99	fn:subsequence	547
5.4.100	fn:substring	550
5.4.101	fn:substring-after	553
5.4.102	fn:substring-before	557
5.4.103	fn:subtract-dateTimes-yielding-dayTimeDuration	560
5.4.104	fn:subtract-dateTimes-yielding-yearMonthDuration	562
5.4.105	fn:sum	563
5.4.106	system-property	567
5.4.107	fn:timezone-from-date	570
5.4.108	fn:timezone-from-dateTime	570
5.4.109	fn:timezone-from-time	571
5.4.110	fn:tokenize	572
5.4.111	fn:trace	576
5.4.112	fn:translate	577
5.4.113	fn:true	580
5.4.114	fn:unordered	581
5.4.115	fn:upper-case	583
5.4.116	unparsed-entity-public-id	585
5.4.117	unparsed-entity-uri	587
5.4.118	unparsed-text	588
5.4.119	fn:year-from-date	591
5.4.120	fn:year-from-dateTime	592
5.4.121	fn:years-from-yearMonthDuration	593
5.4.122	fn:zero-or-one	594
5.5	XPath 2.0-Konstruktorfunktionen (Übersicht)	595
5.6	XPath 2.0-Typumwandlungsfunktionen (Casting Functions)	598
5.6.1	Umwandlung zwischen primitiven Typen	598
5.6.2	Umwandlung von abgeleiteten Typen in Basistypen	604
5.6.3	Umwandlung innerhalb eines Typhierarchiezweigs	605
5.6.4	Umwandlung zwischen Typhierarchiezweigen	605
5.6.5	Ausgehend von xs:string und xs:anySimpleType	605
5.6.6	Nach xs:string, xs:anySimpleType und xdt:untypedAtomic	606
5.6.7	Nach numerischen Typen	607
5.6.8	Nach Typen der Zeitdauer	607
5.6.9	Nach Typen für Datum und Zeit	608
5.6.10	Nach xs:boolean	608
5.6.11	Nach xs:base64Binary und xs:hexBinary	608
5.6.12	Nach xs:anyURI	608
5.6.13	Nach xs:QName	609
5.7	XPath 2.0-Operatorfunktionen	609

6 Referenz XSLT-Elemente 617

6.1	Formale Einteilung der XSLT-Elemente	619
6.1.1	Root-Elemente	619
6.1.2	Toplevel-Elemente	620
6.1.3	Instruktionen	620
6.2	Kurzübersicht über XSLT 1.0	622
6.2.1	Übersicht über die Elemente	622
6.2.2	Standardattribute in XSLT 1.0	623
6.2.3	Attributwert-Templates in Instruktionen von XSLT 1.0	624
6.3	Kurzübersicht über XSLT 2.0	624
6.3.1	Übersicht über die Elemente von XSLT 2.0	624
6.3.2	Zusätzliche Standardattribute ab XSLT 2.0	627
6.3.3	Attributwert-Templates in Instruktionen von XSLT 2.0	630
6.4	Die XSLT-Elemente nach Funktionsgruppen	631
6.5	Alphabetische Referenz der XSLT-Elemente	633
6.5.1	xsl:analyze-string	633
6.5.2	xsl:apply-imports	640
6.5.3	xsl:apply-templates	645
6.5.4	xsl:attribute	649
6.5.5	xsl:attribute-set	656
6.5.6	xsl:call-template	660
6.5.7	xsl:character-map	664
6.5.8	xsl:choose	666
6.5.9	xsl:comment	669
6.5.10	xsl:copy	670
6.5.11	xsl:copy-of	677
6.5.12	xsl:decimal-format	682
6.5.13	xsl:document	686
6.5.14	xsl:element	690
6.5.15	xsl:fallback	695
6.5.16	xsl:for-each	698
6.5.17	xsl:for-each-group	702
6.5.18	xsl:function	712
6.5.19	xsl:if	719
6.5.20	xsl:import	722
6.5.21	xsl:import-schema	732
6.5.22	xsl:include	734
6.5.23	xsl:key	737
6.5.24	xsl:matching-substring	741
6.5.25	xsl:message	744
6.5.26	xsl:namespace	746
6.5.27	xsl:namespace-alias	749
6.5.28	xsl:next-match	754
6.5.29	xsl:non-matching-substring	759
6.5.30	xsl:number	761
6.5.31	xsl:otherwise	774
6.5.32	xsl:output	777
6.5.33	xsl:output-character	799

6.5.34	xsl:param	801
6.5.35	xsl:perform-sort	813
6.5.36	xsl:preserve-space	817
6.5.37	xsl:processing-instruction	819
6.5.38	xsl:result-document	824
6.5.39	xsl:sequence	832
6.5.40	xsl:sort	835
6.5.41	xsl:strip-space	849
6.5.42	xsl:stylesheet	851
6.5.43	xsl:template	863
6.5.44	xsl:text	875
6.5.45	xsl:transform	880
6.5.46	xsl:value-of	881
6.5.47	xsl:variable	887
6.5.48	xsl:when	897
6.5.49	xsl:with-param	899
6.6	Vorwärts- und Rückwärtskompatibilität	906
6.6.1	Rückwärtskompatible Verarbeitung	907
6.6.2	Vorwärtskompatible Stylesheetverarbeitung	907

7 Referenz Saxon 7 909

7.1	Die Installation	911
7.2	Ein Basic XSLT Processor	912
7.3	Betrieb von der Kommandozeile	913
7.3.1	Aufruf mit Quelldokument und Stylesheet	913
7.3.2	Aufruf mit Stylesheet-PI im Quelldokument	914
7.3.3	Aufruf mit Nennung eines Ergebnisdokuments	915
7.3.4	Aufruf mit Parameterübergabe	916
7.3.5	Verarbeitung kompletter Ordnerinhalte	917
7.4	Weitere Optionen für die Kommandozeile	917
7.5	Kompilierte Stylesheets	923

8 XSLT-Entwicklungsumgebungen 927

8.1	Architag XRay	930
8.2	XML Cooktop	932
8.3	Treebeard	936
8.4	NetBeans	942
8.5	Visual XSLT	946
8.6	<oXygen/>-XML-Editor	949
8.7	Komodo	954
8.8	Sonic Stylus Studio 5.1	961
8.9	Altova XML-Spy 2004	966

A	Lösungen	973
A.1	Antworten zu den Testfragen Kapitel 1	973
A.2	Antworten zu den Testfragen Kapitel 2	975
A.3	Antworten zu den Testfragen Kapitel 3	977
A.4	Lösungen zu den Aufgaben Kapitel 1	979
A.5	Lösungen zu den Aufgaben Kapitel 2	990
A.6	Lösungen zu den Aufgaben Kapitel 3	996
B	W3C-Online-Ressourcen	1001
B.1	Ressourcen zu XML	1001
B.2	Ressourcen zu Namensräumen	1002
B.3	Ressourcen zu XML Schema	1002
B.4	Ressourcen zu XSL und XSLT	1003
B.5	Ressourcen zu XPath und XQuery	1005
B.6	Ressourcen zu Data Model und Infoset	1007
B.7	Weitere Ressourcen	1007
C	Glossar	1009
	Index	1043

Einleitung

Warum ein Buch über XSLT? Wie ist dieses Buch aufgebaut und an wen richtet es sich? Welche Themen werden behandelt? Welche Inhalte werden nicht abgedeckt? Welche Darstellungskonventionen werden verwendet? Was befindet sich auf der Begleit-CD?

Ein beinahe 1000 Seiten starkes Werk über eine Sprache zu schreiben, deren Aufgabe im Wesentlichen die Umwandlung eines XML-Formats in ein anderes ist, verlangt möglicherweise nach einer Begründung. Hätte das nicht ein wenig kürzer, kompakter ausfallen können? Ist das Thema denn überhaupt so wichtig, um diesen Umfang zu rechtfertigen?

Warum ein dickes Buch über XSLT?

Nun, abgesehen davon, dass die Arbeit an diesem Buch mir – zumindest die meiste Zeit über – viel Spaß gemacht hat, gibt es auf diese Fragen noch einige weitere Antworten. Der am Anfang einschüchternde Umfang ergibt sich primär aus dem in der zweiten Version von XSLT und XPath erheblich gesteigerten Sprachumfang. Ein Großteil des Buches ist deshalb den Referenzen der einzelnen Instruktionen von XSLT, den Erläuterungen zu den Funktionen von XPath sowie einem abschließenden, umfassenden Glossar der in diesem Umfeld verwendeten Fachausdrücke und Abkürzungen gewidmet.

Der erste Teil des Buches gibt eine didaktische Einführung in das Thema XSLT und XPath. Die Trennung in einen didaktischen Teil und eine Referenz beruht auf der Überlegung, dass eine vollständige Diskussion der jeweils im Stoff auftauchenden Elemente und Konzepte innerhalb eines einführenden Textes nicht möglich sein würde, ohne die Lesbarkeit und Dynamik eben dieser Einführung zu beeinträchtigen. Der Referenzteil erfordert zudem eine umfassende Detaildarstellung und damit einen eher lexikalischen Stil, der beim didaktischen Teil fehl am Platz wäre – dieser darf und soll auch Spaß machen, und muss deshalb auslassen und vereinfachen dürfen, wenn dies der Darstellung des zu vermittelnden Stoffes dient.

Warum überhaupt ein Buch über XSLT?

Eine simple Antwort auf diese Frage lautet natürlich, dass dieses Thema »wichtig« ist – immerhin ist XSLT ein unabhängiger Standard zur Informationsverarbeitung und gehört zur XML-Familie, ist also naturgemäß Favorit für die Verarbeitung von in XML-Format vorliegenden Daten. Nicht ohne Grund, denn genau hierfür wurde XSLT konzipiert.

XML selbst ist über das Stadium des Hypes und reinen Buzzwords inzwischen weit hinaus und hat sich zum anerkannten Standard der industriellen Informationsspeicherung, -weitergabe und -verarbeitung gemauert. Der Sprachstandard wurde vom W3C unter Mitwirkung zahlreicher Fachleute aus der Industrie definiert, was eine breite, inzwischen geradezu allgegenwärtige Unterstützung der Sprache gewährleistet.

Mit der Verbreitung von XML steigt die Notwendigkeit einer Verarbeitung entsprechender Dokumente – auch diese sollte plattformunabhängig erfolgen. Es ist zwar denkbar, eine Verarbeitung von XML in verschiedensten Programmiersprachen zu schreiben, jedoch dann ohne die Möglichkeit, die so erstellten Verarbeitungsvorschriften einfach portieren zu können.

XSLT stellt in dieser Hinsicht die Lösung dar: als plattformunabhängige, selbst in XML geschriebene Anwendungen sind XSLT-Stylesheets so gut wie problemlos von einer Einsatzumgebung in die andere portierbar, sei es aus einer PHP-Umgebung, in der die Verarbeitung mit Sablotron erfolgt, aus der NET-Umgebung oder einer Java-Umgebung von Cocoon. Wo die Umgebungen wechseln, kann das XSLT-Stylesheet gewissermaßen eine Konstante darstellen.

Die Aufgaben, die XSLT dabei zu bewältigen hat, werden zunehmend komplexer. Heutzutage spielt sich ein Großteil der XML-Verarbeitung in Zusammenhang mit Datenbanken ab, die Gewährleistung einer korrekten Verarbeitung in Bezug auf Gültigkeit sowohl der verarbeiteten Daten als auch der Ergebnisse wirft neue Schwierigkeiten auf, denen mit der neuen Version 2.0 von XSLT und XPath Rechnung getragen wird.

Die Aufeinanderzubewegung von Datenbanken und XML wird an der Verschmelzung von XPath und der als Abfragesprache konzipierten XML-Query Language zur allgemeinen XML-Abfragesprache XQuery 1.0 deutlich, von der XPath 2.0 »nur« eine Untergruppe darstellt. Im Zentrum der Verarbeitung von Daten in XML-Form, gleichgültig, ob diese nun als »reale« Dokumente vorliegen oder als dynamisch mittels Datenbankabfrage erzeugte »virtuelle« Dokumente, wird stets XSLT stehen.

Welche Inhalte werden behandelt?

Dieses Buch beschäftigt sich gleichrangig mit **XSLT** und **XPath** – beide Sprachen werden in der Regel gemeinsam verwendet (XPath kann allerdings durchaus auch in anderen Umgebungen auftauchen), sodass eine Trennung nicht sinnvoll ist.

Sowohl XPath als auch XSLT werden in ihrer aktuellen **Version 2.0** vorgestellt. Auf Änderungen gegenüber XSLT 1.0 wird bei der Behandlung der Instruktio-

nen von XSLT 2.0 hingewiesen; der Versionsprung stellt im Wesentlichen eine Erweiterung gegenüber der Vorgängerversion dar. **XPath 2.0/XQuery 1.0** ist hingegen eine fast neu konzipierte, lediglich »rückwärtskompatible« Sprache – sowohl was den erheblich erweiterten Sprachumfang als auch das Verhalten der einschlägigen Funktionen betrifft.

Dieses Buch soll ausdrücklich keine rückwirkende Einführung in **XSLT 1.0** und **XPath 1.0** sein. Vorkenntnisse dieser Versionen beider Sprachen werden allerdings nicht vorausgesetzt. Nach Durcharbeitung des Buches sollten Sie sich ebenfalls in einer XSLT 1.0/XPath 1.0-Umgebung leicht zurecht finden können – für »Umsteiger« weist der Referenzteil auf entsprechende Unterschiede zwischen beiden Versionen und denkbare Klippen hin.

Kenntnisse in **XML** und zumindest rudimentäre Kenntnisse in **XML Schema** sollten beim Leser vorhanden sein. Eine Behandlung hätte einerseits den ohnehin umfangreichen Rahmen gesprengt, auf der anderen Seite existiert Literatur hierzu bereits in hinreichender Menge.

Ebenfalls nicht Thema dieses Buches konnte die vollständige Spezifikation von **XQuery** sein. XPath 2.0 stellt zwar von dieser eine nicht unwesentliche Teilmenge dar, der ganze Sprachumfang von XQuery 1.0 kann jedoch (zumindest zur gegenwärtigen Zeit) nur separat von XSLT eingesetzt werden. Da dieses Buch aus der Sicht von XSLT geschrieben ist, stellte eine Beschreibung der dort nicht nutzbaren Funktionalitäten von XQuery m. E. lediglich Ballast dar, der bestenfalls zur Verwirrung führen würde.

Hinweis Wer keine Erfahrungen bzw. Kenntnisse in XML oder XML Schema besitzt, sollte entsprechende Einführungswerke zur Hand nehmen (z. B. »Einführung in XML« von Helmut Vonhoegen, Galileo Computing oder vergleichbare). Die Erarbeitung von Grundkenntnissen in XML und XML Schema ist nicht Teil dieses Buches, sondern diese müssen in gewissem Rahmen vorausgesetzt werden. Wo immer möglich, wird auf versteckte Problematiken eingegangen.

Welche Darstellungskonventionen werden verwendet?

Ziel war es, den Text gut lesbar zu erhalten, ohne ein Übermaß an zusätzlichen, im Fließtext verwendeten Formatierungen. Sofern erforderlich oder sinnvoll, werden **Schlüsselbegriffe**, die von zentraler Bedeutung sind oder einer schnellen Sinnerfassung dienen, bei ihrem ersten Auftreten **fett** markiert.

Da eine Übersetzung aller erforderlichen **Fachbegriffe** nicht in jedem Fall eindeutig bzw. festgelegt ist, wird bei der Einführung eines solchen Begriffes der

Klarheit halber oft auch die englischsprachige Originalbezeichnung (*in Klammern, kursiv*) beigefügt.

Die Bezeichner von Instruktionen und Funktionen sowie kurze, im Fließtext auftretende Quelltextbeispiele werden wie hier in nicht-proportionaler Schrift dargestellt.

Dasselbe gilt für die unvermeidlichen längeren **Quelltextbeispiele**:

```
<quelltext-beispiel>
  <!--
    Längere Quelltextpassagen werden als separate Blöcke
    abgesetzt. Soll innerhalb dieser auf
    einzelne Elemente besonders hingewiesen werden,
    so werden diese fett markiert.
  -->
</quelltext-beispiel>
```

Wichtige Hinweise

Wichtige Hinweise werden, wie hier, durch grau unterlegte Boxen hervorgehoben. Es kann sich hierbei um eine wiederholende Kurzdarstellung von Schlüsselkonzepten oder um kontextspezifische Anmerkungen handeln.

Was befindet sich auf der Begleit-CD?

Neben den Quelltextbeispielen, die in diesem Buch verwendet werden, befinden sich auf der CD die in Kapitel 8, *XSLT-Entwicklungsumgebungen*, vorgestellten Programme – soweit sich dies realisieren ließ. Wo eine Aufnahme in die CD aus rechtlichen oder platztechnischen Gründen nicht möglich war, sind Links zum Download angegeben.

An dieser Stelle muss ich allerdings darauf hinweisen, dass die gegenwärtig (Mai 2004) erhältlichen kommerziellen Programme nur den Sprachumfang von XSLT 1.0 in Verbindung mit XPath 1.0 unterstützen. Dies ist auch nicht verwunderlich ist, da XSLT 2.0 einen völlig neuen Standard darstellt. Mit einer Adaption von XSLT 2.0 ist seitens der großen Anbieter jedoch spätestens zum Jahresende 2004 zu rechnen.

Der einzige XSLT-Prozessor, der XSLT 2.0 in vollem Umfang implementiert, ist Michael Kays **Saxon 7.x**. Dieser Prozessor ist ebenfalls Teil der aktuellen Version von Rob Rohans Open Source XSLT-Umgebung **Treebeard** (bei den XSLT-Editoren beschrieben), die hiermit tauglich für XSLT 2.0 ist, wenn auch ohne Unterstützung von XML Schema. Die meisten Beispiele dieses Buches sind somit in Treebeard nachvollziehbar.

Über mehr oder minder einfache Umwege lässt sich Saxon 7 auch in die anderen aktuellen Umgebungen einbeziehen, sodass auch in diesen – mit unterschiedlich starken Abstrichen zum ansonsten gebotenen Komfort – testweise eine XSLT 2.0-Entwicklung möglich ist. Das hierfür jeweils erforderliche Vorgehen wird im betreffenden Kapitel beschrieben.

Danksagungen

Kein Buch schreibt sich von alleine oder entsteht im luftleeren Raum. Dieses hier wäre nicht möglich gewesen ohne die Arbeitsgruppen des W3C, denen es gelang, mit XSLT 2.0, XPath 2.0 und den weiteren, hiermit verbundenen Themen eine zwar höchst komplexe, aber in sich logische Spezifikation zu schaffen. Zum Zeitpunkt des Erscheinens dieses Buches wird XSLT 2.0 noch eine sehr, sehr neue Sprache sein – allerdings eine mit einer großen Zukunft.

Dieses Buch wäre aber auch nicht entstanden ohne die Unterstützung einiger Menschen, denen ich hier ausdrücklich danken möchte. An erster Stelle natürlich meiner Frau, Carina Prange, die mir verständnisvoll den Rücken frei hielt und es mir so ermöglichte, einigermaßen guten Gewissens die erforderliche Zeit zu investieren. An zweiter Stelle meinem Lektor bei Galileo Computing, Stephan Mattescheck, ohne dessen Geduld und stetige Aufmunterung ich dieses Projekt vermutlich nie zu Ende gebracht hätte.

Last not least mein Dank an alle Leser dieses Vorworts; besonders an die, die sonst keine Vorworte lesen. Vielen Dank für Ihre Geduld. Ich hoffe, dass dieses Buch Ihnen einen spannenden und kurzweiligen Einstieg in die Welt von XSLT eröffnen wird. So, nun reicht es aber – gehen wir an die Arbeit!

Frank Bongers

Berlin, Mai 2004

1 XSLT und XPath – der Start

1.1	Die Themenverortung.....	25
1.2	Eine erste Transformation	27
1.3	Einteilung der Elemente von XSLT.....	32
1.4	Das Wurzelement <code>xsl:stylesheet</code>	32
1.5	Die Grundstruktur des Stylesheets: <code>xsl:template</code>	35
1.6	Literal Result Elements unter der Lupe	37
1.7	Die Instruktionen im Templaterumpf.....	41
1.8	Das XML-Dokument als Baum	43
1.9	Eine Transformation Schritt für Schritt.....	50
1.10	Start mit XPath: Pfadausdrücke.....	53
1.11	Von Achsen und Nodetests	58
1.12	Pfadausdrücke – Beispiele	61
1.13	Von Stringwerten und Attributwerten	66
1.14	Stylesheets mit mehreren Templates	69
1.15	Ein Vergleich – Ausdruck vs. Instruktion	75
1.16	Struktur steuert Verarbeitung	86
1.17	Mehr Kontrolle über das Ergebnis.....	94
1.18	Zusammenfassung und Aufgaben.....	101

1 XSLT und XPath – der Start

2 XSLT und XPath – Runde zwei

3 XSLT und XPath – Runde drei

4 Referenz XPath – die Sprache

5 Referenz XPath-Funktionen

6 Referenz XSLT-Elemente

7 Referenz Saxon 7

8 XSLT-Entwicklungsumgebungen

A Lösungen

B W3C-Online-Ressourcen

C Glossar

1 XSLT und XPath – der Start

Dieses Kapitel dient als allgemeiner Einstieg in XSLT: Anhand einfacher Beispiele werden die Grundstruktur eines XSLT-Stylesheets und erste XSLT-Instruktionen erläutert. Weiterhin wird ein Blick auf Baumstrukturen und die Bedeutung von XPath im Verhältnis zu XSLT geworfen.

Da dieses Buch sich beinahe ausschließlich mit **XSLT** beschäftigen wird, ist dies der geeignete Platz, um zunächst kurz das Thema zu umreißen. Beginnen wir also beim Anfang, zunächst ohne uns um »technische« Aspekte zu kümmern ...

1.1 Die Themenverortung

Ein **XML-Dokument** beschreibt die in ihm enthaltene Information lediglich in ihrer Struktur. Daher muss diese, z. B. zu **Präsentationszwecken**, in eine hierfür geeignete Form gebracht werden. Oft muss dies mit einer **Umformung** der Informationsstruktur einhergehen, beispielsweise deren Sortierung oder Filterung. Auch für den **Austausch von Informationen** in XML-Format, wie es im B2B-Bereich erforderlich ist, müssen XML-Quellformate in anders strukturierte Zielformate überführt werden. Hierbei kann es erforderlich sein, neue Inhalte hinzuzufügen, die im Dokument selbst nicht enthalten sind – sie können aus anderen Quellen stammen oder durch Zusammenfassung oder Berechnungen aus den Ursprungsinformationen gewonnen werden.

Das ursprüngliche Dokument bleibt dabei erhalten: Aus einem **Quelldokument** wird stets ein neues **Ergebnisdokument** erzeugt. Dieses kann wiederum XML-Format, aber auch ein (fast beliebiges) anderes Format besitzen.

Diese Art der Verarbeitung von XML-Dokumenten, wie sie hier betrachtet werden soll, wird mittels **XSL**, der **eXtensible Style Language** beschrieben, bei der es sich um eine XML-Anwendung handelt. XSL zerfällt eigentlich in zwei Untergruppen, von denen die eine, **XSL-FO** (XSL Formatting Objects¹), die Druckpräsentation zum Ziel hat. Die andere, die uns vorwiegend interessiert und die deshalb Thema dieses Buchs ist, beschreibt ausschließlich die **Transformation** von XML-Dokumenten, also deren Umformung: **XSLT**, die **eXtensible Style Language Transformations**. In XSLT formulierte Dokumente

1 Über XSL-FO müsste ein eigenes Buch geschrieben werden. Für uns interessant mag aber die Tatsache sein, dass der Weg von einem XML-Dokument zu dessen FO-Repräsentation wiederum über eine Transformation mit XSLT führt, XSL-FO also XSLT nachgeschaltet sein kann.

bezeichnet man aus historischen Gründen als **Stylesheets**, obwohl sie funktional weit über eine statische Darstellungsbeschreibung hinausgehen.

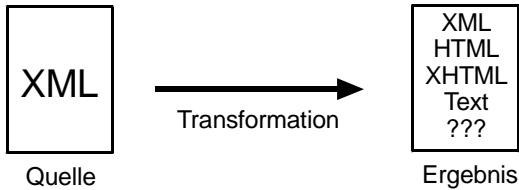


Abbildung 11 Quell- und Ergebnisdokument

► Was bedeutet XSLT?

Die Abkürzung **XSLT** steht für **eXtensible Stylesheet Language Transformations**. XSLT gehört zu den deklarativen Sprachen und beschreibt und steuert die Umwandlung (Transformation) von XML-Dokumenten in andere Formate wie HTML, XHTML, Text oder andere XML-Strukturen.

Obwohl das Buch den Titel »XSLT« trägt, wird genauso oft von **XPath** die Rede sein. In XSLT eingebettet dienen XPath-Ausdrücke sowohl der Steuerung der eigentlichen Transformation als auch der unmittelbaren Erzeugung von Inhalten. XPath ist eine **Pfadbeschreibungssprache**, in der sich der Weg zu einer, in einem XML-Dokument befindlichen Information formulieren lässt.

► Was bedeutet XPath?

Die Langform des Begriffes XPath ist **XML Path Language**. XPath dient der Navigation innerhalb von Dokumentebäumen und kann beliebige Knotenmengen eines solchen Baums benennen sowie die dort liegenden Informationen zugänglich machen.

In Zusammenhang mit **XSLT 2.0** wird stets auch **XPath 2.0** verwendet;² analog arbeitet **XSLT 1.0** ausschließlich mit **XPath 1.0**. Andere Kombinationen sind ausdrücklich nicht möglich.

² Bei XPath 2.0 handelt es sich eigentlich um eine (allerdings wesentliche) Untergruppe von XQuery, einer äußerst mächtigen Abfragesprache für XML, die in voller Funktionalität jedoch nur unabhängig von XSLT nutzbar ist.

Eine **Kompatibilität** zwischen XSLT 1.0 und XSLT 2.0 ist relativ weit reichend gegeben: Sämtliche Elemente von XSLT 1.0 existieren auch in XSLT 2.0, zum Teil jedoch mit verändertem Verhalten oder erweiterter Funktionalität. Dies hat unter anderem mit dem gegenüber XPath 1.0 wesentlich erweiterten Umfang von XPath 2.0 zu tun, jedoch auch mit dem einen, wesentlichen Unterschied von XSLT 2.0 gegenüber XSLT 1.0: der Unterstützung von **Typisierung** der verarbeiteten Information gemäß **XML Schema**.

Wir werden uns in der Folge mit XSLT 2.0 und XPath 2.0 beschäftigen, die ich der Kürze halber deshalb lediglich mit »XSLT« und »XPath« bezeichnen werde.

Hinweis: Installieren Sie jetzt Ihren XSLT-Prozessor

Die auf den folgenden Seiten beschriebenen Beispiele erschließen sich auf Grund ihrer Einfachheit durchaus auch auf theoretischem Weg. Wenn Sie die Transformationen jedoch sofort in der Praxis durchführen wollen, sollten Sie zunächst den XSLT-Prozessor **Saxon 7** auf Ihrem Rechner installieren – eventuell in Zusammenhang mit einer XML-Entwicklungsumgebung Ihrer Wahl. Alle erforderlichen Installationsdateien und die vorgestellten Beispieldateien sind auf der Begleit-CD des Buches enthalten. Eine entsprechende **Hilfestellung** zur Installation und Verwendung von Saxon 7 bzw. zu den Entwicklungsumgebungen finden Sie im Referenzteil des Buches.

1.2 Eine erste Transformation

Anstatt hier lange theoretische Erörterungen zu beginnen, sollen einige möglichst einfache Beispiele vorgestellt werden, an denen sich aber bereits in Ansätzen zeigen lässt, worum es bei der Anwendung von XSLT geht.

1.2.1 Drei beteiligte Dokumente

Bei jeder Transformation sind drei Dokumente beteiligt – zunächst das **Quelldokument**, bei dem es sich um ein XML-Dokument handelt und das die Informationen beinhaltet. Zweitens das **XSLT-Stylesheet**: In diesem stehen die Anweisungen, wie mit den Informationen aus dem Quelldokument zu verfahren ist. Drittens das im Verlauf der Transformation erzeugte **Ergebnisdokument** – sein Aufbau ergibt sich gleichermaßen aus den Anweisungen des Stylesheets als aus Struktur und Inhalt des Quelldokuments.

Das Quelldokument

Ein Quelldokument muss für die Durchführung einer Transformation zwingend vorliegen – hier wird eine möglichst einfache XML-Datei verwendet, um

den Blick nicht durch nebensächliche Aspekte abzulenken. Ein leeres Wurzelement genügt vorläufig bereits:

```
<?xml version="1.0"?>
<quelldokument/>
```

Listing 1.1 beispiel1.xml (alle derart markierten Beispiele: siehe Begleit-CD)

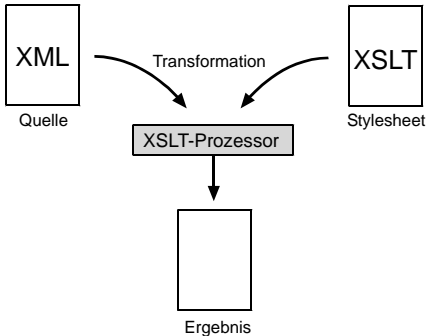


Abbildung 1.2 Schematisch dargestellte Transformation

Das Stylesheet

Zur Verarbeitung wird natürlich auch ein XSLT-Stylesheet benötigt (auch dieses Dokument ist für das Beispiel so einfach wie möglich gehalten). Deutlich ist zu sehen, dass einige Tagnamen mit `xsl:` beginnen – diese bilden das funktionale Gerüst des Stylesheets. Darüber hinaus ist noch »normales« HTML enthalten:

```
<?xml version="1.0"?>
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
    <html>
    <body>
    <h1 align="center">Dies ist ein Ergebnisdokument.</h1>
    </body>
    </html>
  </xsl:template>
</xsl:stylesheet>
```

Listing 1.2 beispiel1.xsl

Das Ergebnisdokument

Folgendes Ergebnisdokument wird erzeugt – in diesem Fall handelt es sich um ein simples HTML-Dokument:

```

<html>
<body>
<h1 align="center">Dies ist ein Ergebnisdokument.</h1>
</body>
</html>

```

Listing 1.3 beispiel.html

Vergleicht man Ergebnisdokument und Stylesheet, so bemerkt man, dass das Ergebnisdokument als Ganzes – wenn auch in weitere Elemente verkapselt – bereits im Stylesheet enthalten war (dort im Quellcode fett ausgezeichnet).

Dies ist in der hier dargestellten Absolutheit natürlich ein Extremfall. Ebenso extrem an diesem Beispiel ist, dass es so scheint, als würde das XML-Quelldokument im Prinzip gar nicht verwendet (was kein Wunder ist, denn es steht nichts drin). Eine gewisse Rolle spielt es dennoch, wie später zu sehen sein wird.

1.2.2 Kopierte Inhalte: Literal Result Elements

Zunächst soll nur einfach vermerkt werden, dass ein Stylesheet Bereiche enthält, die »wie sie sind« im Ergebnisdokument wieder auftauchen. Weil sie dabei nicht verändert werden, also »literal« übernommen werden, bezeichnet man sie auch als **Literal Result Elements** (literale Ergebniselemente).

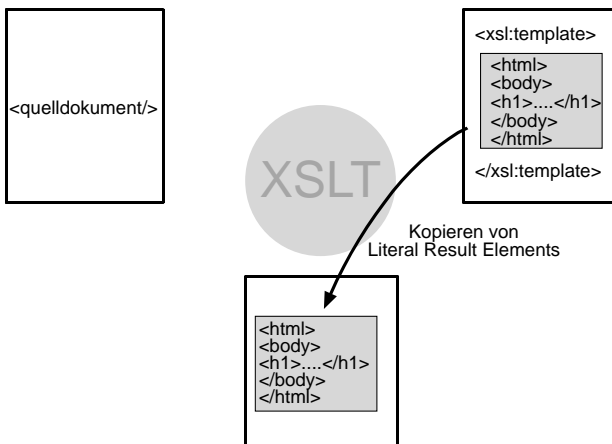


Abbildung 1.3 Kopieren der LRE ins Ergebnisdokument

1.2.3 Erster Einsatz von XSLT-Instruktionen

Das Beispiel wird jetzt erweitert, indem ein XML-Dokument verwendet wird, das verwertbare Information enthält. Im vorliegenden Fall handelt es sich um

eine Zeichenkette, die ins Ergebnisdokument kopiert werden soll. Um das Stylesheet dazu zu veranlassen, muss es gegenüber dem vorigen Beispiel ein wenig ausgebaut werden.

Hier zunächst das neue Quelldokument:

```
<?xml version="1.0"?>
<gruss>Hallo Welt! Dies ist XSLT.</gruss>
```

Listing 14 beispiel.xml

Für das zu erzeugende HTML-Dokument (vorläufig bleibt es bei HTML als Ergebnisformat) wird folgende Form angestrebt:

```
<html>
<body>
<h1 align="center">Dies ist ein Ergebnisdokument.</h1>
<p align="center">Hallo Welt! Dies ist XSLT.</p>
</body>
</html>
```

Listing 15 beispiel.html

Das Ergebnis wird also einen `<p>`-Container mit dem Text enthalten, der im Quelldokument im Element `<gruss>` steht. Das benötigte Stylesheet könnte beispielsweise so aussehen:³

```
<?xml version="1.0"?>
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
    <html>
      <body>
        <h1 align="center">Dies ist ein Ergebnisdokument.</h1>
        <p align="center"><xsl:value-of select="gruss"/></p>
      </body>
    </html>
  </xsl:template>
</xsl:stylesheet>
```

Listing 16 beispiel.xsl

³ Selbst für dieses einfache Beispiel wären bereits andere, gleichwertige Lösungen denkbar, da auch XSLT – wie jede Programmiersprache – für ein Problem unterschiedliche Ansätze ermöglicht.

Das Stylesheet wurde um ein weiteres Literal Result Element ergänzt, nämlich den `<p>`-Container, wie er im Ergebnisdokument erscheinen soll.

Nun folgt etwas, was das bisher statische Stylesheet »aktiv« werden lässt und gleichzeitig erstmals das Quelldokument in die Verarbeitung einbezieht:

Im Inneren dieses `<p>`-Containers steht eine **XSLT-Instruktion**. Die Anweisung `<xsl:value-of select="gruss"/>` extrahiert den gewünschten Inhalt aus dem XML-Dokument und fügt ihn ins Ergebnisdokument ein:

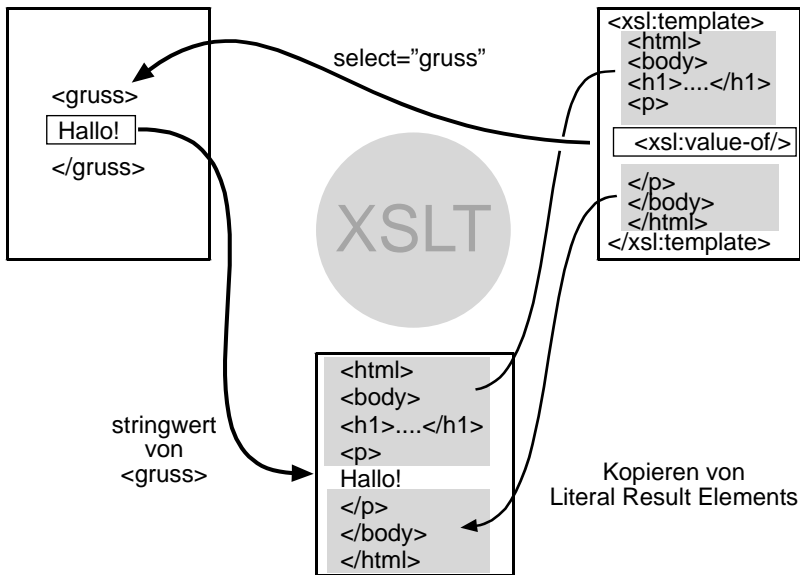


Abbildung 1.4 `xsl:value-of` holt sich die Information aus XML-Dokument

Neben `xsl:value-of`, das wie gesehen einen Befehl darstellt, erscheinen noch zwei weitere XSLT-Elemente: `xsl:stylesheet`, mit dem das Styleshestdokument beginnt (der XML-Prolog wird der Einfachheit halber außer Acht gelassen), und `xsl:template`, welches eine Hierarchieebene weiter innen steht und in dessen Inneren sich die Literal Result Elements befinden.

Auch an dieser Stelle ein kleiner vorläufiger Lehrsatz: Im Inneren der Templateelemente eines XSLT-Stylesheets ist es erlaubt zu mischen: Literal Result Elements, die unverändert kopiert werden (wie hier der `<p>`-Container), und XSLT-Instruktionen dürfen nebeneinander vorkommen.

Hinweis: Alle Codebeispiele, zu denen in der Unterschrift ein Dateiname genannt wird, befinden sich auf der Begleit-CD.

1.3 Einteilung der Elemente von XSLT

Es wurde in den vorangegangenen kurzen Beispielen bereits ansatzweise deutlich, dass Position und Auftreten der XSLT-Anweisungen im Stylesheet festen Regeln unterliegen, sie in diesem Sinne also nicht zueinander gleichrangig sind. Man teilt sie ihrer Stellung in der Stylesheet-Hierarchie entsprechend in drei **Gruppen** ein:

- ▶ **Root-Elemente** (dienen als Wurzelement des Stylesheets)
Die einzigen beiden XSLT-Elemente, die alternativ zueinander als Wurzelement des Stylesheets eingesetzt werden können, sind `xsl:stylesheet` und sein selten verwendetes Synonym `xsl:transform`. Sie bilden daher eine eigene, zwei Elemente umfassende Gruppe.
- ▶ **Toplevel-Elemente** (Deklarationen und Strukturelemente)
Unmittelbar dem Wurzelement untergeordnet, definieren sie die globale Struktur des Stylesheets (Templates, Deklarationen von Funktionen, Parametern, Formatierungsregeln und Weiteres). In XSLT 2.0 existieren rund 16 Toplevel-Elemente.
- ▶ **Instruktionen** (Strukturelementen untergeordnete Anweisungen)
Unterhalb der globalen Ebene auftretende Anweisungen (Subdeklarationen, Templateaufrufe, Bedingungen, Schleifen, Konstruktoren etc.). Dieser Gruppe werden in XSLT 2.0 rund 31 Elemente zugeordnet.

Das hier vorgestellte, kurze Stylesheet enthält einen exemplarischen Vertreter aus jeder dieser drei Gruppen: das Root-Element `xsl:stylesheet`, das Toplevel-Element `xsl:template` und die Instruktion `xsl:value-of`.

Diese drei sollen gleich betrachtet werden, die restlichen Elemente werden nach und nach erarbeitet. Eine vollständige Übersicht aller XSLT-Elemente von XSLT 1.0 und XSLT 2.0 befindet sich weiter hinten im Referenzteil zu XSLT.

1.4 Das Wurzelement `xsl:stylesheet`

Nun soll das Stylesheet etwas im Detail betrachtet werden. Zunächst handelt es sich, wie man am XML-Prolog erkennt, offenkundig um ein **XML-Dokument**:

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

Das `version`-Attribut des Prologs bezeichnet den Versionsstand des im Dokument verwendeten XML. Es besitzt den Wert `1.0`, wie derzeit gefordert.⁴

⁴ Für Dokumente gemäß XML 1.1 müsste hier entsprechend der Wert »1.1« stehen. Wenn Sie mehr über XML wissen wollen, möchte ich Ihnen an dieser Stelle »Einstieg in XML, 2.Auflage« von Helmut Vonhoegen, Galileo 2004 empfehlen.

Zusätzlich enthält der Prolog meist noch ein `encoding`-Attribut, um beispielsweise im Zeichensatz des Dokuments Umlaute verwenden zu können – der Einfachheit halber wird hier darauf verzichtet.

1.4.1 Das Wurzelement und seine Namensräume

Nach dem XML-Prolog folgt das Wurzelement des Stylesheet-Dokuments: `xsl:stylesheet`. Auch `xsl:stylesheet` enthält ein `version`-Attribut, das hier jedoch den Versionsstand der im Stylesheet verwendeten XSLT-Version⁵ nennt.

Für **XSLT 2.0** besitzt es daher den Wert 2.0:

```
<xsl:stylesheet version="2.0" ... >
```

Das andere »Attribut« `xmlns:xsl` ist eigentlich keines (man bezeichnet es auch als Pseudoattribut), sondern die Deklaration eines so genannten **Namensraums**. Das Partikel nach dem Doppelpunkt ist frei wählbar und dient als **Präfix** für Elemente, denen der deklarierte Namensraum zugeordnet wird. Für XSLT hat es sich eingebürgert, hier `xsl:` zu verwenden:

```
<xsl:stylesheet version="2.0" xmlns:xsl="...">
```

Zu beachten: Auch das Wurzelement `xsl:stylesheet` verwendet das Präfix `xsl:`, obwohl es in ihm erst vereinbart wird. Dies ist korrekt; ein Namensraum gilt überall *in* dem Element, für das er vereinbart wurde, und auch *für* das Element selbst.

1.4.2 Der Sinn eines Namensraums

Ein Namensraum dient dazu, ein XML-Element **unmissverständlich** einer Gruppe zuzuordnen. Dies beugt zum einen Problemen mit Namensdoubletten bei gleichzeitiger Verarbeitung mehrerer XML-Dokumente unterschiedlicher Herkunft vor. Zum anderen können so Schlüsselkonzepte bestimmter XML-Anwendungen als einem **Bedeutungsraum** zugehörig gekennzeichnet werden, wie XML Schema, Formatting Objects, SVG, MathML – oder eben XSLT.

Verwendet wird dafür (vergleichbar mit der Kennziffer eines Personalausweises) eine unverwechselbare **Zeichenkette**. Weil eben diese Unverwechselbarkeit am einfachsten mit (per Definition einzigartigen) URI-Strings zu erreichen ist, setzt man Webadressen zur Kennzeichnung von Namensräumen ein⁶ – deshalb spricht man auch von **Namensraum-URIs**:

5 Einzig weiterer erlaubter Wert ist derzeit "1.0" für XSLT 1.0. Verschiedene XSLT-Prozessoren erkennen und verarbeiten allerdings auch – eher aus historischen Gründen – den Wert "1.1" für die offiziell zurückgezogene XSLT-Version 1.1.

```
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```

Für XSLT ist dieser URI **reglementiert**; er muss (kategorisch!) der Zeichenkette `http://www.w3.org/1999/XSL/Transform` entsprechen.⁷ Nur so wird ein Element, das »in« diesem Namensraum ist, für die verarbeitende Anwendung, den **XSLT-Prozessor**, eindeutig als XSLT-Instruktion gekennzeichnet.

Das Präfix kann als Platzhalter bzw. Abkürzung für diesen URI-String verstanden werden – man könnte daher, rein theoretisch, anstelle von `xsl:` auch ein beliebig anders lautendes Präfix verwenden.

Unterscheidung zwischen XSLT 1.0 und 2.0:

Der Namensraum-URI `http://www.w3.org/1999/XSL/Transform` ist für XSLT 1.0 und XSLT 2.0 identisch. Beide Sprachversionen werden nur anhand des Wertes des `version`-Attributs von `xsl:stylesheet` unterschieden. Folglich wird mit diesem Attribut also auch indirekt festgelegt, welche XPath-Version innerhalb des Stylesheets genutzt wird.

Der Gesamtbezeichner des Wurzelements `xsl:stylesheet` ist also im Grunde zusammengesetzt und besteht aus dem **lokalen Teil** `stylesheet` und dem **Präfixteil** `xsl:` – zusammen bilden beide Teile einen so genannten qualifizierten Namen für das Element: einen **QName**

Intern arbeitet eine Anwendung immer mit dem so genannten **expandierten QName**, d. h., ausschlaggebend ist nie das Präfix selbst, sondern der Namensraum-URI, für den es steht. Zwei qualifizierte Bezeichner gelten also als gleich, wenn sie den gleichen lokalen Bezeichner und Namensraum-URI besitzen; das im nicht expandierten QName verwendete Präfix ist irrelevant.⁸

Sollte ein und derselbe Namensraum-URI gleichzeitig an zwei verschiedene Präfixe gebunden sein (denkbar z. B. im Fall zweier gleichzeitig verarbeiteter Dokumente), können für den Prozessor auch zwei, auf den ersten Blick unterschiedlich erscheinende QNames als gleich gelten.

⁶ Es ist eine verbreitete Fehlannahme, dass dieser URI auf eine reale Webressource zeigen müsste. Dies ist nicht der Fall; in der Tat kommt es nur auf die unverwechselbare Zeichenkette als solche an (eine Transformation setzt keine Onlineverbindung voraus).

⁷ Der alte, von Microsoft in einer Übergangszeit verwendete URI `http://www.w3.org/TR/WD-xsl` wird von aktuellen Prozessoren nicht erkannt – inzwischen besteht über den zu verwendenden URI jedoch Einigkeit.

⁸ Auch wenn daher XSLT-Instruktionen bei entsprechender Namensraum-Deklaration mit einem anderen Präfix als »xsl:« versehen werden könnten, unterlässt man dies aus Gründen der Deutlichkeit (für menschliche Leser). Einer verarbeitenden Anwendung wäre es jedoch egal.

Beispiel für expandierten QName:

- ▶ Der nicht expandierte QName beispielsweise des Stylesheetelements ist `xsl:stylesheet`. In dieser Form steht der QName im Quellcode.
- ▶ Der expandierte QName, mit dem intern gearbeitet wird, lautet jedoch `{http://www.w3.org/1999/XSL/Transform}:stylesheet`.
- ▶ Dies bedeutet, dass der Prozessor ein in einem QName verwendetes Präfix bei der Verarbeitung im Grunde nicht sieht, sehr wohl aber kennt – dies ist für die Rückwandlung eines Namensraum-URIs in ein Präfix bei der Serialisierung des Ergebnisdokuments von Bedeutung (siehe Anhang C, *Glossar*, *Namespace-FixUp*).

Neben dem hier vorgestellten, obligatorischen XSLT-Namensraum können in `xsl:stylesheet` auch weitere Namensräume deklariert werden (dann mit anderem Präfix und URI). Für das Element existieren noch weitere, optionale Attribute, die zu diesem Zeitpunkt jedoch noch nicht von Interesse sind.

Merkwertes zu `xsl:stylesheet`:

- ▶ Als Wurzelement des Stylesheets wird entweder `xsl:stylesheet` oder (selten, aber funktionsgleich) `xsl:transform` verwendet.
- ▶ Im Wurzelement muss die XSLT-Version festgelegt und müssen Namensräume vereinbart werden; darunter mindestens der für XSLT selbst.
- ▶ Hier vereinbarte Namensräume sind überall im Inneren des Stylesheets verfügbar; Elemente, für die sie gelten sollen, bekommen zur Kennzeichnung das in der Namensraumdeklaration vereinbarte Präfix vorangestellt.
- ▶ Durch den XSLT-Namensraum werden die XSLT-Instruktionen innerhalb des Stylesheets gekennzeichnet (dies gilt auch für das Wurzelement selbst, das ebenfalls das Präfix `xsl:` trägt).

Gehen wir nun im Stylesheet eine Ebene weiter nach innen.

1.5 Die Grundstruktur des Stylesheets: `xsl:template`

Der Inhalt von `xsl:stylesheet` bildet den Rumpf des »eigentlichen« XSLT-Stylesheet. Unmittelbar im Inneren des Wurzelements befinden sich weitere XSLT-Anweisungen, die, weil sie hier sozusagen auf oberstem Niveau im Stylesheet stehen, als **Toplevel-Elemente** bezeichnet werden.

In diesem Beispiel wird nur ein Toplevel-Element verwendet, dafür aber das wohl wichtigste: `xsl:template`

Das Element `xsl:template` bildet zusammen mit seinem jeweiligen Inhalt die entscheidende Grundstruktur eines Stylesheets, ein so genanntes **Template**.

bekannt sein, **dass** es ein Element namens `<gruss>` gibt, und **wo** es sich im Quelldokument befindet. Dies ist weniger trivial, als es im Augenblick erscheinen mag.

1.7.2 XPath-Ausdruck vs. XPath-Pattern

Als **XPath-Ausdruck** bzw. **Expression** wird im Grunde alles bezeichnet, was mit Hilfe von XPath formulierbar ist – die »Expression« bildet also den allgemeinen Oberbegriff; das Pattern ist eine spezielle Untergruppe der Expression.

Eine Expression kann also einerseits in Form eines Pfadausdrucks dazu dienen, einen Teil des Quelldokuments zu bezeichnen. Sie kann aber auch Funktionsaufrufe der Form `funktionsname()` beinhalten, die einen Wert zurückgeben, der dann der aufrufenden Instruktion übergeben wird. In diesem Zusammenhang kann ein XPath-Ausdruck auch Schleifen mit `for` und Bedingungen mit `if` enthalten.

XPath-Ausdrücke werden in den folgenden Kapiteln ausführlicher thematisiert. Vorläufig soll die Betrachtung ihrer einfacheren Untergruppe genügen: der Pfadausdrücke. Vor deren eingehenderen Betrachtung ist zunächst ein weiterer Blick auf XML-Dokumente unter einem ganz anderen Aspekt sinnvoll.

1.8 Das XML-Dokument als Baum

Um den eigentlichen Ablauf einer Transformation und in diesem Zusammenhang die Bedeutung von XPath und Pfadausdrücken besser zu visualisieren, soll der ganze Vorgang nun aus einem neuen Blickwinkel betrachtet werden: dem der Baumstrukturen.

1.8.1 Warum Baumstrukturen?

Ein Dokument als Baumstruktur zu betrachten, mutet zwar abstrakt an, erleichtert jedoch das Verständnis im Fall von XSLT und XPath erheblich:

- ▶ Für die hierarchische Struktur eines XML-Dokuments stellt ein **Baum** die ideale Metapher dar. Ein Baum besitzt genau eine Wurzel, ein XML-Dokument genau ein Wurzelelement. Die diesem untergeordneten Elemente nebst Inhalten bilden die Zweige.
- ▶ XPath wiederum beschreibt die **Bewegung** in einem solchen abstrakten Baum – dient also gewissermaßen als »Motor« von XSLT. Mittels XPath-Ausdrücken können Positionen, Relationen und Bewegungsrichtungen in einem Baum beschrieben sowie einzelne Knoten oder ganze Zweige eines Baums ausgewählt werden.

Zur Darstellung eines XML-Dokuments wird ein Baum im mathematischen Sinne verwendet. Dieser wird stets mit nach unten gerichteter Spitze dargestellt und besitzt einen einzelnen, oben liegenden Startpunkt, der als »Wurzel« bezeichnet wird. Exakt betitelt handelt es sich bei einem Baum, wie er hier vorliegt, um einen **gerichteten Graph**¹⁷. Er setzt sich aus **Knoten** (engl. *nodes*) und **Verbindungen** (Kanten) zusammen. Knoten, die am Ende eines Zweiges liegen, werden als Blätter (engl. *leaf-nodes*) bezeichnet.

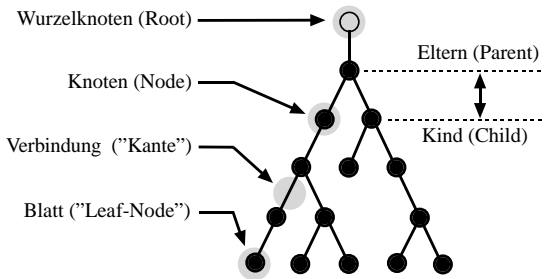


Abbildung 1.5 Dokumentstruktur als gerichteter Graph

1.8.2 Eltern und Kinder im Baumdiagramm

Vergleicht man ein XML-Dokument mit dem es repräsentierenden Baum, so sieht man, dass in den Knoten – soweit diese ein Element darstellen – jeweils **Start- und Endtag** dieses Elements zusammenfallen. Der Inhalt eines Elements wird als von diesem nach unten abgehender Zweig umgesetzt, in dem seinerseits wieder Knoten auftreten können.

Ein Knoten entspricht Start- und Endtag in Einem: Ein Elementknoten als konzeptionelle Einheit lässt die im seriellen Quelltext vorliegende räumliche Trennung von Start- und Endtag entfallen. Beide kann man so als »gleichzeitig« schreibbar verstehen; Inhalte werden an Knoten »angehängt« statt in Container »eingefügt«.

Die durch einen Zweig unmittelbar miteinander verbundenen Knoten stehen zueinander in einem »Eltern-Kind«-Verhältnis: Der enthaltene Knoten gilt als **Kindknoten** (*child*) desjenigen Knotens, von dem er abstammt; dieser wiederum gilt als dessen **Elternknoten**¹⁸ (*parent*). Das Element `` ist ein Kind-

¹⁷ Noch genauer: Es handelt sich um einen »gerichteten azyklischen Graph« (directed acyclic graph, DAG) im Sinne der Graphentheorie. Dass der Baum »azyklisch« ist, also keine Zweige in sich selbst zurücklaufen, hat den Vorteil, dass beim Traversieren des Baums keine Endlosschleifen entstehen können.

¹⁸ Ich versuche den neutralen englischen Begriff »parent« hier ebenso neutral mit »Elternknoten« zu übersetzen – üblich sind auch: Vater- bzw. Mutterknoten, Vorgänger oder übergeordneter Knoten. Im Gegensatz zur menschlichen Gesellschaft hat ein »Kind« in einer Baumstruktur allerdings niemals zwei Elternteile.

knoten von `<a>` und hat seinerseits zwei Kindknoten `<c>` und `<d>`. Auch die beiden Zeichenketten erscheinen im Baum als Kindknoten des jeweiligen Elements, dessen Inhalt sie bilden:

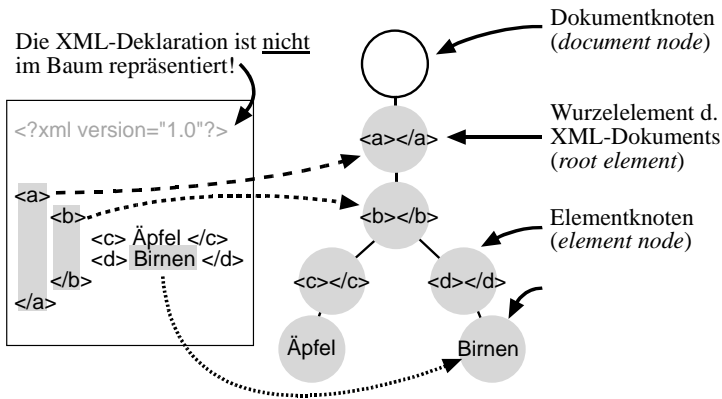


Abbildung 1.6 XML-Dokument und seine Baumentsprechung

1.8.3 Knotengattungen im Dokumentbaum

Eine Besonderheit fällt an dem in Abbildung 1.6 gezeigten Dokumentbaum auf: Seine Wurzel besteht aus einem Knoten (symbolisiert durch einen leeren Kreis), für den es im XML-Dokument **keine Entsprechung** gibt – er repräsentiert auch nicht, wie man vielleicht meinen könnte, die XML-Deklaration!

Man kann, etwas lässig formuliert, sagen, dass dieser Knoten für das Dokument »an sich« steht; das Dokument ist in ihm vollständig enthalten bzw. verkapselt. Dieser Knoten wird daher als **Dokumentknoten** (*document node*) bezeichnet. Bei diesem Knoten setzt stets der Verarbeitungsprozess an, nicht etwa bei dem ihm stets untergeordneten eigentlichen Wurzelement.¹⁹

Direkt unterhalb des Dokumentknotens, also als dessen Kindknoten, findet man das Wurzelement `<a>` des XML-Dokuments. Ein Knoten, der ein Element repräsentiert, wird als **Elementknoten** bezeichnet. Dieser kann als Inhalt wieder andere Elementknoten, jedoch auch andere Arten von Knoten enthalten.

In diesem einfachen Beispiel besteht der Inhalt der Elemente `<c>` und `<d>` nur aus Text, der im Baumdiagramm folgerichtig in Form zweier **Textknoten** erscheint. Kommentare werden durch **Kommentarknoten** vertreten, Processing-Instructions durch **PI-Knoten**.

¹⁹ Die etwas abstrakte Notwendigkeit einer solchen, außerhalb des Dokuments liegenden Instanz wird deutlicher, wenn man zulassen möchte, dass ein »Dokument« nicht strikt ein einzelnes Wurzelement besitzen muss. Überhaupt fasst XSLT 2.0 den Begriff »Dokument« recht weit.

Nicht im Baum dargestellt²⁰ (aber als Information für XPath zugänglich!) werden üblicherweise die **Attribute** und **Namensräume**, für die aber dennoch entsprechende Knotengattungen existieren.

Ebenfalls **nicht** als Knoten im Baum enthalten ist die XML-Deklaration²¹ des Prologs. Im Gegensatz zu Attributwerten und Namensräumen ist sie für XPath nicht zugänglich. Sie ist für uns glücklicherweise auch nicht weiter von Bedeutung; lediglich der XML-Parser verwendet sie, u. a. um zu bestimmen, auf welche Art das Dokument in Bezug auf das Zeichenencoding zu lesen ist.

Wir unterscheiden in XPath **sieben verschiedene Knotengattungen**, anhand der unterschiedlichen in XML-Dokumenten auftretenden Arten von Informationseinheiten. Alle besitzen jeweils individuelle Eigenschaften und werden im XSLT-Prozess dementsprechend unterschiedlich behandelt:

- ▶ Dokumentknoten
- ▶ Elementknoten
- ▶ Textknoten
- ▶ Kommentarknoten
- ▶ Processing-Instruction-Knoten
- ▶ Attributknoten
- ▶ Namensraumknoten

Ein paar der oben aufgelisteten Knotentypen sollen nun etwas näher betrachtet werden.

1.8.4 Dokumentknoten (document node)

Der Dokumentknoten ist ein abstraktes Konzept. Es ist ein **namenloser** Knoten, dessen Inhalt das gesamte Dokument umfasst. Den Dokumentknoten mit einem XPath-Ausdruck auszuwählen, bedeutet daher, den gesamten Dokumentbaum auszuwählen.

Im Fall eines normalen wohlgeformten XML-Dokuments enthält der Dokumentknoten mindestens einen Kindknoten, nämlich das **Wurzelement** des Dokuments. Außer diesem können auf der obersten Ebene aber auch weitere Knoten erscheinen: Besitzt das Dokument einen Prolog, der **Kommentare** oder

²⁰ Dies liegt daran, dass der Baum strikt nach Eltern-Kind-Prinzip aufgebaut ist, Attribut- und Namensraumknoten jedoch aus bestimmten Gründen nicht als Kindknoten (Children) der Elemente gelten, denen sie zugeordnet sind.

²¹ Die XML-Deklaration gilt trotz identischen Aussehens nicht als Processing-Instruction! Sie daher auch nicht als solche zu behandeln, ist folgerichtig.

Prozessanweisungen enthält, so gehen diese als Geschwisterknoten dem Wurzelementknoten voran (dies gilt jedoch nicht für die XML-Deklaration selbst).

Achtung: Es ist wichtig, den Dokumentknoten (*document node*) und das Wurzelement (*root element*) zu unterscheiden.²² Der Dokumentknoten ist **kein Element**. Er verfügt zwar über einem Element vergleichbare Eigenschaften (er besitzt Kindknoten), hat jedoch keinen Elternknoten, keinen individuellen Bezeichner und kann weder Attribute noch Namensräume tragen.

1.8.5 Elementknoten (element node)

Ein Element wird im Baummodell durch einen Elementknoten vertreten. Ein Elementknoten besitzt einen **Bezeichner** in Form eines QName, dieser kann also ein Präfix mit an dieses gebundenen Namensraum-URI besitzen. Ein Elementknoten kann als einziger Knotentyp (neben dem Dokumentknoten) **Kindknoten** besitzen.

Als Kindknoten dürfen wiederum Elementknoten, aber auch Textknoten, Kommentarknoten und PI-Knoten erscheinen. Die Attribute und Namensräume eines Elements werden diesem auch in Form von entsprechenden Knoten zugeordnet, jedoch nicht ihrem Inhalt zugerechnet. Entsprechend zählen sie nicht zu den Kindknoten.

1.8.6 Textknoten (text node)

Ein Textknoten enthält ausschließlich **Zeichendaten**. Er besitzt also keine hierarchisch geordnete innere Struktur. Ein zusammenhängender Bereich aus Zeichendaten in einem Dokument wird immer durch genau einen Textknoten repräsentiert. Daraus folgt, dass in einem Baum nie zwei unmittelbar benachbarte Textknoten²³ auftreten können – diese würden zu einem Knoten zusammengefasst.

Auch **CDATA-Bereiche**, wie sie in XML-Dokumenten auftreten, werden stets durch Textknoten repräsentiert – es existiert für sie keine eigene Knotengattung

²² Verwirrenderweise wurde der »document node« früher mit »root node« bezeichnet oder einfach mit »root« (dies wieder analog zur UNIX-Pfadsyntax), während das Wurzelement des XML-Dokuments wiederum gelegentlich »document element« genannt wird.

²³ Ich möchte darauf hinweisen, dass bei der Verarbeitung auch Stringlitterale »xs:string« entstehen können – bei diesen handelt es sich jedoch nicht um Textknoten, sondern um atomare Werte. Solche Stringlitterale können in unmittelbarer Folge hintereinander auftreten und werden erst bei der Serialisierung des Ergebnisses zusammengefasst.

im Dokumentbaum. Ob es sich bei einem beliebigen Textknoten ursprünglich um einen CDATA-Bereich gehandelt hat oder nicht, lässt sich nicht mehr ermitteln, nachdem ein Dokument den XML-Parser durchlaufen hat.

In vergleichbarer Weise werden auch **Textentities** beim Parsing-Vorgang expandiert. Der Ersetzungstext nimmt die Stelle der ursprünglichen Entityreferenz ein. Auch hier lässt sich im Nachhinein nicht feststellen, ob im Dokument entsprechende Referenzen vorhanden waren.

1.8.7 Der Attributknoten (attribute node)

Die Attribute eines Elements werden durch Attributknoten vertreten, die einen Bezeichner (QName) und einen Wert besitzen. Der Elementknoten gilt zwar als Elternknoten »seines« Attributknotens, dieser wiederum aber nicht als »Kind« des Elements – man sagt vielmehr lediglich, der Attributknoten »gehört« dem Elementknoten (vergleichbar einer Eigenschaft).

Attributknoten sind daher nicht »normaler« Teil des Dokumentbaums – dessen Knoten besitzen per Definition alle eine vollständige Eltern-Kind-Relation. Ein Attributknoten ist daher umständlicher zu erreichen als die anderen Knotentypen (mit Ausnahme der Namensraumknoten).

Achtung: Ist ein Attribut über eine DTD oder ein Schema mit einem Defaultwert belegt, so ist es Sache des XML-Parsers, es einzufügen, wenn es im Quelldokument weggelassen wurde. Da ein XML-Parser aber zum Verarbeiten einer DTD oder eines Schemas nicht verpflichtet ist, kann es sein, dass für ein solches Defaultattribut kein entsprechender Knoten existiert.

1.8.8 Der Baum als Abbild eines Infosets oder PSVI

Vereinfacht wurde gesagt, dass ein XML-Dokument beim Einlesen durch den XML-Parser in eine baumähnliche Form gebracht wird. Dies ist jedoch erst die zweite bzw. dritte Stufe. Vielmehr setzt der Parsingvorgang das Dokument zunächst in eine für eine Folgeanwendung verwertbare Sammlung von Informationen um, das **Infoset**. Innerhalb des Infosets sind die Bestandteile des geparsten Dokuments als **Information-Items** vertreten – ein Element beispielsweise in Form eines Element-Information-Items.

Eigenschaften des Infosets

Teil dieses Infosets sind naheliegenderweise alle im geparsten XML-Dokument enthaltenen Informationen (soweit diese als »relevant«²⁴ angesehen werden).

²⁴ Hierüber gehen die Meinungen allerdings bereits auseinander – sollen beispielsweise Kommentare als relevante Informationen betrachtet werden oder nicht?

Darüber hinaus einbezogen sind (in Form so genannter **Properties**) noch zusätzliche Informationen über die einzelnen Information-Items: Hierzu gehören dessen **Basis-URI**, **Datentyp**, **Name** (für Elemente und Attribute), Informationen über die Position im Dokument (Parent- und Child-Properties) und weitere Eigenschaften. Das Infoset selbst ist jedoch noch kein Baum – dieser und die in ihm versammelten Knoten werden aus dem Infoset erst konstruiert.

Das Post Schema Validation Infoset – PSVI

Ist für den Parser ein **XML Schema**-Dokument zugänglich und ist dieser in der Lage, es auch zu verarbeiten, so wird das Infoset validiert – mit anderen Worten wird zum einen die Gültigkeit des Dokuments geprüft und zum anderen mit den Informationen jeweils ein **Schema-Datentyp** verknüpft (*type annotation*). Hierbei kann es sich um einen Basistyp oder um einen userdefinierten Typ im Rahmen des Schemas handeln. Das Ergebnis dieser Validierung ist ein so genanntes **PSVI** (Post Schema Validation Infoset).

Konstruktion der Knoten des Baums aus dem Infoset

Ausgehend entweder von einem Infoset (ohne erfolgte Validierung des Dokuments) oder von einem PSVI (mit Validierung gemäß eines Schemas) erfolgt nun die »Konstruktion« der Knoten und des aus ihnen zusammengesetzten Baums. Der wesentliche Unterschied im Ergebnis besteht in den mit den Knoten jeweils verknüpften Datentypen.

Ist ein Knoten aus dem »gewöhnlichen« Infoset konstruiert, so ist sein Datentyp meist unspezifisch – so hat beispielweise ein aus dem Infoset stammender Elementknoten grundsätzlich den XML Schema-Typ `xs:anyType`. Für Attributknoten ist die Lage etwas komplexer; hier kann der unspezifische XPath-Typ `xdt:untypedAtomic` zugeteilt werden, jedoch auch ein anderer (wenn z.B. das Attribut in einer DTD als ID, IDREF, NMTOKEN o.Ä. deklariert wurde).

Liegt dem Baum ein PSVI zugrunde, so besitzt jeder Element- und Attributknoten – sofern er validiert wurde – ein `validity`-Property. Dieses kann die Werte »valid«, »invalid« oder »notKnown« annehmen. Elementknoten erhalten, sofern sie »valid« sind, den entsprechenden Schematyp gemäß des Schemas, andernfalls `xs:anyType`. Analog wird für Attributknoten verfahren.

Weitere Informationen zum Datenmodell:

Die offizielle Beschreibung des XPath-Datenmodells sowie der Konstruktion von Knoten aus Infoset und PSVI befindet sich in der W3C-Spezifikation »XQuery 1.0 and XPath 2.0 Data Model« (www.w3.org/TR/xpath-data-model/).

1.9 Eine Transformation Schritt für Schritt

Betrachtet man eine XSLT-Transformation aus der Sicht von Baumstrukturen, so arbeitet man anstelle von Dokumenten mit **Dokumentbäumen**. Auch das Stylesheet wird durch den XML-Parser eingelesen und in einen Baum umgeformt. In Bezug auf das Ergebnisdokument unterscheiden sich XSLT 1.0 und XSLT 2.0 insofern, als dass XSLT 1.0 hier von einem **Ergebnisbaum** spricht, in den unmittelbar geschrieben wird. Dieses mit »Top-Down« bezeichnete Verfahren ist daher relativ anschaulich.

XSLT 2.0 legt dagegen ein abstrakteres »Bottom-Up«-Verfahren zugrunde und betrachtet das Ergebnis zunächst nicht als Baum, sondern als Datenstrom in Form einer **Ergebnissequenz**. Diese wird allerdings im Zuge der zu Beginn der Serialisierung erfolgenden Normalisierung ebenfalls in einen Baum umgewandelt, sodass es (halbwegs) zulässig erscheint, hier das gleiche Bild anzuwenden.

Im Zentrum der XSLT-Transformation steht der Baum des Quelldokuments; genauer gesagt **steuert** die Struktur des Quelldokumentbaums sogar indirekt die Verarbeitung. Dieser Baum wird von oben her durchwandert, was auch der Leserichtung im Dokument entspricht.

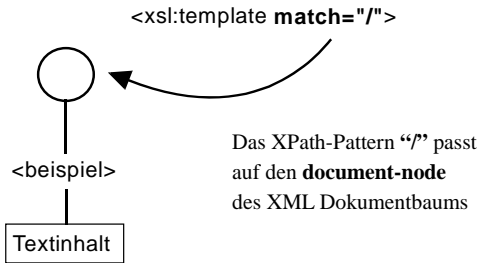
1.9.1 Aktuell: Der Current Node im Quelldokument

Den Vorgang der Verarbeitung kann man als durch den Quelldokumentbaum wandernden **Cursor** visualisieren, der stets auf genau einem Knoten platziert wird: Dieser Knoten wird als »aktueller Knoten« bezeichnet, als **Current Node**.

Zu Anfang fällt der Current Node automatisch auf den **Dokumentknoten** des Quelldokuments. Das XPath-Muster, das den Dokumentknoten repräsentiert, lautet »/.« Die für ihn aktivierte Templaterregel ist bekanntlich jene mit `match="/"`. Die Verarbeitung **beginnt** also stets mit dem Dokumentknoten des Quelldokuments und schreitet von da ab sukzessive in Richtung der Kindknoten voran.

Der XSLT-Prozessor beginnt also für den Current Node stets mit der Verarbeitung der Inhalte der passenden Templaterregel, die dann als **aktuelle Templaterregel** bezeichnet wird. So wie es immer nur genau einen aktuellen Knoten gibt, so existiert auch nur eine aktuelle Templaterregel.

Mit dem aktuellen Knoten als Kontext werden durch Instruktionen in der aktuellen Templaterregel mittels XPath-Ausdrücken Knotenmengen (Sequenzen) im Quelldokumentbaum ausgewählt. Diese Knotenmenge bezeichnet man als die **aktuelle Sequenz**.



gedanklich mit einer Quellcodeansicht des Ergebnisses, in der die Tagcontainer quasi gleichzeitig von vorne und hinten nach innen geschrieben werden müssten, so fällt die Übersichtlichkeit der Baummetapher ins Auge.

Ist das Quelldokument komplett durchlaufen, so ist die Transformation beendet und der Ergebnisbaum vollständig. Er kann jetzt in ein wirkliches Dokument umgewandelt werden: Diesen Vorgang bezeichnet man als »Plätten« des Baums oder (die geläufigere Benennung) als dessen **Serialisierung**.

Die Realität in XSLT 2.0

Dieses so angenehm anschauliche Bild des wachsenden Ergebnisbaums mag für XSLT 1.0 gelten, für XSLT 2.0 ist es leider unkorrekt. Dort wird nicht abschnittsweise ein Baum konstruiert, sondern die Inhalte werden nacheinander in eine Ergebnissequenz ausgegeben. Diese wird nach Abschluss der Transformation serialisiert.

Die **Serialisierung** stellt einen eigenständigen, von der Transformation unabhängigen Verarbeitungsschritt dar. Bei der dabei erfolgenden Normalisierung der Sequenz entsteht (vorübergehend) entweder ein vollständiger Baum oder die Serialisierung schlägt vollständig fehl. Keinesfalls kann (z.B. durch einen vorzeitigen Abbruch) ein unfertiger Baum bzw. ein unfertiges Dokument Ergebnis einer Transformation sein.

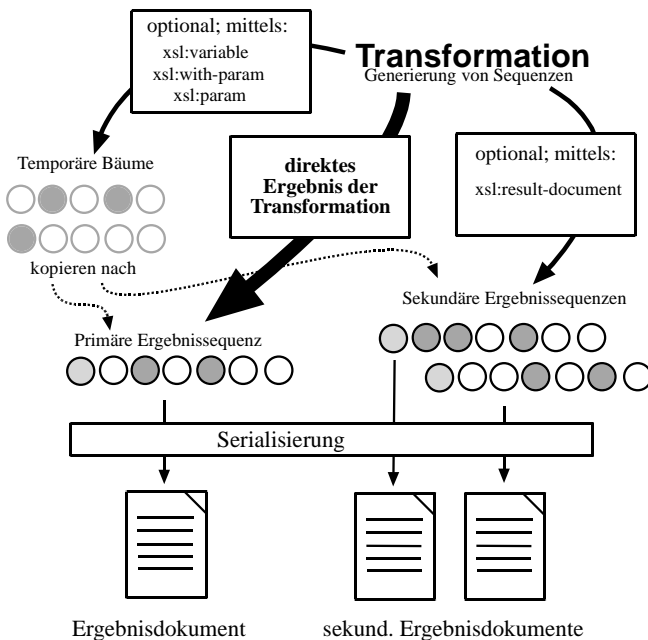


Abbildung 1.9 XSLT 2.0 – eine Vielzahl möglicher Sequenzen

Da XSLT 2.0 im Gegensatz zu XSLT 1.0 mehrere Ergebnisdokumente gleichzeitig produzieren kann, erzeugt der Prozessor gegebenenfalls auch mehrere Ergebnissequenzen. Darunter ist jedoch nur eine **primäre Ergebnissequenz**, die zum oben beschriebenen Ergebnisdokument korrespondiert. Die anderen Sequenzen werden zu sekundären Ergebnisdokumenten serialisiert – sie werden mittels der Instruktion `xsl:result-document` ins Leben gerufen.

Zusätzlich existieren quasi »frei vagabundierende« Sequenzen in Form so genannter **temporärer Bäume**²⁶ (diese sind in Variablen abgelegt und verfügen über eigene Dokumentknoten), die aber nicht serialisiert werden, sondern höchstens zu einem frei bestimmbar Zeitpunkt in eine der Ergebnissequenzen kopiert werden.

Dieses komplexe Bild wird früh genug Thema werden. Zunächst soll jedoch noch bei der einfachen Konstellation geblieben werden, anhand derer Sie einen ersten Eindruck von XPath-Pfadausdrücken gewinnen.

1.10 Start mit XPath: Pfadausdrücke

XPath-Pfadausdrücke dienen, vom Stylesheet ausgehend, der Lokalisation von Teilen des Quelldokuments. Die derart durch einen Pfadausdruck beschriebenen Knoten (Elemente, Attribute, Text etc.) werden ausgewählt und können anschließend verarbeitet werden, beispielsweise – wie eben gesehen – indem ihr Textinhalt extrahiert wird.

Im Rahmen des letzten Beispielstylesheets (`beispiel2.xsl`) wurde ein Pfadausdruck vorgestellt, der einfach nur aus dem **Bezeichner** des zu adressierenden Elements bestand. Ganz so simpel ist es allerdings nicht. Um dies darzulegen, muss das vorige Beispieldokument etwas umgeschrieben werden.

1.10.1 Gewusst wo: Vom Auffinden der Inhalte

Zur Demonstration werden zwei um eine Tag-Ebene erweiterte Versionen des simplen XML-Dokuments (`beispiel2.xml`) erstellt – indem einfach ein Tag `<hallo>` hinzugefügt wird, allerdings an unterschiedlichen Positionen.

In der ersten Version liegt dieser im Inneren des Elements `<gruss>`:

```
<?xml version="1.0"?>
<gruss>
  <hallo>
    Hallo Welt! Dies ist XSLT.
```

²⁶ Die Bezeichnung »Baum« ist insofern irreführend, als dass es sich auch hier »nur« um Sequenzen handelt.

2 XSLT und XPath – Runde zwei

2.1	XPath – Filtern von Sequenzen	109
2.2	Bedingungen mit XSLT und XPath	119
2.3	Nummerieren, Zählen, Summieren	125
2.4	Sortierung – xsl:sort	138
2.5	Schleifen.....	144
2.6	Gruppierung mit xsl:for-each-group	150
2.7	Attributwerte zur Laufzeit: Attributwert-Templates ...	153
2.8	Erzeugung von Knoten	159
2.9	Kopieren von Knoten: xsl:copy und xsl:copy-of	164
2.10	Zusammenfassung und Aufgaben.....	167

1 XSLT und XPath – der Start

2 XSLT und XPath – Runde zwei

3 XSLT und XPath – Runde drei

4 Referenz XPath – die Sprache

5 Referenz XPath-Funktionen

6 Referenz XSLT-Elemente

7 Referenz Saxon 7

8 XSLT-Entwicklungsumgebungen

A Lösungen

B W3C-Online-Ressourcen

C Glossar

2 XSLT und XPath – Runde zwei

Das zweite Kapitel des Buchs schildert die Filterung von Sequenzen – entweder mittels Predicates bei ihrer Erstellung oder bei ihrer Verarbeitung durch Bedingungen. Darüber hinaus werden Konzepte zur Formulierung von Schleifen, zur Gruppierung, Sortierung, Nummerierung und zum Kopieren von Inhalten sowie zur Erzeugung von Knoten vorgestellt.

2.1 XPath – Filtern von Sequenzen

Der Verarbeitungsablauf einer XSLT-Transformation ist direkt abhängig von der Zusammensetzung der zur Verarbeitung anstehenden Sequenzen. Eine Sequenz wird, wie im vorigen Kapitel gezeigt, durch einen Pfadausdruck zusammengestellt. Dies findet in der Regel im Rahmen der Verarbeitung von Instruktionen wie `xsl:apply-templates` statt, aber auch in Instruktionen, die Sequenzen unmittelbar ausgeben, wie `xsl:value-of`.

Ein Pfadausdruck setzt sich dabei aus einer Folge von Location-Steps zusammen. Jeder Schritt erfolgt entlang einer Achsenrichtung und erzeugt eine Ergebnissequenz, deren Knoten jeweils Ausgangspunkt des folgenden Schritts werden. Hat daher der erste Schritt eine Ergebnismenge von drei Knoten, so teilt sich der Pfadausdruck für den folgenden Schritt in drei Zweige.

Der Pfadausdruck ist dabei umfassend, seine endgültige Ergebnissequenz beinhaltet daher alle Knoten der Ergebnismenge des letzten Schritts (bzw. der »Schrittinstanzen«, deren Anzahl wieder von der Ergebnismenge des Schritts davor abhängt). Es ist, da diese Menge sehr groß werden kann und auch Knoten enthalten mag, die im Endeffekt nicht verarbeitet werden, sehr wichtig, frühzeitig **Filterbedingungen** setzen zu können – und zwar bereits für jeden erfolgenden Zwischenschritt einzeln.

Das Beispieldokument – ein Buchkatalog

Das Dokument, anhand dessen in Folge einige Grundkonzepte gezeigt werden, setzt sich zusammen aus einer Reihe von `<buch>`-Elementen, die Attribute `ersch-jahr` und `kategorie` besitzen. Das Attribut `kat` ist in der DTD mit einem Defaultwert `"edv"` versehen:

```
<!ATTLIST buch
  ersch-jahr CDATA #IMPLIED
  kat CDATA "edv">
```

Listing 2.1 buchhandel.dtd

Bei den meisten Buchelementen ist dieses Attribut in der Datei nicht explizit genannt, wird aber durch den XML-Parser eingefügt. Die restlichen Informationen stehen in Child-Elementen, das `<preis>`-Element besitzt ein Attribut `wahrung`, das die Währung anzeigt – Default (ebenfalls per DTD) ist "Euro". Für einige Bücher ist dieser mit dem Wert "Dollar" überschrieben.

```
<buchhandel>
... <!-- mehr Bücher -->

<buch ersch-jahr="1996">
  <buchtitel>Where Wizards Stay Up Late</buchtitel>
  <buchautor>Hafner, Katie</buchautor>
  <buchautor>Lyon, Matthew</buchautor>
  <buchverlag>Touchstone</buchverlag>
  <preis wahrung="Dollar">14.00</preis>
</buch>

<buch ersch-jahr="1950" kat="tiere">
  <buchtitel>Unter Korallen und Haien</buchtitel>
  <buchautor>Hass, Hans </buchautor>
  <buchverlag>Ozean Verlag</buchverlag>
  <preis wahrung="Euro">37.95</preis>
</buch>

... <!-- mehr Bücher -->

</buchhandel>
```

Listing 2.2 buchhandel.xml

Es ist nun auf dem bereits bekannten Wege verhältnismäßig einfach, die Verarbeitung pauschal aller `<buch>`-Elemente des Dokuments zu veranlassen. Angenommen jedoch, dies soll genau nicht geschehen, so sollten die unerwünschten Buchdatensätze bereits bei der Zusammenstellung der Sequenz ausgefiltert werden. Dies geschieht innerhalb des Pfadausdrucks.

2.1.1 Predicates – Filterbedingungen für Location-Steps

In XPath wird eine hierfür erforderliche Filterbedingung mittels eines so genannten **Predicates** definiert (gewöhnlich etwas unschön mit »Prädikat«

übersetzt¹). Ein solches Predicate wird, in eckige Klammern gestellt, einfach an Achsenbezeichner und Knotentest eines Schritts angehängt:

```
achsenbezeichner::knotentest[predicate]
```

Der Inhalt des Predicates kann ein beliebiger XPath-Ausdruck sein, der Bezeichner, Operatoren und Funktionsaufrufe enthält. Ein Schritt kann dabei mehrere Predicates in Folge besitzen:

```
achsenbezeichner::knotentest[predicate1] [predicate2]
```

Dies kann man als eine Art Folge von Sieben verstehen, durch die die Ergebnismenge des Location-Steps gefiltert wird. Es ist wichtig, sich zu vergegenwärtigen, dass die resultierende Menge **nicht unabhängig von der Reihenfolge** der Predicates ist, zumindest nicht in jedem Falle: Stellt man sich beispielsweise eine Bedingung vor, die pauschal jedes zweite Item einer Sequenz entfernt, und eine weitere, die den verbleibenden Rest mit einem Schwellenwertvergleich filtert, so wird dieser Umstand deutlicher.

Setzt sich ein Pfadausdruck aus einer Reihe von Schritten zusammen, so kann jeder Schritt eine beliebige Menge von Predicates erhalten:

```
step1[predicate]/step2[predicate]/step3[predicate]
```

Jeder Predicate-Ausdruck wird in Zusammenhang mit einem geprüften **Kandidatenknoten** zu einem **Booleschen Wert** ausgewertet. Ist dieser `true`, so wird der geprüfte Knoten der Ergebnismenge des Schritts zugeschlagen (bzw. beibehalten), andernfalls verworfen. Die nach dem letzten Schritt verbleibende Sequenz ist die Ergebnissequenz des Pfadausdrucks.

Auf diesem Wege ist es möglich, die Zusammensetzung einer Sequenz bis ins Detail zu bestimmen. Dies soll jetzt an ein paar einfachen Beispielen gezeigt werden.

2.1.2 Die Filterung einer Sequenz

Prüfung auf Existenz eines Attributs oder Kindelements

Ein simples Predicate prüft auf das Vorhandensein eines Kindelements oder eines Attributs, ohne dessen konkreten Wert zu berücksichtigen. So wählt

```
select="beispiel[test]"
```

1 Adäquater wäre »Filterbedingung« oder »Prädikatfilter«. Ich favorisiere in diesem Fall den kürzeren englischen Ausdruck – im Sinne eines Fachausdrucks, nicht aus Hang zur Anglisierung dieses Textes.

alle Elementknoten `<beispiel>`, die ein Kindelement `<test>` besitzen, unabhängig von dessen Inhalt. Analog ist dies für Attributknoten möglich:

```
<xsl:apply-templates select="buch[@kat]"/>
```

wählt alle Bücher in unserem Beispiel, die über ein `kat`-Attribut verfügen. In diesem Falle nützt dies als Filterbedingung konkret nichts, da für das Attribut ein Defaultwert vereinbart ist, und es, selbst wenn es im Quelldokument faktisch fehlt, durch den XML-Parser ergänzt wird. Da dies jedoch geschieht, *bevor* der XSLT-Prozessor das Predicate prüft, ist für diesen ein `real` im Quelltext vorhandenes und ein virtuell eingefügtes Defaultattribut nicht unterscheidbar.

Prüfung auf Gleichheit und Ungleichheit mit konkretem Wert

Soll beispielsweise eine Liste aller Bücher erstellt werden, deren Preis in Dollar angegeben wurde, so kann dies einfach bei der Zusammenstellung der Sequenz geschehen (Current Node sei das Wurzelement `<buchhandel>`):

```
<xsl:apply-templates select="buch[preis/@waehrung='Dollar']"/>
```

Das Predicate enthält einen XPath-Ausdruck, dessen Auswertung für jeden Knoten der Resultatsequenz des Steps einen Booleschen Wert ergibt: Ein Buch besitzt entweder Kindelement `<preis>`, das über ein `waehrung`-Attribut mit Wert "Dollar" verfügt – dann verbleibt es in der Sequenz –, oder das Attribut hat einen anderen Wert (bzw. existiert nicht), woraufhin der entsprechende Knoten verworfen wird. Geprüft werden aber *alle* Bücher.

Achtung: Da der gesamte XPath-Ausdruck in doppelten Anführungszeichen steht, muss als Stringbegrenzer des Attributwerts innerhalb des Predicates das einfache Anführungszeichen verwendet werden:

```
buch[buchverlag='Ozean Verlag']
```

Es lassen sich ohne weiteres Filter kombinieren. Wir können aus dem EDV-Sektor nur die Bücher übernehmen, die nicht von Katie Hafner² stammen. Hierfür fügen wir einfach ein weiteres Predicate hinzu:

```
<xsl:apply-templates
  select="buch[@kat='edv'][buchautor!='Katie Hafner']"/>
```

2 Katie Hafner, Matthew Lyon: »Where Wizards Stay Up Late«, dt. »Arpa Kadabra«; dpunkt-Verlag, 1997. Lesenswerte und historisch korrekte Darstellung der technischen Geschichte des Internets.

Triviale Prüfung auf Position in der Sequenz

Die Items einer Sequenz sind ihrer Reihenfolge entsprechend durchnummeriert, also über einen Index ansprechbar bzw. beschreibbar. Der Index beginnt immer bei 1 für das erste Item. Eine **Indexziffer** kann direkt als Filterkriterium verwendet werden – hierfür wird die entsprechende **Ganzzahl** als Predicate-Ausdruck eingesetzt:

```
buch[1]
```

lässt aus der Sequenz aller Buchelemente, die der Knotentest zusammenstellt, nur das erste Item passieren, in diesem Fall also das erste Buch in Dokumentreihenfolge. Ein solcher Ausdruck ist auch im Inneren eines anderen Predicate-Ausdrucks möglich: Sollten Sie beispielsweise den Namensvergleich auf das erste `<buchautor>`-Element beschränken müssen (ein möglicher Grund hierfür wird weiter unten gezeigt), so können Sie

```
buch[buchautor[1]='Lyon, Matthew']
```

schreiben, was nur die Bücher durchlässt, für die Matthew Lyon im jeweils ersten `<buchautor>`-Element genannt wird (im betrachteten Fall ist er Co-Autor eines Buchs, also an zweiter Stelle erwähnt – die Ergebnissequenz ist also leer).

Bei dieser Form des Predicates darf nur genau eine Ganzzahl `xs:integer` übergeben werden. Der Versuch, mit Hilfe zweier gleichzeitig übergebener Indexzahlen – etwa in der Form `beispiel[1 3]` – mehrere konkrete Items zu wählen, scheitert (Syntaxfehler). Ebenso wenig funktioniert die Übergabe einer Sequenz aus Ganzzahlen, wie sie mit einem Ausdruck `2 to 5` erzeugt würde: Dieser Ausdruck ergibt in der Tat die Sequenz (2, 3, 4, 5), leider jedoch wählt `beispiel[2 to 5]` nicht das »zweite bis fünfte« Item aus,³ sondern lässt alle Items passieren, ohne überhaupt zu filtern!

Der Grund besteht darin, dass eine übergebene Sequenz, sobald sie nicht leer ist, stets den Booleschen Wert `true` ergibt. Einen Syntaxfehler stellt dies nicht dar – die Übergabe einer Sequenz ist prinzipiell erlaubt, lediglich wenig sinnvoll ...

Wertvergleich – kleiner und größer

In einem Predicate kann auch ein Vergleich auf »größer als« oder »kleiner als« stattfinden. Die Operatorsymbole `>` und `<` können allerdings im Stylesheet nicht direkt eingesetzt werden, da der XML-Parser diese als Elementbegrenzer missverstehen würde. Man verwendet deshalb die Entities `>` und `<`.

³ Eine Lösung hierfür wird weiter unten gezeigt.

Der XML-Parser ersetzt die Entities, bevor der XSLT-Prozessor sich an die Arbeit macht – dieser findet daher die richtigen Symbole vor.

Sollen nur vor 2000 erschienene Bücher ausgegeben werden, so schreibt man:

```
select="buch[@ersch-jahr < '2000']"
```

2.1.3 Logische Verknüpfung von Bedingungen mit AND und OR

Innerhalb eines Predicates können problemlos **mehrere, logisch verknüpfte Bedingungen** auftreten, die dann gleichzeitig AND oder alternativ OR erfüllt sein müssen. Die beiden XPath-Ausdrücke, die jeweils für sich einen Booleschen Wert ergeben müssen, werden in der eckigen Prädikatsklammer durch die Token `and` bzw. `or` verknüpft. Da die Präzedenz der **logischen Operatoren** gegenüber anderen Operatoren (Rechenoperatoren, Vergleichsoperatoren etc.) niedriger ist, müssen die verknüpften Ausdrücke nicht geklammert werden.

Vereinigungsmenge – einschließendes Oder

Will ich die Autoren 'Hafner, Katie' *und* 'Hass, Hans' sehen, so muss ein Filterprädikat mit OR-Relation gebildet werden, das beide Autoren passieren lässt:

```
buch[buchautor='Hafner, Katie' or buchautor='Hass, Hans']
```

Dies würde Bücher auswählen, die beide – einzeln oder auch gemeinsam – als Autoren besitzen. Es müssen nicht ähnliche Bedingungen sein, denkbar wäre z.B. auch

```
buch[buchautor='Panter, Peter' or @kat='reisen']
```

Dies entspricht der **Vereinigungsmenge** aller Bücher des Autors Peter Panter mit allen Büchern der Kategorie »Reisen«. Hierbei sind Buchelemente, die beide Bedingungen erfüllen, jedoch nicht doppelt in der Ergebnissequenz enthalten (obwohl Knoten durchaus doppelt in Sequenzen auftreten dürften).

Schnittmenge – ausschließendes Und

Sollen beide Bedingungen gleichzeitig gelten, so verwendet man die AND-Relation. Dies entspricht im Wesentlichen vom Ergebnis zwei unabhängigen Predicates: In der Ergebnissequenz befinden sich nur diejenigen Nodes, die beide Bedingungen erfüllen.

```
[buchautor='Hafner, Katie' and buchautor='Lyon, Matthew']
```

Dies kann nur der Fall sein, wenn ein Buch mindestens zwei `<buchautor>`-Elemente besitzt und unter diesen die beiden getesteten Werte auftreten.

Operatoren in Predicates

Hier noch einmal eine kurze Auflistung der möglichen Operatoren, die bei der Formulierung von Predicates verwendet werden können. Die Reihenfolge von oben nach unten entspricht der oben angesprochenen **Operatorpräzedenz**.

Test/Aufgabe	Operator
Unäres Minus (Vorzeichen)	-
Multiplikative Operatoren	*, div, idiv, mod
Additive Operatoren	+, -
Vergleichsoperatoren (<, > zu ersetzen durch > und <)	=, !=, <, >, <=, >=, eq, ne, lt, gt, le, ge, is, isnot
Bedingung, additiv	and
Bedingung, alternativ	or

Tabelle 2.1 Operatoren in Predicates und ihre Präzedenz

2.1.4 XPath-Funktionen in Predicate-Ausdrücken

Die zur Bildung von Predicates eingesetzten XPath-Ausdrücke erlauben auch den Einsatz von **Funktionen** zur Formulierung der Bedingung. In erster Linie von Interesse ist hier die Teilmenge von Funktionen, die Auskunft über den aktuellen Kontext geben kann – hier als **Kontextfunktionen** bezeichnet. Aber auch alle anderen Funktionen aus XPath und XSLT dürfen in diesem Zusammenhang eingesetzt werden, wobei einige der **Stringfunktionen** und **Sequenzfunktionen** von vorrangiger Bedeutung sind.

XPath-Funktionen – Streiflicht auf die Syntax

Hier soll zunächst ein kurzer Überblick über die Syntax eines Funktionsaufrufs gegeben werden. Wie sieht eine solche Funktion aus?

Ganz allgemein verwendet man für XPath-Funktionen innerhalb von XPath-Ausdrücken folgende Syntax:

```
funktionsname()
```

Ähnlich wie in den meisten anderen Programmiersprachen auch, steht hinter dem Funktionsnamen eine Argumentklammer – diese kann dazu dienen, Eingaben entgegenzunehmen, die verarbeitet werden:

```
funktionsname(argument1, argument2)
```

Jede Funktion hat einen Rückgabewert, den der Prozessor nach der Verarbeitung durch den Rückgabewert (das Ergebnis) ersetzt. Der Funktionsaufruf ist also eine Art von Platzhalter für seinen Rückgabewert.

```
<xsl:value-of select="funktionsaufruf()"/>
```

entspricht daher

```
<xsl:value-of select="rückgabewert"/>
```

wobei dieser Rückgabewert ein atomarer Wert, ein Knoten oder eine Sequenz sein kann – gelegentlich gibt eine Funktion auch eine leere Sequenz bzw. einen leeren String zurück, was beides im jeweiligen Kontext einen neutralen Wert bedeutet (der also keinesfalls Schaden anrichtet).

Der zum Aufruf verwendete Funktionsname kann **qualifiziert** sein, d.h. mit einem Namensraumpräfix verwendet werden. Für XPath-Funktionen verwendet man das Präfix `fn:`, und zwar mit folgender Namensraumdeklaration:⁴

```
xmlns:fn="http://www.w3.org/2003/11/xquery-functions"
```

In der Regel werden XSLT-Prozessoren die XPath-Funktionen auch an ihrem unqualifiziertem Bezeichner erkennen. Sofern Sie die Funktionsnamen mit Präfix verwenden möchten, ist es jedoch obligatorisch, den entsprechenden Namensraum im Stylesheet zu deklarieren! In Folge wird zur Erleichterung der Lesbarkeit auf das Präfix verzichtet.

Zu beachten, wenn auch im aktuellen Zusammenhang noch nicht von Bedeutung, ist die **Typbindung** von Funktionsargumenten: Funktionen erwarten in XPath 2.0 Eingangswerte eines bestimmten Typs und melden einen Typfehler, wenn dies nicht gewährleistet ist. Eine automatische Typkonvertierung hinter den Kulissen, wie man sie von XPath 1.0 her kennt, gibt es nicht mehr. Im Gegenzug besitzt auch der Rückgabewert verlässlich einen bestimmten Datentyp.

Kontextfunktionen: position() und last(),

Hilfreich bei der Verarbeitung einer Sequenz sind Kenntnisse über die Zahl der in ihr existierenden Items und die Position des aktuell verarbeiteten Items in der Sequenz – mit anderen Worten über den Kontext. Im Fall einer Sequenz aus Elementen `<buch>` mag für die Verarbeitung von Interesse sein, wie viele Bücher ausgegeben werden sollen, und ob das aktuell ausgegebene Buch möglicherweise das letzte in der Sequenz ist.

XPath stellt hierfür zwei Funktionen zur Verfügung, die die genannten Informationen liefern können: `position()` und `last()`.

⁴ Dieser Namensraum-URI entspricht dem der Working Draft vom November 2003 und ist nicht als endgültig zu betrachten.

`position()`

gibt eine Ganzzahl zurück, die der Position des **aktuellen Items** in der aktuellen Sequenz entspricht.

`last()`

gibt eine Ganzzahl zurück, die der Position des **letzten Items** der Sequenz entspricht. Dies ist gleichbedeutend mit der Zahl der Items.

Beide Funktionen werden häufig innerhalb von Predicates benötigt (auch für andere Zwecke, etwa Nummerierungen, wie im Folgenden gezeigt werden wird).

Um aus der Initialsequenz alle bis auf die ersten fünf Bücher mittels eines Predicates auszufiltern, können Sie folgenden Ausdruck verwenden:

```
<xsl:apply-templates select="buch[position() &lt; 6]"/>
```

Hierbei ist die Sequenz, auf die `position()` angewendet wird, die zunächst vom Location-Step gebildete Sequenz aus allen Büchern. Von denen werden diejenigen gewählt, die das Predicate erfüllen – dies sind die ersten fünf `<buch>`-Elemente.

Um nur die letzten fünf Bücher in der Sequenz zu behalten, können Sie Folgendes schreiben:

```
<xsl:apply-templates select="buch[position() &gt; last()-6]"/>
```

Dies funktioniert, weil der Subtraktionsoperator höhere Präzedenz als der Vergleichsoperator besitzt. Eine Klammersetzung ist nicht nötig.

Um auf die oben angerissene Problemstellung mehrerer Indexzahlen zurückzukommen – mit Hilfe von `position()` sind diese einfach lösbar.

```
buch[position()=1 or position()=3]
```

wählt das erste und dritte Buch der Sequenz,

```
buch[position() &gt;1 and position() &lt; 6]
```

wählt das zweite bis fünfte Buchelement der Sequenz.

Boolesche Funktionen: `not()`

Die Funktion `not()` negiert ihr Eingabeargument und kann zur Formulierung von Predicates herangezogen werden. Ist das Funktionsargument `true`, so gibt die Funktion `false` zurück und umgekehrt:

```
buch[not(buchautor='Panter, Peter' or @kat='reisen')]
```

Dieser Pfadausdruck enthält nur jene Bücher, deren Autor weder Peter Panter heißt noch die in die Kategorie 'reisen' fallen.

Die Verwendung der Funktion ist selten wirklich erforderlich; sie lässt sich oft umgehen, indem ihr Argument umgekehrt wird. Hier wäre dies

```
buch[buchautor!='Panter, Peter' and @kat!='reisen']
```

Stringfunktionen: contains(), starts-with(), ends-with()

Wenn es um Bedingungen für den Stringwert von Knoten geht, kann auf entsprechende Funktionen von XPath zurückgegriffen werden – für Predicates eignen sich vordringlich die Funktionen `contains()` und `starts-with()`, gegebenenfalls (eher selten) auch `ends-with()`, die alle zwei Stringargumente entgegennehmen und einen Booleschen Wert zurückgeben. So kann beispielsweise mittels

```
buch[contains(buchautor[1], 'Hans')]
```

eine Sequenz aus Buchelementen zusammengestellt werden, die ein `<buchautor>`-Kindelement besitzen, dessen Stringwert die Zeichenkette 'Hans' enthält. Dies wäre für 'Hass, Hans' der Fall oder für 'Hansen, Carl'.

Interessanter ist vielleicht in diesem speziellen Zusammenhang die Funktion `starts-with()`, falls nach Anfangsbuchstaben gefiltert werden soll.

```
buch[starts-with(buchautor[1], 'A')]
```

liefert Bücher aller Autoren, deren Nachname mit A beginnt (was bei der hier verwendeten Inhaltsstruktur des Buchautor-Elements funktioniert).

Hinweis: Das zusätzliche innere Predicate `buchautor[1]` ist notwendig, weil die Stringfunktionen `contains()` und `starts-with()` genau einen String als erstes Argument erwarten. Ein zweites oder drittes Autorelement muss vorsichtshalber ausgefiltert werden, damit die Verarbeitung stattfinden kann.

RegEx-Funktionen: matches()

Komplexere Bedingungen erfordern die Verwendung **regulärer Ausdrücke**, die von XPath 2.0 im Rahmen mehrerer Funktionen unterstützt werden. Für Predicates bietet sich `matches()` an, das `true` zurückgibt, wenn der reguläre Ausdruck auf den untersuchten String passt:

```
buch[matches(buchautor[1], '^[A-H].*ss[a-z]+')]
```

Listing 2.3 buchhandel-regex.xml

Selektiert die Bücher, deren erstes Autorelement einen Nachnamen (mit denen der untersuchte String beginnt) mit einem Anfangsbuchstaben von A bis H besitzt, dem irgendwo im Namen ein Doppel-S folgt, an das sich mindestens ein weiterer Buchstabe anschließt. Dies gilt für 'Assmann, Peter', 'Esser, Klaus', 'Hesse, Hermann', aber nicht für 'Hass, Hans' (hier folgt dem zweiten S ein Komma, kein Buchstabe).

2.2 Bedingungen mit XSLT und XPath

Nachdem jetzt die gezielte Filterung der Zusammensetzung einer Sequenz möglich ist, besteht immer noch das Bedürfnis, die Verarbeitung ihrer Items innerhalb einer Templateregeln nach bestimmten Kriterien variieren zu können. Von Programmiersprachen her bekannt sind die Entscheidungsmöglichkeiten IF-ELSE (wenn-dann-anderenfalls). In XSLT-Stylesheets ergeben vergleichbare Strukturen die Möglichkeit, die Verarbeitung etwa durch die Inhalte des XML-Dokuments zu steuern (bestimmte Werte, Vorhandensein von Kindelementen, Attributen, sonstige Bedingungen).

Es wird immer ein Test mit einem XPath-Ausdruck durchgeführt, der, wie bei den soeben behandelten Predicates, einen Booleschen Wert erzeugt. Dieser Test bestimmt, wie der Entscheidungsblock verarbeitet wird. XSLT kennt zwei Grundformen: das simple `xsl:if` und das komplexere `xsl:choose`.

2.2.1 Entscheidungen mit XSLT: `xsl:if`

Die Instruktion `xsl:if` stellt im Grunde eine Art praktischen Schalter dar, der Teile eines Templateblocks an oder abschalten kann. Hierbei ist nur der IF-Zweig einer typischen Bedingung vorhanden. Es gibt für `xsl:if` nur die Wenn-Entscheidung, kein Andernfalls, also kein korrespondierendes `xsl:else`.

Ergibt die Auswertung des Werts des `test`-Attributs »true«, so wird der Inhalt des `xsl:if`-Containers ausgeführt, anderenfalls wird er ersatzlos übersprungen.

```
<xsl:template match="buch">
  <p><xsl:apply-templates/>
  <xsl:if test="@isbn">
    <br/>(ISBN: <xsl:value-of select="@isbn"/>)
  </xsl:if>
</p>
</xsl:template>
```

Listing 2.4 buchhandel-ifi.xml

Index

A

Abbreviated Syntax 1009
 Abgekürzte Schreibweise 60, 1009
 Absoluter Location Path 1009
 Abwärtskompatibilität
 XPath 2.0 zu XPath 1.0 313
 accessible collections 247
 accessible documents 247
 Achse 58, 272, 273, 1009
 Abgekürzte Schreibweise 60, 1009
 Achsenbezeichner 58
 Ancestor-Achse 59, 285, 1009
 Ancestor-Or-Self-Achse 59, 286, 1009
 Attribut-Achse 60, 67, 281, 299, 1009
 Bewegungsrichtung 58
 Child-Achse 59, 276, 299, 1012
 Descendant-Achse 59, 277, 299, 1013
 Descendant-Or-Self-Achse 59, 278, 1013
 Elementachsen 60, 1015
 Following-Achse 60, 280, 1018
 Following-Sibling-Achse 60, 279, 1018
 Namespace-Achse 60, 283, 1026
 Parent-Achse 59, 65, 284, 299, 1029
 Preceding-Achse 60, 288, 1029, 1030
 Preceding-Sibling-Achse 60, 287
 Reichweite 58
 Self-Achse 59, 64, 275, 299, 1032
 unabbreviated syntax 60
 vollständige Schreibweise 60
 Achsenschnitt 289, 296, 297, 299
 ancestor 298
 ancestor-or-self 298
 Architag XRay 930
 Arithmetische Ausdrücke
 Operatoren 250
 Arithmetische Ausdrücke
 Additive Ausdrücke 250
 Multiplikative Ausdrücke 251
 Rückblick auf XPath 1.0 251
 Arithmetische Ausdrücke 248
 Attribut 298
 Attribute Value Template 1010
 Attributset 1010
 Attributwert-Template 1010

Attributknoten 46, 48, 68
 Erzeugung 649
 Validierung 651
 Attributlisten 656
 Attributsammlungen 160
 Attributsets
 Konflikt zweier Attributdefini-
 tionen 658
 Attributwerte
 Umwandlung 156
 Weitergabe von 155
 Attributwert-Templates 41, 153
 Einschränkung der Verwendbarkeit
 158
 in XSLT 1.0 624
 in XSLT 2.0 630
 Maskierung der geschweiften
 Klammer 157
 Verwendung von 157
 XPath-Ausdrücke als Attributwerte
 154
 XPath-Ausdrücke, eingebettete 155
 Ausdruck
 Ausdruck vs. Instruktion 75
 Auswertungskontext 55
 Auswertungsreihenfolge von Bedin-
 gungen 317

B

Base64 1010
 Basic XSLT Processor 1010
 Basis-URI 1010
 Auflösen des Basis-URI 522
 in inkludierten Modulen 736
 Baumstruktur 43
 Elternknoten 44
 gerichteter Graph 44
 Kindknoten 44
 Knoten 44
 Verbindungen 44
 Bedingte Ausdrücke 248
 Bedingung
 xsl:if 719
 xsl:otherwise 775
 xsl:when 897
 Bedingungen
 mit choose 666
 Boolesche Ausdrücke 1011
 Boolesche Funktionen
 not() 117

Boolesche Negation 504, 1011
 Boolescher Wert 1011
 Built-In Template Rule → s. Template-
 regeln, eingebaute

C

casting function → s. Typumwand-
 lungsfunktionen
 CDATA 1011
 CDATA-Abschnitte 1011
 Character Data → s. CDATA
 Character Reference 1011
 Character-Map-Deklaration 665
 Character-Mapping 799
 deaktiviertes Output-Escaping 800
 Escaping für gemappte Zeichen 800
 child 277, 297, 298
 Child-Achse 299
 Collation 246, 1012
 Collation-basierter Vergleich 316
 Conditional Expressions 123
 context item 247
 context node 247
 context position 247
 count() 136
 current date and time 247
 Current Node 50
 Current Output Destination 1012
 current sequence 247
 Current Template Rule 1013
 current-captured-substring 248
 current-group 248
 current-grouping-key 248

D

Data Model
 Ressourcen 1007
 date formatting functions → s. Zeitfor-
 matfunktionen
 Datenmodell 49
 Datentypen 1013
 Datumswerte 318
 Debugging 576
 Defaultregel
 Element- und Dokumentknoten 91
 in XSLT 1.0 190
 in XSLT 2.0 190
 Kommentar- und PI-Knoten 91
 Text- und Attributknoten 91
 Defaultwert 1013
 descendant 298
 descendant-or-self 298
 Descendant-Or-Self-Achse 300

Deserialisierung 1013
 DocBook 1014
 Document Object Model 240
 Document Type Declaration 1014
 document-node() 293
 Dokumentbaum 50, 1014
 Dokumentelement 1014
 Dokumentknoten 45, 46, 160, 1014
 Dokumentreihenfolge 1014
 DOM 1015
 Doppelslash 271
 DSSSL 1015
 DTD 1015
 dynamischer Fehler 246
 dynamischer Kontext 244, 246

E

EDI 1015
 Einfügen mit xsl:include 213
 element() 293
 Elementknoten 45, 47, 273, 1015
 Erzeugung 690
 Erzeugung mit xsl:element 160
 Validierung 691
 else-expression 123
 Encodierung 1016
 Entities, externe 1016
 Entity 1016
 Unparsed Entity 1036
 Entity-Referenz 1016
 Ergebnisbaum 50, 1016, 1031
 Ergebnisdokument 27, 1016
 Ergebnisdokument, sekundäres 824
 Validierung 825
 Ergebnissequenz 50
 primäre Ergebnissequenz 53
 Erweiterungsfunktion 321
 EXSLT 322
 Prüfung auf Verfügbarkeit 438
 Erweiterungsinstruktion
 exsl:document 206
 saxon:output 206
 xt:document 206
 Existenzielle Quantifizierung 262
 Expanded Name 1017
 Expression 1017
 Externe Quellen
 Dokumente 201
 Entities 201

F

Fallback 1018
 Filterbedingungen 271, 302

Filterbedingungen für Location-Steps
110
fn:resolve-uri() 245
Formatierung
von Zahlen mit format-number 137
Formatierungsmuster 411
format-number() 137
Forwards-Compatible Processing 1018
Funktionen 1019
Erweiterungsfunktion 1017
Stringfunktionen 193
Sylesheetfunktion 197, 1034

G

Ganzzahldivision 252
Generierte Identifier → s. Laufzeit-
identifier
Glyphen 1019
GML 1019
Gruppierung 702
Current Group 153
Current Grouping Key 153
group-adjacent 703
group-by 703
group-ending-with 703
group-starting-with 703
Gruppierung mit xsl:for-each-group
150
Gruppierungskriterium 150, 703
Gruppierungsschlüssel 705
Kontextitem 705
Muenchsche Methode 150
Population Order 703
Sortierung 705
Verarbeitungsreihenfolge 705
Gültigkeitsbereich von Variablen und
Parametern 1019

H

HTML 1019
HTML-Framesets 206

I

IANA 1019
ID 1020
Identifier 218
Deklarierte Identifier 218
Fragment Identifier 1019
Laufzeitidentifier erzeugen mit
generate-id() 219
Laufzeitidentifier für Elemente 219
implicit timezone 247

Importieren
Attributsets 724
Globale Parameter und Variable 724
Importieren mit xsl:import 213
Importieren von externen Styles-
heets mit xsl:import 214
Importieren vs. Inkludieren 213
Importkaskade 723
Importpräzedenz 640, 723
Importzweig 641
Namespace-Aliasing 725
Outputdeklarationen 725
Position der Import-Deklaration 722
Präzedenz, niedrigere 641
Reaktivierung von Templaterregeln
640
Schlüsseldeklarationen 724
Stylesheetmodul 641
Templaterregeln 724
Vergleich mit Inkludieren 725
Whitespacebehandlung 725
XML-Schema-Definitionen 732
Zahlenformatdeklarationen 724

Importpräzedenz 735, 1020

Infoset 48

Information-Item 48

Ressourcen 1007

Inhaltsmodell 1020

Inkludieren

globale Parameter und Variable 736

Importpräzedenz 735

Konfliktfall 736

Schlüsseldeklarationen 736

Stylesheetmodule 735

in-scope attribute declaration 245

in-scope element declaration 245

in-scope schema definitions 245

in-scope type definition 245

Instanzieren 1020

Instruktion 1020

Ausdruck vs. Instruktion 75

ISO 1020

Item 242, 1021

Itemtest 310

J

JAXP 1021

JDK 1021

K

Kardinalität 1021

Key-Definition 1021

Key-Deklaration 461

- Kindknoten 1021
 - KindTest 289, 291, 1021
 - Allgemeiner KindTest 291
 - attribute(@*) 295
 - Attributknoten 295
 - document-node(elementtest) 293
 - Dokumentknoten 293
 - Elementknoten 293
 - in XPath 1.0 291
 - Kommentarknoten 292
 - Textknoten 292
 - Knoten 46, 242, 1027
 - Attributknoten 1010
 - Comment Node 1012
 - Current Destination Node 1012
 - Current Node 1012
 - Document Node 1014
 - Dokumentknoten 1014
 - Elementknoten 1015
 - Kindknoten 1021
 - Kommentarknoten 1022
 - Kontextknoten 1022
 - Kopieren von 164
 - Namenlose Knoten 1025
 - Namensraumknoten 1025
 - Textknoten 1035
 - Whitespace-Textnodes 1038
 - Knotenbezeichner 289
 - Knotentest 58, 289, 297
 - allgemeiner NameTest 62
 - KindTest 59
 - NameTest 59
 - Knotentyp 243, 289
 - Kommaoperator 307
 - Kommentar 248, 1022
 - Kommentarknoten 45, 160
 - Erzeugung 669
 - Komodo 954
 - Kompatibilität
 - XSLT 1.0 und XSLT 2.0 27
 - Konstruktorfunktionen 245, 595
 - nutzerdefinierte Datentypen 598
 - Ur-Typen 597
 - XML-Schema Datentypen 596
 - XPath-Datentypen 597
 - Kontextfunktionen
 - last() 116
 - position() 116
 - Kontextgröße 1022
 - Kontextposition 1022
 - Kopieren
 - Abstreifen der Attribute 165
 - Attributknoten 674
 - deep copy 167, 677
 - Dokumentknoten, flache Kopie 674
 - Dokumentknoten, tiefe Kopie 679
 - Einsatz von Attributsets 165
 - Elementknoten, flache Kopie 673
 - Elementknoten, tiefe Kopie 679
 - gleichzeitige Validierung 671
 - shallow copy 164
 - Textknoten 674
- L**
- Languagecode 1022
 - Laufzeitidentifizier 441, 442
 - Listentypen, 242
 - Literal Result Element 38, 1023
 - Leerzeichen 41
 - Zeilenumbrüche 41
 - Literal Result Elements 29
 - Literal Data Characters 40
 - Literale 307, 1023
 - Local Name 1023
 - Location-Path 1023
 - absoluter 1009, 1023
 - relativer 1023
 - Location-Step 56, 109, 289, 1023
 - Filterbedingung 109
 - Kandidatenknoten 111
 - Predicate 110
 - Logische Ausdrücke 248
- M**
- Maximum
 - Vergleich numerischer Werte 478
 - Vergleich von Datums- und Zeitwerten 479
 - Vergleich von Strings 479
 - Vergleich von Zeitdauerwerten 479
 - Metadaten 1024
 - Metainformationen 1024
 - MIME-Type 1024
 - Minimum
 - Vergleich numerischer Werte 481
 - Vergleich von Datums- und Zeitwerten 482
 - Vergleich von Strings 482
 - Vergleich von Zeitdauerwerten 482
 - Mixed Content 1024
 - Modularisierung 1024
 - Modulo-Operation 252
 - Mustervergleiche 173

N

Namensraum 33, 38, 1025
 Ausgabe des Namensraum-URI 494
 Auslesen des Namensraumpräfixes 458
 Default-Namensraum 38, 747, 1025
 Ersetzen 750
 Namensraumdeklaration 1025
 Namensraum-URI 1026
 namespace undeclaration 748
 Namespace-Fixup 748
 Null-Namensraum 40, 406, 1028
 XSLT-Namensraum 1041
 Namensraumdeklaration
 in inkludierten Modulen 737
 Namensräume 245
 Default Namespace 1013
 Ressourcen 1002
 Namensraumknoten 46, 160
 Defaultregel für Namensraum-
 knoten 92
 Erzeugung 746
 NameTest 289, 290, 1026
 NaN 1026
 NCName 1026
 Netbeans 942
 Node Test 1027
 node test 289
 node tests 297
 nodes 242
 nodes → s. Knoten
 Nodesequenz 243
 Zusammenstellung mit
 xsl:apply-templates 645
 Node-Sequenz, aktuelle 1027
 Nodeset 1027
 Nodetest 58, 271
 Nodetyp 1027
 Non Colonized Name 1028
 Normalisierung 1028
 Null-Attribut 294
 Number 1028
 Nummerieren 125
 alphabetische Bezifferung 127
 einfache Nummerierung 126
 führende Nullen 129
 römische Bezifferung 128
 Nummerierung
 count-Pattern 762
 einfache mit position() 141
 Formatierung 763
 level='any' 763
 level='multiple' 762

level='single' 762
 mit Buchstaben 768
 mit xsl:number und value-Attribut
 142
 mit Zahlworten 768
 mit Ziffern 767
 Nummerierung sortierter
 Sequenzen 141
 Ordinalform 764

O

OASIS 1028
 Operatoren für Sequenztypen 248
 Operatorfunktionen 609
 Operatorkeywords 250
 Operatorpräzedenz 249
 Operatorsymbole 609
 Output-Deklaration 777
 benannte Output-Deklaration 783
 CDATA-Sections 782
 Character-Mapping 785
 content-type-Description 784
 Doctype-Deklaration 782
 Einrückungen 782
 Escaping des Outputs 784
 JavaScript-Attribute 789
 method='html' 779
 method='text' 780
 method='xhtml' 778
 method='xml' 778
 MIME-Typangabe 783
 Namensraum-Undeklaration 785
 standalone-Attribut 781
 Unicode-Normalisierung 784
 Verhalten unter XSLT 1.0 786
 Verhalten unter XSLT 2.0 786
 version 780
 XML-Deklaration 781
 Zeichenencoding 781
 Output-Escaping 125
 Output-Methode 1028

P

Parameter 801, 1028
 Datentyp des Parameterwertes 803
 Defaultwert 802, 805
 Funktionsparameter 199
 Globale Parameter 805
 Lokale Parameter 806
 Namensgleichheit 802
 Parameter in Stylesheetfunktionen
 810
 required 803

- Shadowing 802
 - tunnel 804
 - Tunnelparameter 185, 809
 - Tunnelstrom 183
 - Wertübergabe an globale Parameter 805
 - Wertübergabe an lokale Parameter 807
 - Wertübergabe aus `xsl:apply-imports` 808
 - Wertübergabe aus `xsl:apply-templates` 807
 - Wertübergabe aus `xsl:call-template` 807
 - Wertübergabe aus `xsl:next-match` 809
 - Parameterübergabe 899
 - an aktivierte Templaterregeln 643
 - an benanntes Template 662
 - Durchtunneln von Defaultregeln 190, 872
 - Tunnelparameter 643, 903
 - Verwendungskontext 902
 - Wertermittlung 901
 - `xsl:with-param` 183
 - parent 298
 - Parent-Achse 299
 - Parser, validierender 1037
 - Parser, XML-Parser 1040
 - Path Expression 1029
 - Pattern 1029
 - Oder-Operator 89
 - PCDATA 1029
 - Pfadausdrücke 53, 61, 109, 248
 - absolute 271
 - absoluter Pfadausdruck 56
 - Doppelslash 271
 - einfache 297
 - einfacher Pfadausdruck 56
 - Grundlagen 270
 - Inklusivität 57
 - kontextunabhängig 271
 - Location-Steps 289
 - relative 271
 - relativer Pfadausdruck 55
 - zusammengesetzte 298
 - zusammengesetzter Pfadausdruck 56
 - Pfadausdrücke → s. Achsen
 - Pfadausdrücke → s. Doppelslash
 - Pfadausdrücke → s. Elementknoten
 - Pfadausdrücke → s. Location-Steps
 - Pfadausdrücke → s. Processing-Instruction-Knoten
 - Pfadausdrücke → s. Textknoten
 - picture string → s. Zeitformatfunktionen
 - Formatierungsmuster
 - Platzhalter → s. Wildcard
 - Portabilität 1029
 - Post Schema Validation Infoset 49
 - Präfix 33
 - Preceding-Siblings 127
 - Predicate 289, 302, 1030
 - allgemeiner 303
 - Ergebnissequenz 305
 - Kontextfunktionen 115
 - Logische Verknüpfung von Bedingungen 114
 - mehrere 304
 - numerischer 304
 - numerisches Predicate 113
 - Operatoren 115
 - Sequenzfunktionen 115
 - Stringfunktionen 115
 - Wertvergleich 113
 - XPath-Funktionen 115
 - Predicate-Ausdruck 302
 - Predicates → s. Filterbedingungen
 - principal node kind 290
 - Principal Node Type 1030
 - Processing-Instruction 1030
 - Erzeugung 819
 - Syntax in SGML 823
 - Processing-Instruction-Knoten 46, 273, 292
 - Prolog 1031
 - Prozessanweisungen 160
 - PSVI 1031
- Q**
- QName 34, 35, 1031
 - expandierter QName, Export von 34, 245
 - lokaler Bezeichner 34
 - Quantifizierende Ausdrücke 248
 - Quelldokument 27, 1031
- R**
- RegExp-Funktionen
 - matches() 118
 - Reguläre Ausdrücke 472
 - Backreferences 475
 - dot-all 473
 - dot-all-Modus 634

- Flag 472
 - Flags 634
 - ignore 473
 - Mehrzeilenmodus 473
 - Metazeichen 636
 - multiline 473
 - Multiline-Modus 634
 - numerische Quantifizierer 637
 - Quantifier 474
 - Quantifizierer, greedy 476
 - Quantifizierer, reluctant 476
 - regex-group() 742
 - Rekursiver Templateaufruf vs. regulärer Ausdruck 194
 - Stringmodus 473, 634
 - Subgruppen 512
 - Subpattern 742
 - Whitespace 473
 - Whitespace-Zeichen 635
 - Result Tree → s. Ergebnisbaum
 - Result Tree Fragment 265, 1031
 - Reverse Document Order 1031
 - Root Elements 1032
 - Root Node 1032
 - Runden 135
 - Runden mit XPath 137
 - Rundung
 - nach Präzisionsangabe 528
- S**
- Saxon
 - Instant Saxon 911
 - Saxon 7 27, 911
 - Arbeiten mit kompilierten Stylesheets 923
 - Basic XSLT Processor 912
 - Fehlerbehandlung 923
 - globale Parameter 916
 - Installation 911
 - Java-Tracer 921
 - Kommandozeile 913
 - kompilierte Stylesheetdatei 918
 - Timinginformationen 922
 - Tinytree-Modell 918
 - Tracinginformation 921
 - Validierung 922
 - Verarbeitung kompletter Ordnerinhalte 917
 - Zeilennummern 919
 - Schaltsekunden 532
 - Schema-Aware-XSLT-Processor 733, 1032
 - Schleife
 - mit xsl:for:each 698
 - Sortierung 699
 - Schleifen in XSLT
 - xsl:for-each 144
 - Schleifenausdrücke 248
 - Schlüssel 218
 - Deklaration von Schlüsseln 223
 - Externe keys 226
 - Gegenüberstellung
 - Keys-ID 229
 - Gleichnamige Schlüssel verschmelzen 228, 229
 - Keys in importierten Stylesheets 228
 - Listen von Schlüsseln zur Laufzeit 223
 - xsl:key 219
 - Schlüsseldefinition 461, 1032
 - Schlüsseldeklaration 738, 739
 - Import und Inklusion 739
 - Schlüsselname 738
 - Schlüsselwert 739
 - Schreibweise, abgekürzte 1009
 - Scope 1032
 - Seiteneffekte 174
 - self 275, 276, 298
 - Self-Achse 299
 - sequence order 384
 - SequenceType-Matching 309
 - Sequenz 57, 79, 1032
 - aktuelle Sequenz 50
 - Bildung einer Teilsequenz 548
 - Entfernen eines Items 513
 - Klammern 79
 - Knotendubletten 82
 - Kommaoperator 79
 - leere Sequenz 80
 - Literale in einer Sequenz 81
 - Optimierung 581
 - primäre Ergebnissequenz 206
 - sekundäre Ergebnissequenz 206
 - Sequenzkonstruktor 1032
 - Singleton-Sequenz 82
 - Umkehrung der Reihenfolge 524
 - Sequenzausdrücke 248, 307
 - Sequenzprüfung
 - Prüfung auf genau ein Item 403
 - Prüfung auf kein oder ein Item 594
 - Zurückweisung der leeren Sequenz 510
 - Sequenztypen 309
 - Kardinalität 310
 - Testausdrücke 310

- Serialisierung 52, 1032
- SGML 1032
- Singleton-Sequenz 244
- Slash
 - doppelter 301
 - einfacher 302
- SMIL 1033
- Sonic Stylus Studio 5.1 961
- Sortieren von Sequenzen
 - xsl:for-each 147
- Sortierung
 - alphabetisch 837
 - Ausnahmewerte 839
 - case-order 837
 - data-type='number' 836
 - data-type='text' 836
 - Gewöhnliche Werte 839
 - in xsl:apply-templates 839
 - in xsl:for-each and
 - xsl:for-each-group 839
 - in xsl:perform-sort 840
 - Kontextgröße 838
 - Kontextposition 838
 - Leere Werte 839
 - mit xsl:perform-sort 814
 - numerisch 140, 837
 - primärer Sortierschlüssel 838
 - Schleife 699
 - Sortierschlüssel 835
 - Sprache 837
 - von Sequenzen 138
- Sprachangabe 464
- Standalone 1033
- Standardattribute
 - exclude-result-prefixes 628
 - extension-element-prefixes 629
 - in XSLT 1.0 623
 - in XSLT 2.0 627
 - version 629
 - xml:lang 623
 - xml:space 623
 - xpath-default-namespace 630
- statically-known collections 245
- statically-known documents 245
- statischer Kontext 244
- String 1033
 - Anzahl der Zeichen 545
 - Austausch von Zeichen 578
 - Tokenizing 573
 - Umwandlung in Großbuchstaben 583
 - Umwandlung in String 538
- Stringbegrenzer 112
- Stringfunktionen
 - contains() 118
 - ends-with() 118
 - starts-with() 118
- Stringvergleich 316
- Stringwert 42, 71, 1033
 - eines Elementknotens 66
- Stringwerte 66
- Stylesheet 1033
 - default-validation 855
 - default-validation='lax' 856
 - default-validation='strict' 856
 - default-validation='strip' 856
 - default-validation='preserve' 856
- Deklaration der Namensräume 858
- Embedded Stylesheet 1015
- exclude-result-prefixes 853
- extension-element-prefixes 853
- festgelegte Namensraum-URIs 859
- Modulare Stylesheets 212
- Namensraumdefinitionen 852
- Reihenfolge der Deklarationen 857
- Vereinfachtes Stylesheet 1038
- version-Attribut 857
- xmlns:xsl 33
- xpath-default-namespace 854
- Stylesheetfunktion 712
 - Ergebnisausgabe 714
 - Fokus 714
 - Funktionsparameter 715
 - override 713
 - Sequenztyp 713
 - und globale Variable 714
 - und Tunnelparameter 714
- Summieren 125, 135
- Summieren mit XPath 136
- SVG 1034
- Syntax 1009
- System Properties 1034
- Systemeigenschaft
 - Basic XSLT Processor 569
 - Ermittlung der Systemeigenschaften 568
 - Herstellerbezeichnung 568
 - Name der Implementierung 568
 - Rückwärtskompatibilität 569
 - Schema Aware Processor 569
 - Serialisierung 569
 - Versionsnummer 568
 - XSLT-Version 568

T

Tag 1034
 Template
 Aktuelle Templaterregel 642
 Aufruf benanntes Template 661
 Benannte Templates 866
 Defaultmodus 864
 Defaultregel 641
 Defaulttemplate 647
 Default-Templaterregeln 90, 870
 Defaulttemplates 90
 Durchtunnelung der Defaultregeln 872
 mode='#current' 646
 mode='#default' 646
 mode-Attribut 191
 natürliche Priorität 864, 867
 Präzedenz 91
 Priorität 863
 Priorität von Templaterregeln 1030
 Rekursiver Templateaufruf vs. regulärer Ausdruck 194
 Sequenzkonstruktor 620
 Sequenztyp 865
 sticky mode 647
 Stylesheets mit mehreren Templates 69
 Template-Body 1034
 Templatekörper 620
 Template-Mode 1035
 Templatemodes 191
 Templatemodus 641, 864
 Templatemodus, Tokenliste 864
 Templaterregel 36, 1035
 Templaterregel für zwei Elemente 89
 Templaterregeln 866
 Templaterumpf 36, 865
 Template, benanntes 1034
 Templaterregel 50, 173
 Aktivierung mit next-match 754
 aktuelle Templaterregel 50
 Bestimmung der Priorität 755
 eingebaute 1035
 Instanzierung 37, 1011
 natürliche Priorität 755
 Temporärer Baum 53
 Validierung 687
 test-expression 123
 Textdeklaration 1035
 Textknoten 45, 47, 160, 273
 Token 1035
 Tokenliste 1036
 Toplevel-Element 1036

total order 384
 Treebeard 936
 Tunnelstrom 660
 Typfehler 1036
 Typgültigkeit 1036
 Typ-Promotion 564
 Typprüfung 258
 Typstrukturangabe, Einschränkung 295, 296
 Typumwandlung 259, 1036
 ausgehend von Booleschen Werten 607
 ausgehend von Stringtypen 607
 ausgehend von Zahlentypen 607
 nach numerischen Typen 607
 nach Typen der Zeitdauer 607
 nach Typen für Datum und Zeit 608
 nach xdt:untypedAtomic 606
 nach xs:anySimpleType 606
 nach xs:anyURI 608
 nach xs:base64Binary und xs:hexBinary 608
 nach xs:boolean 608
 nach xs:QName 609
 nach xs:string 606
 von xs:anySimpleType 605
 von xs:anyURI 606
 von xs:datetime, xs:date oder xs:time 606
 von xs:string 605
 xs:QName 606
 Typumwandlungsfunktionen 598
 abgeleitete Typen in Basistypen 604
 Crosscasting 605
 Downcasting 605
 innerhalb eines Typhierarchie-zweigs 605
 lexikalische Übereinstimmung 599
 Stammtypen 605
 Upcasting 605
 zwischen primitiven Typen 598
 zwischen Typhierarchie-zweigen 605

U

Unicode 1036
 FULLY-NORMALIZED 502
 Kanonische Komposition 502
 Kompositzeichen 503
 NFC 502
 Output-Normalisierung 784
 Unicode Case Mapping 470
 Unicode Normalisation Form C 502
 Unicode SpecialCasing 470

- Unicode Standard 470
- Unicode Surrogate Pair 546
- Unicode-Codepoint 547
- Unicode-Normalisierungsformen 503
- Zeichen Kategorie L 470
- Zeichen Kategorie P 470
- Universale Quantifizierung 263
- Unparsed Entity 204
- URI 1037
 - Basis-URI 1010
- URI-Escaping 400
 - alphanum 401
 - control characters 402
 - escape-reserved 400
 - Escapesequenzen 400
 - marks 401
 - nicht reservierte Zeichen 401
 - reservierte Zeichen 402
- URL 1037
- URN 1037
- UTF-8, UTF-16 1037

- V**
- Validierung 1037
- Variable 887, 1037
 - Benennung und Referenzierung 890
 - binding shadow 178
 - Globale Variablen 891
 - Gültigkeitsbereich 175, 890
 - Lokale Variablen 892
 - Namensgleichheit 888
 - Referenz in Variablendeklaration 892
 - Sequencetype 889
 - temporäre Bäume 179
- Variablendeklarationen 245
- Verarbeitung
 - Rückwärtskompatible Verarbeitung 907
 - Vorwärtskompatible Verarbeitung 907
- Vergleichsausdrücke 248, 252
- Verknüpfung von Sequenzen 248
- Visual XSLT 946, 969
- vorwärtskompatible Verarbeitung 696

- W**
- Well-balanced 1038
- Well-formed 1038
- Whitespace 1038
- Whitespace-Nodes 77
 - beibehalten 817
 - entfernen 849
 - schützen 875
- Whitespace-Stripping 1039
- Whitespacezeichen
 - Normalisierung 500
- Wildcard 62, 1029, 1039
- Wohlgeformtheitsregeln 1039
- Wurzelement 1039
- Wurzelknoten 301

- X**
- XML Cooktop 932
- XML Information Set 240
- XML Schema 27, 240, 312, 1040
 - Ressourcen 1002
- xml:lang 157
- XML-Datenbank 1039
- XML-Deklaration 1040
- XML-Dokumente, Gültigkeit 1019
- XML-Ressourcen 1001
- XMLSpy 2004 966
- XPath 26, 53, 173, 1040
 - Dokumentreihenfolge 240
 - Filtern von Sequenzen 109
 - Filterung einer Sequenz 111
 - for ... in ... return 149
 - Operatoren 242
 - Ressourcen 1005
 - Schlüsselwörter 242
 - Symbole 242
 - XPath-Ausdruck 43
 - XPath-Pattern 43
- XPath 1.0 252, 259, 269, 290, 295
 - Abwärtskompatibilität 245
 - Boolescher Wert 314
 - Explizite Zahlentypen 315
 - Vergleich mit XPath 2.0 311
 - Zahl 314
- XPath 1.0 compatibility mode 245
- XPath 2.0 239
 - Kommentare 268
 - Kompatibilitätsmodus 313
 - Umwandlung von String nach Zahl 315
 - Umwandlung von Zahl nach String 315
- XPath → s. Ganzzahldivision
- XPath → s. Modulo-Operation
- XPath als Pfadbeschreibungssprache 239
- XPath Data Model 240

- XPath-Ausdrücke 241
 - Auswertungskontext 244
 - Bedingte Ausdrücke 261
 - Bedingungen 123
 - Fokus 246
 - Fokuskomponenten 247
 - Operatoren für Sequenztypen 258
 - Operatoren und Keywords 248
 - Quantifizierende Ausdrücke 261
 - Schleifenausdrücke mit for 264
 - Vergleichsausdrücke 252
- XPath-Ausdrücke → s. Arithmetische Ausdrücke
- XPath-Ausdrücke → s. Bedingte Ausdrücke
- XPath-Ausdrücke → s. Collations
- XPath-Ausdrücke → s. dynamischen Kontext
- XPath-Ausdrücke → s. dynamischer Fehler
- XPath-Ausdrücke → s. Dynamischer Kontext
- XPath-Ausdrücke → s. Existenzielle Quantifizierung
- XPath-Ausdrücke → s. Kommentare
- XPath-Ausdrücke → s. Logische Ausdrücke
- XPath-Ausdrücke → s. Namensräume
- XPath-Ausdrücke → s. Pfadausdrücke
- XPath-Ausdrücke → s. Quantifizierende Ausdrücke
- XPath-Ausdrücke → s. Result Tree Fragment
- XPath-Ausdrücke → s. Schleifenausdrücke
- XPath-Ausdrücke → s. Sequenzausdrücke
- XPath-Ausdrücke → s. statischer Kontext
- XPath-Ausdrücke → s. Universale Quantifizierung
- XPath-Ausdrücke → s. Vergleichsausdrücke
- XPath-Ausdrücke → s. Verknüpfung von Sequenzen
- XPath-Funktion
 - Namensräume 323
- XPath-Funktionen
 - Aggregation 330
 - Argumentanzahl (arity) 324
 - Argumente 324
 - Argumentklammern 323
 - Arithmetische Ausdrücke 250
 - Aufruf 323
 - Ausgabeformatierung 328
 - Boolesche Funktionen 328
 - Codepoint-Funktionen 329
 - Core-Funktionen 321
 - Fehlermeldungen und Debugging 334
 - fn:abs 335
 - fn:adjust-dateTime-to-timezone 339
 - fn:adjust-date-to-timezone 337
 - fn:adjust-time-to-timezone 342
 - fn:avg 345
 - fn:base-uri 347
 - fn:boolean 348
 - fn:ceiling 137, 350
 - fn:codepoints-to-string 352
 - fn:collection 353
 - fn:compare 354
 - fn:concat 357
 - fn:contains 361
 - fn:count 135, 364
 - fn:current-date 369
 - fn:current-dateTime 370
 - fn:current-time 373
 - fn:data 374
 - fn:day-from-date 375
 - fn:day-from-dateTime 376
 - fn:days-from-dayTimeDuration 377
 - fn:deep-equal 379
 - fn:default-collation 382
 - fn:distinct-values 383
 - fn:doc 386
 - fn:document-uri 392
 - fn:empty 395
 - fn:ends-with 396
 - fn:error 398
 - fn:escape-uri 399
 - fn:exactly-one 403
 - fn:exists 405
 - fn:expanded-QName 406
 - fn:false 407
 - fn:floor 137, 408
 - fn:hours-from-dateTime 443
 - fn:hours-from-dayTimeDuration 445
 - fn:hours-from-time 446
 - fn:id 222, 448
 - fn:idref 452
 - fn:implicit-timezone 455
 - fn:index-of 456
 - fn:in-scope-prefixes 458
 - fn:insert-before 459

- fn:lang 463
- fn:last 141, 466
- fn:local-name 467
- fn:local-name-from-QName 469
- fn:lower-case 469
- fn:matches 472
- fn:max 477
- fn:min 480
- fn:minutes-from-dateTime 483
- fn:minutes-from-dayTimeDuration 484
- fn:minutes-from-time 486
- fn:month-from-date 487
- fn:month-from-dateTime 488
- fn:months-from-yearMonthDuration 489
- fn:name 491
- fn:namespace-uri 494
- fn:namespace-uri-for-prefix 497
- fn:namespace-uri-from-QName 497
- fn:node-name 498
- fn:normalize-space 499
- fn:normalize-unicode 501
- fn:not 504
- fn:number 506
- fn:one-or-more 509
- fn:position 510
- fn:remove 513
- fn:replace 514
- fn:resolve-QName 519
- fn:resolve-uri 522
- fn:reverse 524
- fn:root 525
- fn:round 137, 526
- fn:round-half-to-even 137, 528
- fn:seconds-from-dateTime 531
- fn:seconds-from-dayTimeDuration 532
- fn:seconds-from-time 534
- fn:starts-with 535
- fn:string 538
- fn:string-join 542
- fn:string-length 194, 545
- fn:string-to-codepoints 547
- fn:subsequence 547
- fn:substring 550
- fn:substring-after 194, 553
- fn:substring-before 194, 557
- fn:subtract-dateTimes-yielding-dayTimeDuration 560
- fn:subtract-dateTimes-yielding-yearMonthDuration 562
- fn:sum 136, 563
- fn:timezone-from-date 570
- fn:timezone-from-dateTime 570
- fn:timezone-from-time 571
- fn:tokenize 196, 572
- fn:trace 576
- fn:translate 194, 577
- fn:true 580
- fn:unordered 581
- fn:upper-case 583
- fn:year-from-date 591
- fn:year-from-dateTime 592
- fn:years-from-yearMonthDuration 593
- fn:zero-or-one 594
- Funktionen für Sequenzen 332
- Funktionen zur Assoziation von Ressourcen 333
- Funktionssignaturen 324
- Kontextinformationen – Datum und Zeit 331
- Kontextinformationen innerhalb einer Sequenz 333
- Nodennamen, Identifier und URI-Informationen 331
- Nummerieren 125
- Präfix 323
- RegEx-Funktionen 329
- Rundungsfunktionen 328
- Stringfunktionen 328
- Stylesheetfunktionen 322
- Syntax 115
- Typbindung 116
- Typkonvertierung 328
- Überladen 323
- Verwendungsgebiet 324
- Zeit- und Datumswerte 330
- Zeitzone 331
- XPath-Kommentar 124
- XPath-Operatoren, Präzedenz 249
- XPath-Pattern 36
- XPointer 1041
- XQuery 269, 272, 1041
 - Ressourcen 1005
- XQuery 1.0 311
- xsi:nil 294
- XSL 25
- XSL und XSLT
 - Ressourcen 1003
- xsl:analyze-string 158, 248
- xsl:apply-templates 109
- xsl:attribute 158
- xsl:element 158

- xsl:for each
 - innerhalb von xsl:for-each-group 152
- xsl:for-each-group 158, 248
- xsl:import 213
- xsl:include 213
- xsl:message 158
- xsl:namespace 158
- xsl:number 158
- xsl:processing-instruction 158
- xsl:result-document 158, 206
- xsl:sort 158
- xsl:strip-space 128
- xsl:value-of 109, 149, 158
- xsl:xpath-default-namespace 157
- XSL-Formatierungsobjekte 1018
- XSLT 25
 - Standard 17
- XSLT 2.0
 - Abwärtskompatibilität 239
 - Neue Instruktionen 625
 - Neue Toplevel-Elemente 625
 - zusätzliche Standardattribute 627
- XSLT als Host-Language für XPath 239
- XSLT und XPath
 - Bedingungen 119
- XSLT-Elemente 119
 - Deklarationen 620
 - Erweiterungsinstruktion 1017
 - Funktionsgruppen 631
 - Hierarchie der XSLT 1.0-Elemente 622
 - Hierarchie der XSLT 2.0-Elemente 627
 - Instruktionen 32, 620
 - Root-Elemente 32, 619
 - Sub-Instruktionen 621
 - Toplevel-Elemente 32, 620
- xslf 719
- xsl:analyze-string 633
- xsl:apply-imports 216, 640
- xsl:apply-templates 72, 88, 145, 148, 193, 203, 208, 645
- xsl:attribute 162, 649
- xsl:attribute-set 163, 656
- xsl:call-template 181, 660
- xsl:character-map 664
- xsl:choose 119, 121, 666
- xsl:comment 669
- xsl:copy 164, 670
- xsl:copy, Vor- und Nachteile 166
- xsl:copy-of 164, 677
- xsl:decimal-format 682
- xsl:document 686
- xsl:element 690
- xsl:fallback 695
- xsl:for-each 145, 698
- xsl:for-each:group-by-Attribut 150
- xsl:for-each-group 151
- xsl:function 712
- xsl:if 119
- xsl:import 722
- xsl:import-schema 732
- xsl:include 734
- xsl:key 158, 219, 223, 224, 737
- xsl:key, use-Attribut 224
- xsl:matching-substring 741
- xsl:message 744
- xsl:namespace 746
- xsl:namespace-alias 749
- xsl:next-match 754
- xsl:non-matching-substring 759
- xsl:number 126, 127, 761
- xsl:number, count-Attribut 129
- xsl:number, Formatstring 130
- xsl:number, level-Attribut 131
- xsl:otherwise 121, 774
- xsl:output 95, 777
- xsl:output-character 799
- xsl:param 174, 801
- xsl:perform-sort 813
- xsl:preserve-space 99, 817
- xsl:processing-instruction 819
- xsl:result-documen
 - Outputdeklaration 211
- xsl:result-document 206, 209, 824
- xsl:sequence 832
- xsl:sort 138, 148, 153, 835
- xsl:strip-space 98, 849
- xsl:stylesheet 32, 851
- xsl:template 35, 863
- xsl:text 77, 875
- xsl:text, Unterdrückung des Output-
 - Escapings 877
- xsl:transform 35, 880
- xsl:value-of 31, 42, 124, 881
- xsl:value-of, separator-Attribut 82
- xsl:variable 174, 887
- xsl:when 897
- xsl:with-param 899
- xslfor-each-group 702
- xsl
 - umber, from-Attribut 134
- XSLT-Entwicklungsumgebungen 929
- Architag XRay 930
- Komodo 954

- NetBeans 942
- Sonic Stylus Studio 5.1 961
- Treebeard 936
- Visual XSLT 946
- XML Cooktop 932
- XML-Spy 2004 966
- XSLT-Funktionen
 - current 365
 - current-group 371
 - current-grouping-key 372
 - document 202, 227, 388
 - element-available 393
 - format-date 410
 - format-dateTime 424
 - format-number 429
 - format-time 433
 - function-available 438
 - generate-id 219, 441
 - Information über die Laufzeitumgebung 334
 - key 219, 224, 226, 461
 - regex-group 511
 - system-property 567
 - unparsed-entity-public-id 585
 - unparsed-entity-uri 203, 587
 - unparsed-text 588
- XSLT-Instruktion 31
- XSLT-Instruktionen zur Knotenerzeugung 159
- XSLT-Namensraum
 - Maskierung 750
- XSLT-Prozessor 34, 1041
- XSLT-Transformation 246
 - Verarbeitungsablauf 109
- Wert 'unendlich' 683
- Wert NaN 683
- Zeichenersetzung 584
- Zeilenende, im Quelltext 1042
- Zeitformatfunktionen 412
 - component specifier 412
 - Formatierungsmuster 411
 - GMT-Zeit 414
 - Gregorianischer Kalender 417
 - islamischer Kalender 417
 - jüdischer Kalender 417
 - Julianischer Kalender 417
 - Kalender nach ISO 8601 418
 - Kalenderangabe 417
 - Komponentensymbole 413
 - Modifikatoren 414
 - primäre Modifikatoren 415
 - sekundäre Modifikatoren 415
 - UTC-Zeit 414
 - Zeitzonensymbole 414

Z

- Zählen 125, 135
- Zahlen formatieren
 - format-number 137
- Zahlenformatierung 682
 - Dezimalformat-Deklaration 682
 - Dezimaltrennsymbol 430, 683
 - Formatierungsstring 430
 - format-number() 684
 - Gruppierungssymbol 430
 - Präfix und Suffix des Formatierungsstrings 430
 - Promillewert 683
 - Prozentwert 683
 - Regionalisierung 685
 - Submuster für negative Zahlen 431
 - Tausendertrennzeichen 683
 - Vorzeichen 683