helion.pl

IDŹ DO
PRZYKŁADOWY ROZDZIAŁ
SPIS TREŚCI

KATALOG KSIĄŻEK
KATALOG ONLINE
ZAMÓW DRUKOWANY KATALOG

TWÓJ KOSZYK
DODAJ DO KOSZYKA

CENNIK I INFORMACJE
ZAMÓW INFORMACJE O NOWOŚCIACH
ZAMÓW CENNIK

CZYTELNIA
FRAGMENTY KSIĄŻEK ONLINE

XSLT dla każdego



Autor: Michiel van Otegem
Tłumaczenie: Tomasz Żmijewski
ISBN: 83-7197-785-9
Tytuł oryginału: TY XSLT in 21 Days
Format: B5, stron: 578
Przykłady na ftp: 115 kB

Język XSLT jest jedną z najważniejszych technologii związanych z XML. Za pomocą XSLT możemy przekształcić dokumenty XML, możemy także wygenerować plik HTML lub nawet zwykły plik tekstowy. Specyfikacja tego języka opracowana przez konsorcjum W3C nie jest łatwa w lekturze. Nawet jeśli programowałeś wcześniej w języku zorientowanym obiektowo lub języku strukturalnym, zrozumienie zasady działania XSLT może nastręczać pewne trudności. XSLT został bowiem oparty o zupełnie inny paradygmat, który poznasz w czasie lektury tej książki.

Książka „XSLT dla każdego" w przystępny sposób pomoże zrozumieć i opanować język XSLT. Od czytelników wymaga się jedynie podstawowej znajomości XML i technologii z XML związanych. Podczas pracy z XSLT będziemy jednocześnie pogłębiali swoją znajomość XML. Nauczymy się przekształcać dokumenty XML na kod HTML, dzięki czemu można je przeglądać w przeglądarce, nauczymy się też przetwarzać jedną postać XML na inną.

Przedstawiono:

- Zalety, wady i obszary zastosowania XSLT
- Narzędzia ułatwiające tworzenie i debugowanie plików XSLT, procesory XSLT: MSXML, Saxon, Xalan
- Wybieranie fragmentów dokumentu XML za pomocą XPath
- Użycie reguł, wstawianie tekstu i elementów
- Przetwarzanie warunkowe i iteracyjne, zastosowanie rekurencji
- Tworzenie dokumentów HTML
- Użycie zmiennych i parametrów
- Operacje na łańcuchach, formatowanie danych, sortowanie i numerowanie
- Tworzenie arkuszy złożonych z wielu plików
- Zagadnienia związane z przestrzeniami nazw
- Metodologię projektowania aplikacji opartych na XML i XSLT

Książka „XSLT dla każdego" jest przeznaczona dla osób uczących się XSLT, ale nie tylko dla nich. Przydatna będzie też dla osób znających już podstawy XSLT, gdyż wiele zagadnień omawiamy bardzo szczegółowo. To kompletny przewodnik po tym języku programowania, od podstaw po zagadnienia zaawansowane.

Wydawnictwo Helion
ul. Chopina 6
44-100 Gliwice
tel. (32)230-98-63
e-mail: helion@helion.pl

Spis treści

O Autorze **17**

Wstęp **19**
Dla kogo jest ta książka? 19
Co trzeba wiedzieć przed rozpoczęciem lektury? 20
Czego można się z tej książki nauczyć? 20
Jakie oprogramowanie jest potrzebne do zrealizowania przykładów z tej książki? 20
Jak zorganizowana jest ta książka? 20
Konwencje przyjęte w książce 21

Część I **23**

Rozdział 1. Rozpoczynamy naukę XSLT **25**
XSLT w skrócie 25
Wprowadzenie do XML i XSLT 26
XSLT i rodzina XML 29
Zalety XSLT 33
Kiedy nie należy używać XSLT 37
Jak działa XSLT? 38
Przekształcanie XSLT 38
Programowanie deklaratywne 41
Tworzenie plików XSLT 42
Użycie edytora tekstowego 42
Użycie edytora XML 42
Edytory i debuggery XSLT 43
Narzędzia projektowe XSLT 44
Procesory przekształcające XML za pomocą XSLT 44
MSXML 44
Saxon 45
Xalan 46
Podsumowanie 47
Pytania i odpowiedzi 47
Zadania 48
Quiz 48
Ćwiczenie 48

Rozdział 2. Pierwsze przekształcenia XML **51**
Anatomia arkusza stylów 51
Czym jest arkusz stylów? 51
Podstawowe składniki arkusza stylów 52
Uproszczona składnia arkuszy stylów 61

Wykonywanie arkusza stylów na dokumencie XML ... 62
Łączenie arkusza stylów z dokumentem XML ... 63
Zanurzanie arkuszy stylów w dokumentach XML ... 65
Programowe wykonywanie arkusza stylu ... 66
Podsumowanie ... 68
Pytania i odpowiedzi ... 68
Zadania ... 68
Quiz ... 69
Ćwiczenie ... 69

Rozdział 3. Wybieranie danych ... 71
Drzewa dokumentów XML ... 71
Co to jest węzeł? ... 72
Czym jest zbiór węzłów? ... 74
XPath ... 75
Wybieranie elementów ... 75
Wybieranie atrybutów ... 81
Techniki bardziej zaawansowane ... 82
Podsumowanie ... 88
Pytania i odpowiedzi ... 89
Zadania ... 89
Quiz ... 90
Ćwiczenie ... 90

Rozdział 4. Użycie reguł ... 91
Reguły ... 91
Reguły z bliska ... 91
Korzyści wynikające ze stosowania reguł ... 93
Tworzenie i użycie reguł ... 98
Więcej o dopasowywaniu reguł ... 98
Użycie nazwanych reguł ... 102
Sprawdzanie, której reguły użyto ... 103
Różne reguły na różne przypadki ... 103
Priorytety reguł ... 105
Definiowanie własnych priorytetów ... 108
Podsumowanie ... 109
Pytania i odpowiedzi ... 109
Zadania ... 110
Quiz ... 110
Ćwiczenia ... 110

Rozdział 5. Wstawianie tekstu i elementów ... 111
Wstawianie tekstu ... 111
Tekst ze znakami specjalnymi ... 112
Znaki specjalne XSLT ... 114
Wstawianie elementów i atrybutów ... 116
Wstawianie elementów ... 116
Wstawianie atrybutów ... 124
Kopiowanie elementów z dokumentu źródłowego ... 129
Kopiowanie jedynie węzła kontekstu ... 129
Kopiowanie zbiorów węzłów i fragmentów drzewa ... 131
Wstawianie komentarzy i instrukcji przetwarzania ... 132
Wstawianie komentarzy ... 133
Wstawianie instrukcji przetwarzania ... 134

Podsumowanie ... 135
Pytania i odpowiedzi ... 136
Zadania ... 136
Quiz ... 136
Ćwiczenie ... 136

Rozdział 6. Przetwarzanie warunkowe i iteracyjne ... 137
Iteracja po zbiorze węzłów ... 137
Przetwarzanie kolejnych węzłów ze zbioru ... 138
Filtrowanie zbiorów węzłów ... 141
Użycie funkcji operujących na zbiorach węzłów ... 143
Przetwarzanie warunkowe ... 144
Proste przetwarzanie warunkowe ... 144
Przetwarzanie warunkowe — wiele możliwości ... 147
Jeszcze o wyrażeniach ... 149
Formalizm wyrażeń ... 149
Użycie wielu predykatów ... 150
Łączenie wyrażeń ... 151
Użycie funkcji logicznych ... 152
Negacja wyniku wyrażenia ... 153
Konwersja na wartości logiczne ... 153
Podsumowanie ... 154
Pytania i odpowiedzi ... 154
Zadania ... 155
Quiz ... 155
Ćwiczenia ... 155

Rozdział 7. Kontrola postaci danych wynikowych ... 157
Różne formaty wynikowe ... 157
Dane wynikowe XML ... 158
Format wynikowy HTML ... 160
Dane wynikowe w innych formatach ... 163
Określanie typu nośnika ... 166
Kodowanie wynikowe i cytowanie ... 166
Określanie kodowania wynikowego ... 167
Kodowanie nieobsługiwanych znaków ... 167
Wyłączanie cytowania ... 168
Obsługa białych znaków ... 170
Jak są obsługiwane białe znaki ... 170
Odrzucanie białych znaków z dokumentu źródłowego ... 171
Obsługa białych znaków w arkuszu ... 175
Wcięcia ... 177
Podsumowanie ... 178
Pytania i odpowiedzi ... 179
Zadania ... 179
Quiz ... 179
Ćwiczenie ... 180

Część II ... 181

Rozdział 8. Użycie zmiennych ... 183
O zmiennych ... 183
Czym są zmienne? ... 183
Jakie są zalety stosowania zmiennych? ... 184

Tworzenie i używanie zmiennych ... 186
Użycie zmiennych prostych ... 186
Użycie zmiennych złożonych ... 189
Tworzenie zmiennych na podstawie wyrażeń ... 191
Użycie zmiennych zastępujących wyrażenia ... 191
Użycie zmiennych do danych spoza kontekstu ... 194
Tworzenie zmiennych za pomocą elementów XSLT ... 198
Podsumowanie ... 199
Pytania i odpowiedzi ... 199
Zadania ... 200
Quiz ... 200
Ćwiczenie ... 200

Rozdział 9. Użycie parametrów ... 201
Parametry ... 201
Czym są parametry? ... 202
Jakie zalety ma stosowanie parametrów? ... 203
Użycie parametrów ... 203
Użycie parametrów do zmiany postaci wyniku ... 204
Użycie parametrów do tworzenia reguł — funkcji ... 207
Pobieranie danych spoza arkusza ... 210
Podsumowanie ... 216
Pytania i odpowiedzi ... 216
Zadania ... 217
Quiz ... 217
Ćwiczenie ... 217

Rozdział 10. Typy danych ... 219
Typy danych — podstawy ... 219
Co to jest typ danych? ... 220
Typy danych w XSLT ... 220
Konwersje typów danych ... 227
Jawne konwersje typów danych ... 228
Niejawne konwersje typów danych ... 230
Pułapki konwersji ... 230
Porównywanie wartości ... 231
Pułapki związane z porównywaniem ... 231
Podsumowanie ... 232
Pytania i odpowiedzi ... 233
Zadania ... 233
Quiz ... 234
Ćwiczenie ... 234

Rozdział 11. Użycie łańcuchów ... 235
Operacje na łańcuchach ... 235
Sklejanie łańcuchów ... 235
Odszukiwanie znaków w łańcuchu ... 238
Określanie długości łańcucha ... 242
Użycie części łańcuchów ... 243
Zastępowanie części łańcucha ... 247
Formatowanie danych ... 248
Formatowanie liczb ... 248
Formatowanie dat i czasu ... 252
Formatowanie innych danych ... 254

Podsumowanie ... 254
Pytania i odpowiedzi ... 254
Zadania ... 255
Quiz ... 255
Ćwiczenia ... 255

Rozdział 12. Sortowanie i numerowanie ... 257
Sortowanie ... 257
Sortowanie statyczne ... 258
Sortowanie dynamiczne ... 265
Numerowanie ... 266
Wstawianie numerowania ... 267
Kontrola postaci numerowania ... 273
Podsumowanie ... 275
Pytania i odpowiedzi ... 276
Zadania ... 276
Quiz ... 277
Ćwiczenia ... 277

Rozdział 13. Wieloplikowe arkusze stylów ... 279
Użycie wieloplikowych arkuszy stylów ... 279
Zalety wieloplikowych arkuszy stylów ... 280
Wady wieloplikowych arkuszy stylów ... 280
Włączanie arkuszy stylów ... 281
Zduplikowane reguły ... 286
Zduplikowane zmienne i parametry ... 287
Importowanie arkuszy stylów ... 287
Różnice między włączaniem a importowaniem ... 287
Jak zaimportować arkusz stylów ... 288
Nadpisywanie reguł ... 289
Zasady importu innych składników arkusza ... 292
Podsumowanie ... 295
Pytania i odpowiedzi ... 296
Zadania ... 297
Quiz ... 297
Ćwiczenie ... 297

Rozdział 14. Użycie wielu źródłowych plików XML ... 299
Korzystanie z innych źródeł XML ... 299
Pobieranie danych ze źródła XML ... 300
Dynamiczne definiowanie dodatkowych dokumentów ... 309
Wskazywanie dokumentów źródłowych ... 311
Wskazywanie pliku znajdującego się na innym serwerze ... 312
Dostęp do elementów arkusza stylów ... 313
Wiele dokumentów: za i przeciw ... 314
Pułapki związane z używaniem wielu dokumentów ... 314
Wskazówki dotyczące używania wielu dokumentów ... 315
Podsumowanie ... 316
Pytania i odpowiedzi ... 316
Zadania ... 317
Quiz ... 317
Ćwiczenia ... 317

Część III 319

Rozdział 15. Przestrzenie nazw 321
Co to są przestrzenie nazw 321
Wszystko o przestrzeniach nazw 322
Zalety przestrzeni nazw 325
Wady przestrzeni nazw 326
Przestrzenie nazw, DTD i schematy XML 327
Przetwarzanie XML zawierającego przestrzenie nazw 328
Pobieranie informacji o przestrzeniach nazw 330
Wstawianie i usuwanie przestrzeni nazw 331
Wstawianie węzłów z przestrzenią nazw 331
Zmiana przestrzeni nazw 333
Usuwanie przestrzeni nazw 335
Podsumowanie 338
Pytania i odpowiedzi 339
Zadania 339
Quiz 339
Ćwiczenie 340

Rozdział 16. Zaawansowane metody wybierania danych 341
Jeszcze o wyrażeniach 341
Dopasowywanie a wybieranie danych 342
Porównywanie wartości 343
Pobieranie wartości bez powtórzeń 343
Użycie kluczy 345
Co to jest klucz? 345
Wybieranie danych za pomocą kluczy 346
Użycie niepowtarzalnych identyfikatorów ID 352
Wybieranie danych według identyfikatora ID 354
Wstawianie niepowtarzalnych wartości ID 354
Użycie kluczy i generowanych identyfikatorów do wybierania niepowtarzających się wartości 357
Podsumowanie 358
Pytania i odpowiedzi 359
Zadania 359
Quiz 359
Ćwiczenie 360

Rozdział 17. Rekurencja 361
Rekurencja 361
Co to jest rekurencja? 361
Po co i kiedy używa się rekurencji? 362
Wady rekurencji 364
Tworzenie reguł rekurencyjnych 365
Rekurencja z pojedynczymi wartościami 368
Sumowanie za pomocą rekurencji 369
Podsumowanie 372
Pytania i odpowiedzi 373
Zadania 373
Quiz 373
Ćwiczenia 374

Rozdział 18. Obliczeniowe arkusze stylów ... 375
Wszystko o obliczeniowych arkuszach stylów ... 375
Co to jest obliczeniowy arkusz stylów? ... 375
Kiedy używa się arkuszy obliczeniowych? ... 376
Operatory i funkcje używane w obliczeniach ... 377
Operatory ... 377
Funkcje ... 379
Aplikacje obliczeniowe ... 383
Ranking drużyn ... 383
Przetwarzanie łańcuchów ... 388
Przeliczanie walut ... 390
Podsumowanie ... 392
Pytania i odpowiedzi ... 392
Zadania ... 393
Quiz ... 393
Ćwiczenie ... 393

Rozdział 19. Korzystanie z rozszerzeń XSLT ... 395
Wszystko o rozszerzeniach XSLT ... 395
Czym są rozszerzenia? ... 395
Zalety rozszerzeń ... 396
Wady rozszerzeń ... 396
Jak działają rozszerzenia XSLT? ... 397
Użycie wbudowanych rozszerzeń ... 398
Użycie elementów rozszerzających ... 398
Użycie funkcji rozszerzających ... 401
Użycie rozszerzeń w innych procesorach ... 403
Tworzenie własnych funkcji rozszerzających ... 403
Użycie funkcji języka Java jako funkcji rozszerzających ... 403
Tworzenie skryptowych funkcji rozszerzających ... 406
Podsumowanie ... 409
Pytania i odpowiedzi ... 410
Zadania ... 410
Quiz ... 410
Ćwiczenia ... 411

Rozdział 20. Użycie różnych procesorów ... 413
Przenośne arkusze XSLT ... 413
Najważniejsze różnice między procesorami ... 414
Radzenie sobie z różnicami między procesorami ... 417
Radzenie sobie z różnymi wersjami XSLT ... 421
Możliwości serwerów baz danych w zakresie XML ... 423
Pobieranie i przekształcanie danych XML w Oracle ... 424
Pobieranie danych XML z SQL Servera Microsoftu ... 427
Podsumowanie ... 428
Pytania i odpowiedzi ... 429
Zadania ... 429
Quiz ... 430
Ćwiczenie ... 430

Rozdział 21. Projektowanie aplikacji XML i XSLT ... 431
Projektowanie XML ... 431
Uwagi o projektowaniu XML ... 433
Narzędzia projektowe ... 440

Projektowanie XSLT ... 441
Uwagi o projektowaniu XSLT ... 441
Zalecenia dla programistów XSLT ... 445
Podsumowanie ... 446
Pytania i odpowiedzi ... 447
Zadania ... 447
Quiz ... 447
Ćwiczenie ... 448

Dodatki ... 449

Dodatek A Odpowiedzi do quizu i rozwiązania ćwiczeń ... 451
Odpowiedzi do rozdziału 1. ... 451
Odpowiedzi do quizu ... 451
Rozwiązanie ćwiczenia ... 451
Odpowiedzi do rozdziału 2. ... 452
Odpowiedzi do quizu ... 452
Rozwiązanie ćwiczenia ... 452
Odpowiedzi do rozdziału 3. ... 452
Odpowiedzi do quizu ... 452
Rozwiązanie ćwiczenia ... 453
Odpowiedzi do rozdziału 4. ... 453
Odpowiedzi do quizu ... 453
Rozwiązania ćwiczeń ... 454
Odpowiedzi do rozdziału 5. ... 455
Odpowiedzi do quizu ... 455
Rozwiązanie ćwiczenia ... 456
Odpowiedzi do rozdziału 6. ... 456
Odpowiedzi do quizu ... 456
Rozwiązania ćwiczeń ... 457
Odpowiedzi do rozdziału 7. ... 458
Odpowiedzi do quizu ... 458
Rozwiązanie ćwiczenia ... 458
Odpowiedzi do rozdziału 8. ... 459
Odpowiedzi do quizu ... 459
Rozwiązanie ćwiczenia ... 459
Odpowiedzi do rozdziału 9. ... 460
Odpowiedzi do quizu ... 460
Rozwiązanie ćwiczenia ... 461
Odpowiedzi do rozdziału 10. ... 462
Odpowiedzi do quizu ... 462
Rozwiązanie ćwiczenia ... 462
Odpowiedzi do rozdziału 11. ... 463
Odpowiedzi do quizu ... 463
Rozwiązania ćwiczeń ... 463
Odpowiedzi do rozdziału 12. ... 465
Odpowiedzi do quizu ... 465
Rozwiązania ćwiczeń ... 466
Odpowiedzi do rozdziału 13. ... 467
Odpowiedzi do quizu ... 467
Rozwiązanie ćwiczenia ... 467
Odpowiedzi do rozdziału 14. ... 468
Odpowiedzi do quizu ... 468
Rozwiązania ćwiczeń ... 468

Odpowiedzi do rozdziału 15. ... 469
Odpowiedzi do quizu ... 469
Rozwiązanie ćwiczenia ... 470
Odpowiedzi do rozdziału 16. ... 471
Odpowiedzi do quizu ... 471
Rozwiązanie ćwiczenia ... 471
Odpowiedzi do rozdziału 17. ... 472
Odpowiedzi do quizu ... 472
Rozwiązania ćwiczeń ... 472
Odpowiedzi do rozdziału 18. ... 474
Odpowiedzi do quizu ... 474
Rozwiązanie ćwiczenia ... 474
Odpowiedzi do rozdziału 19. ... 476
Odpowiedzi do quizu ... 476
Rozwiązania ćwiczeń ... 476
Odpowiedzi do rozdziału 20. ... 477
Odpowiedzi do quizu ... 477
Rozwiązanie ćwiczenia ... 477
Odpowiedzi do rozdziału 21. ... 478
Odpowiedzi do quizu ... 478
Rozwiązania ćwiczeń ... 479

Dodatek B Zestawienie elementów i funkcji XSLT ... 481

Elementy XSLT ... 482
xsl:apply-imports ... 482
xsl:apply-templates ... 482
xsl:attribute ... 483
xsl:attribute-set ... 483
xsl:call-template ... 484
xsl:choose ... 484
xsl:comment ... 485
xsl:copy ... 485
xsl:copy-of ... 486
xsl:decimal-format ... 486
xsl:element ... 488
xsl:fallback ... 488
xsl:for-each ... 489
xsl:if ... 489
xsl:import ... 490
xsl:include ... 490
xsl:key ... 491
xsl:message ... 491
xsl:namespace-alias ... 492
xsl:number ... 493
xsl:otherwise ... 494
xsl:output ... 494
xsl:param ... 495
xsl:preserve-space ... 496
xsl:processing-instruction ... 496
xsl:sort ... 497
xsl:strip-space ... 498
xsl:stylesheet ... 498
xsl:template ... 499
xsl:text ... 500

xsl:transform ... 500
xsl:value-of ... 500
xsl:variable ... 501
xsl:when ... 502
xsl:with-param ... 502
Funkcje XSLT i XPath ... 503
boolean() ... 503
ceiling() ... 504
concat() ... 504
contains() ... 504
count() ... 505
current() ... 505
document() ... 505
element-available() ... 506
false() ... 506
floor() ... 507
format-number() ... 507
function-available() ... 508
generate-id() ... 508
id() ... 509
key() ... 509
lang() ... 509
last() ... 510
local-name() ... 510
name() ... 511
namespace-uri() ... 511
normalize-space() ... 511
not() ... 512
number() ... 512
position() ... 513
round() ... 513
starts-with() ... 514
string() ... 514
string-length() ... 515
substring() ... 515
substring-after() ... 515
substring-before() ... 516
sum() ... 516
system-property() ... 517
translate() ... 517
true() ... 518
unparsed-entity-uri() ... 518

Dodatek C Wywoływanie wybranych procesorów z wiersza poleceń ... 519
MSXSL ... 519
Użycie ... 520
Opcje ... 520
Saxon ... 520
Użycie ... 521
Opcje ... 521
Xalan ... 521
Użycie ... 521
Opcje ... 521

Dodatek D XML w Sieci ... 525

Dodatek E Przykładowe projekty ... 527
Tworzenie artykułu ze spisem treści ... 527
Założenia projektowe ... 527
Artykuł oznakowany jako XML ... 528
Dokument XSLT ... 530
Tworzenie wieloplikowego arkusza stylów z parametrami ... 537
Początek projektu ... 538
Tworzenie drugiego arkusza stylów ... 542
Arkusz stylów do wszystkich danych ... 544
Koszyk na zakupy w XSLT ... 545
Katalog towarów ... 546
Koszyk na zakupy ... 547
Wyświetlanie danych ... 548
Aktualizacja koszyka ... 552
Wywołanie procesora ... 555

Skorowidz ... 563

Rozdział 3.

Wybieranie danych

W poprzednim rozdziale dowiedzieliśmy się, czym jest arkusz stylów i jak się go używa. Wiemy też nieco o regułach i pobieraniu wartości z dokumentów XML. Używane dotąd wyrażenia doboru reguł i wybierania danych były bardzo proste, więc nasze możliwości były ograniczone.

W tym rozdziale skoncentrujemy się na precyzyjniejszym wybieraniu danych. Dowiemy się:

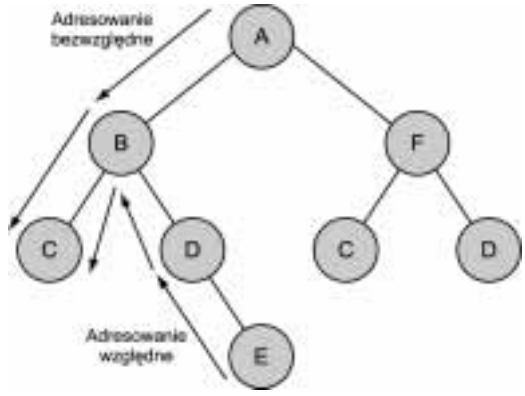
- Jak działają drzewa dokumentów XML.
- Czym jest XPath.
- Jak wybierać pojedyncze elementy.
- Jak wybierać wiele elementów.
- Jak wybierać atrybuty.

Drzewa dokumentów XML

Dokument XML to zbiór elementów posiadających pewną hierarchię. Każdy element w dokumencie może mieć dowolną liczbę „dzieci", począwszy od zera, te „dzieci" też mogą mieć „dzieci" i tak dalej. Poza tym każdy element może mieć dowolną liczbę atrybutów. Te stwierdzenia nie są zbyt odkrywcze, ale to one określają strukturę dokumentu XML. Każdy element i każdy atrybut mają jednoznacznie określone miejsce w drzewie dokumentu. Jako że każdy element i każdy atrybut można jednoznacznie zidentyfikować, można wskazać pojedynczy element lub atrybut i pobrać jego wartość. Zobrazowano to wszystko na rysunku 3.1.



Każdy element drzewa na rysunku 3.1 zaznaczono kołem. Poszczególne „dzieci" jednego elementu można rozróżnić dzięki oznaczającym je literom. Jednak, jak widać, czasem „dzieci" oznaczone są takimi samymi literami, więc nie można powiedzieć „daj element C", gdyż może chodzić o element C będący „dzieckiem" elementu B lub



Rysunek 3.1. *Graficzna postać drzewa*

elementu F. Nie znaczy to jednak, że w ogóle nie jesteśmy w stanie rozróżnić elementów, po prostu trzeba wyrażać się bardziej precyzyjnie. Aby pobrać pewien element, możemy powiedzieć: „daj element C będący 'dzieckiem' elementu B będącego 'dzieckiem' elementu głównego A". Takie wskazywanie elementów nazywamy *adresowaniem bezwzględnym*. Adresowanie takie nazywamy bezwzględnym dlatego, że podajemy bezwzględne położenie elementu w drzewie. W przypadku takiego adresowania zawsze jednoznacznie wskazujemy wybrane miejsce.



Każdy element drzewa na rysunku 3.1 zaznaczono kołem. Poszczególne „dzieci" jednego elementu można rozróżnić dzięki oznaczającym je literom. Jednak, jak widać, czasem „dzieci" oznaczone są takimi samymi literami, więc nie można powiedzieć „daj element C", gdyż może chodzić o element C będący „dzieckiem" elementu B lub elementu F. Nie znaczy to jednak, że w ogóle nie jesteśmy w stanie rozróżnić elementów, po prostu trzeba wyrażać się bardziej precyzyjnie. Aby pobrać pewien element, możemy powiedzieć: „daj element C będący 'dzieckiem' elementu B będącego 'dzieckiem' elementu głównego A". Takie wskazywanie elementów nazywamy *adresowaniem bezwzględnym*. Adresowanie takie nazywamy bezwzględnym dlatego, że podajemy bezwzględne położenie elementu w drzewie. W przypadku takiego adresowania zawsze jednoznacznie wskazujemy wybrane miejsce.



Innym sposobem adresowania jest adresowanie względem wybranego elementu. Załóżmy, że element E z rysunku 3.1 jest punktem startowym. Chcemy wskazać ten sam element co poprzednio, więc mówimy: „daj mi element C, sąsiada mojego 'rodzica'". Tego typu adresowanie nazywamy *adresowaniem względnym*, gdyż położenie elementu w drzewie podajemy względem aktualnej pozycji.

W przypadku adresowania względnego nie wskazuje się w drzewie dokumentu niepowtarzalnej lokalizacji, gdyż to, który element zostanie wybrany, zależy od punktu startowego zapytania.

Co to jest węzeł?

Mówimy konsekwentnie o elementach i atrybutach, ale różnice między jednymi a drugimi nie są tak duże, jak mogłoby się wydawać. Najważniejszą różnicą jest to, że elementy

mogą mieć elementy „dzieci", zaś atrybuty nie. Wobec tego atrybut ma zawsze wartość prostą (tekstową), podczas gdy element może mieć w wartości także inne elementy i atrybuty.

Skoro elementy i atrybuty zanadto się nie różnią, można je w drzewie dokumentu zapisać jako takie same obiekty. Element E z rysunku 3.1 może być na przykład równie dobrze atrybutem, gdyż nie ma żadnych „dzieci". Niektórzy uważają nawet, że nie powinno się stosować atrybutów, gdyż są to przypadki szczególne elementów. Elementy i atrybuty są wzajemnie zamienne, o ile tylko element nie ma „dzieci" ani atrybutów. Jako że atrybuty są po prostu nazwami z wartością, element musi zawierać atrybuty różniące się nazwami. Wartość elementu z kolei może zawierać wiele innych elementów o takich samych nazwach. To rozróżnienie jest bardzo istotne przy projektowaniu dokumentów XML, szczególnie kiedy przewiduje się ich modyfikację w przyszłości.



W Obiektowym Modelu Dokumentu (DOM) oraz w XSLT, a właściwie w XPath, elementy i atrybuty różnią się od siebie tak nieznacznie, że są traktowane bardzo podobnie. Niektóre funkcje DOM Level 2 na elementach działają równie dobrze, jak na atrybutach. Funkcje `nodeName` i `nodeValue` nie odróżniają elementów od atrybutów, choć ich wyniki mogą być różne w obu przypadkach. Elementy i atrybuty noszą wspólną nazwę *węzłów*, czyli prostych obiektów zawierających dane w drzewie dokumentu.

Węzeł bieżący

W rozdziale 2. użyliśmy pojęcia bieżącego elementu. Wprawdzie sama nazwa zdaje się już sporo wyjaśniać, jednak przyda się jeszcze krótki opis. Poza tym, ze względu na podobieństwo elementów i atrybutów, od teraz będziemy mówić o *bieżącym węźle*.



W rozdziale 2. pokazywaliśmy arkusz stylów XSLT przetwarzający dokument XML, w którym elementy XML były dopasowywane do reguł z arkusza. Nie wiemy jeszcze jednak, jak się tworzy wyrażenia atrybutu `match`. Zawsze, kiedy pojawia się dopasowanie i wykonywana jest reguła, węzeł, na rzecz którego ta reguła została uruchomiona, staje się *węzłem bieżącym*, czyli wskazaniem pewnego węzła w drzewie XML.



W przypadku z debuggerem XSLT, który umożliwia krokowe wykonywanie przekształcenia, węzeł bieżący jest zawsze pokazywany. Debugger śledzi bieżący węzeł i regułę, która jest uruchamiana.

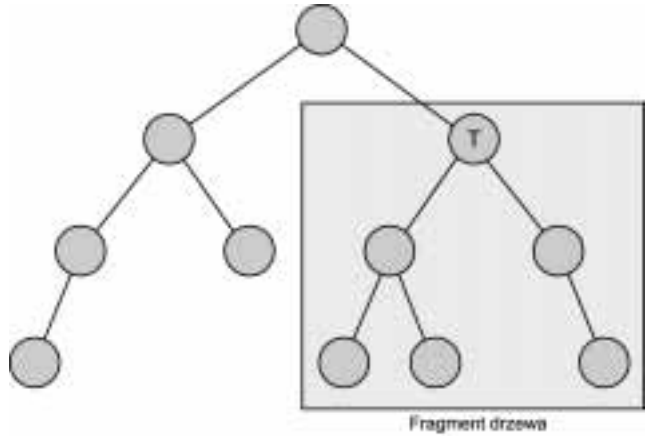


Bieżący węzeł to po prostu wskaźnik do przetwarzanego węzła, ale reguła nie musi ograniczać się do wykorzystywania wartości tego właśnie węzła. W regule można użyć adresowania bezwzględnego lub względnego do sięgnięcia do dowolnego innego węzła z dokumentu XML. Jak wspomniano wcześniej, nie trzeba też korzystać akurat z pojedynczej wartości. Jeśli element ma atrybuty i elementy potomne, one też są częścią jego wartości. Taka wartość nazywana jest *fragmentem drzewa* — jest to część drzewa dokumentu XML zaczynająca się od wskazanego węzła. Sam fragment drzewa jest poprawnym dokumentem XML.

Użycie fragmentów drzewa pokazywaliśmy już wcześniej, w rozdziale 2., w którym mówiliśmy, że funkcja `text()` wybiera jedynie wartość tekstową elementu. Jeśli odwołamy

się po prostu do wartości elementu, otrzymamy tekst elementu i wszystkich jego potomków — zatem otrzymamy wartość tekstową fragmentu drzewa. Aby lepiej sobie to wszystko uzmysłowić, spójrzmy na rysunek 3.2.

Rysunek 3.2.
Fragment drzewa węzła T



Na rysunku graficznie przedstawiono fragment drzewa, w tym wypadku należący do węzła T. Odpowiada on wartości węzła T.

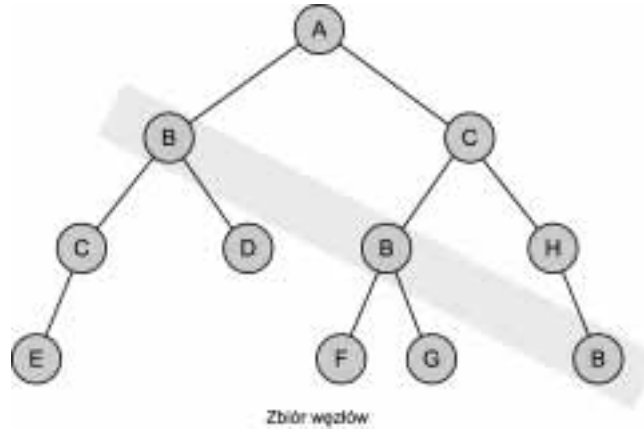
Czym jest zbiór węzłów?

Teraz, kiedy wiemy już, czym jest węzeł, wydaje się, że pojęcie zbiorów węzłów jest oczywiste — no cóż, i tak, i nie.



Kiedy wybieramy dane według pewnego wyrażenia, wyrażenie to nie musi pasować do dokładnie jednego węzła, ale może pasować do ich większej liczby. Takie właśnie dopasowane do wyrażenia węzły nazywamy *zbiorem węzłów*. Najbardziej typowym zbiorem węzłów są węzły „dzieci" wskazanego elementu. Niektórzy uważają, że jest to jedyny rodzaj zbioru węzłów, ale tak nie jest. Bez trudu można stworzyć wyrażenie, które wybierze węzły z różnych części dokumentu XML; przykład pokazano na rysunku 3.3.

Rysunek 3.3.
Zbiór węzłów zawierający węzły porozrzucane po całym dokumencie



Na rysunku 3.3 pokazano zbiór węzłów, jaki otrzymamy żądając „wszystkich węzłów o nazwie B". Jak widać, w takim wypadku położenie tych węzłów w dokumencie nie ma znaczenia. Każdy węzeł pasujący do zadanego wyrażenia jest wstawiany do zbioru węzłów. Zbiór węzłów z rysunku 3.3 składa się z kilku węzłów; z tych węzłów mamy dostęp do fragmentów drzewa składających się z danego węzła i jego potomków. Jeśli wyrażenie obejmowało jedynie atrybuty, zbiór węzłów składa się z węzłów o wartościach prostych.

Zbiory węzłów są w XSLT bardzo istotne. Umożliwiają tworzenie spisów treści, indeksów i wszelkiego rodzaju dokumentów, których dane są porozrzucane po dokumencie XML. Dzięki temu możliwe jest uzyskiwanie różnych wyników służących do różnych celów na podstawie jednego dokumentu źródłowego XML.

XPath

Dotąd rozdział ten był ściśle teoretyczny. Teoria ta jest potrzebna, gdyż jest ona podstawą zastosowań praktycznych, którym teraz poświęcimy nasz wykład.

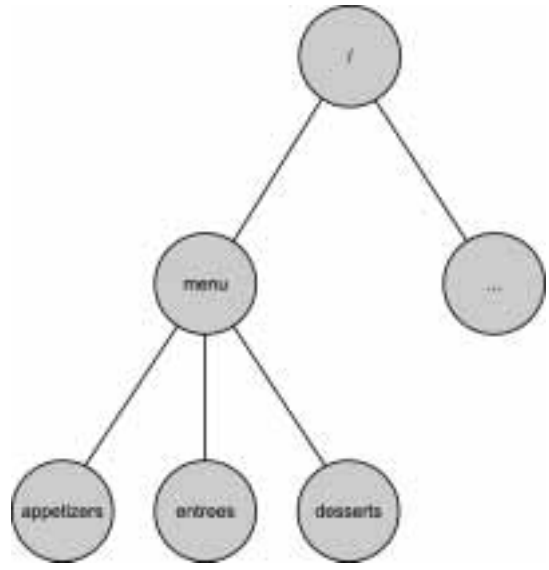
XSLT nie mógłby działać, jeśli nie byłoby mechanizmu dopasowywania i wybierania węzłów i potem ich przetwarzania. Trzeba móc wskazać, który węzeł lub węzły mają być dobrane. Do tego właśnie służą wyrażenia XPath. XPath to język używany do wskazywania, którego węzła lub węzłów zamierzamy użyć. Wyrażenia XPath mogą być bardzo proste, mogą wskazywać pewne miejsce w drzewie dokumentu za pomocą adresu bezwzględnego. Mogą też być złożone. Będziemy pokazywać tworzenie coraz bardziej złożonych wyrażeń, ale zaczniemy oczywiście od podstaw.

Wybieranie elementów

Osoby zaznajomione ze sposobem adresowania stosowanym w Sieci nie będą miały problemu ze zrozumieniem podstawowych wyrażeń XPath. Witryna sieciowa jest hierarchią plików, tak jak dokument XML jest hierarchią elementów. Odwiedzając jakąś witrynę podaje się adres jej strony głównej, na przykład *http://www.jakas.witryna.com*. Takie samo znaczenie ma adres *http://www.jakas.witryna.com/*, który jest bardziej poprawny formalnie. Dalej w adresie podaje się, które miejsce całej hierarchii nas interesuje, na przykład adres *www.jakas.witryna.com/menu/entrees* może wskazywać katalog *entrees* będący podkatalogiem katalogu *menu* znajdującego się w katalogu głównym. */menu/entrees* to szczególnie interesująca nas część — jednoznacznie identyfikuje położenie w ramach hierarchii witryny, jak to pokazano na rysunku 3.4.

Rysunek 3.4 pokazuje część hierarchii witryny. Warto zauważyć, że */menu/entrees* jednoznacznie identyfikuje węzeł `entrees` znajdujący się w drzewie. Aby wybrać węzeł `desserts`, należałoby użyć wyrażenia */menu/desserts*. Spójrz na wydruk 3.1.

Rysunek 3.4.
Hierarchia witryny sieciowej



Wydruk 3.1. *Menu odpowiadające rysunkowi 3.4 zapisane jako XML*

```
 1: <?xml version="1.0" encoding="UTF-8"?>
 2: <menu>
 3:   <appetizers title="Na rozbudzenie apetytu">
 4:     <dish id="1" price="8.95">Ciasteczka krabowe</dish>
 5:     <dish id="2" price="9.95">Krewetki tygrysie</dish>
 6:     <dish id="3" price="10.95">Wędzony łosoś i quesadilla z awokado</dish>
 7:     <dish id="4" price="6.95">Sałatka Cezara</dish>
 8:   </appetizers>
 9:   <entrees title="Czas coś zjeść">
10:     <dish id="5" price="19.95">Łosoś> z grilla</dish>
11:     <dish id="6" price="17.95">Frutti di mare</dish>
12:     <dish id="7" price="16.95">Linguini al Pesto</dish>
13:     <dish id="8" price="18.95">Bitki jagnięce</dish>
14:     <dish id="9" price="16.95">Żeberka i skrzydełka</dish>
15:   </entrees>
16:   <desserts title="Na koniec">
17:     <dish id="10" price="6.95">Biała dama</dish>
18:     <dish id="11" price="5.95">Mus czekoladowy</dish>
19:     <dish id="12" price="6.95">Lody bananowe</dish>
20:   </desserts>
21: </menu>
```



Wydruki do tego rozdziału można pobrać z witryny wydawcy.

analiza Kod XML z wydruku 3.1 ma taką samą strukturę drzewiastą, jak witryna przedstawiona na rysunku 3.4. Tak jak w witrynie, `/menu/entrees` wskazuje element `entrees`. Jak widać, wskazywanie za pomocą XPath pewnego węzła w dokumencie XML jest bardzo proste. Zasady są zapewne już wszystkim znane, choć nie z XPath. Spójrzmy na wydruk 3.2.

Wydruk 3.2. *Arkusz stylów, w którym wybiera się węzeł entrees z wydruku 3.1*

```
1: <?xml version="1.0" encoding="UTF-8"?>
2: <xsl:stylesheet version="1.0"
3:   xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
4:
5:   <xsl:template match="/">
6:     <xsl:value-of select="/menu/entrees" />
7:   </xsl:template>
8:
9: </xsl:stylesheet>
```

analiza Reguła z wiersza 5. pasuje do elementu głównego dokumentu źródłowego. Wartość pobrana w wierszu 6. jest wybierana za pomocą wyrażenia `/menu/entrees`, które pasuje do elementu `entrees` będącego „dzieckiem" elementu głównego `menu`. Wynik zastosowania wydruku 3.2 do wydruku 3.1 pokazano na wydruku 3.3.

źródło/wynik **Wydruk 3.3.** *Wynik zastosowania arkusza 3.2 do wydruku 3.1*

```
<?xml version="1.0" encoding="utf-8"?>
    Łosoś z grilla
    Frutti di mare
    Linguini al Pesto
    Bitki jagnięce
    Żeberka i skrzydełka
```



Taki wynik uzyskano za pomocą procesora Saxon. Jeśli użyty zostanie inny procesor, nieco inny może być wynik; MSXML generuje domyślnie wyniki jako UTF-16, więc w jego przypadku dokument wynikowy będzie miał między kolejnymi literami spacje.

analiza Wydruk 3.3 prezentuje wartość wszystkich „dzieci" węzła `entrees`. Zgodnie z tym, co mówiliśmy w poprzednim rozdziale — w wierszu 6. z wydruku 3.2 żądamy podania wartości węzła `entrees`. Wartość tego węzła zawiera wszystkich jego potomków, więc pobranie jej oznacza pobranie ich wartości tekstowej. Takie działanie może być nieco mylące, gdyż wydaje się, że na wydruku 3.2 wybierany jest zbiór węzłów składający się z wszystkich elementów „dzieci" węzła `entrees`. Jeśli węzeł ten zawierałby też wartość tekstową, okazałoby się, że nie jest to prawda. Można jednak stworzyć dodatkową regułę obsługującą elementy `dish`, jak to pokazano na wydruku 3.4.

Wydruk 3.4. *Arkusz stylów z dokładniejszą kontrolą elementów dish*

```
 1: <?xml version="1.0" encoding="UTF-8"?>
 2: <xsl:stylesheet version="1.0"
 3:   xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
 4:
 5:   <xsl:template match="/">
 6:     <xsl:apply-templates />
 7:   </xsl:template>
 8:
 9:   <xsl:template match="/menu/appetizers" />
10:
11:   <xsl:template match="/menu/entrees">
```

```
12:      Przystawki:
13:      <xsl:apply-templates />
14:   </xsl:template>
15:
16:   <xsl:template match="/menu/desserts" />
17:
18:   <xsl:template match="dish">
19:     -<xsl:value-of select="text()" />
20:   </xsl:template>
21:
22: </xsl:stylesheet>
```

analiza Warto zauważyć, że wiersze 9. i 16. powyższego wydruku powodują pominięcie węzłów `appetizers` i `desserts`. Wynik zastosowania arkusza 3.4 do wydruku 3.1 pokazano na wydruku 3.5.

źródło/wynik **Wydruk 3.5.** *Wynik zastosowania arkusza 3.4 do wydruku 3.1*

```
<?xml version="1.0" encoding="utf-8"?>


     Przystawki:


    -Łosoś z grilla

    -Frutti di mare

    -Linguini al Pesto

    -Bitki jagnięce

    -Żeberka i skrzydełka
```

analiza Na wydruku 3.5 każdy węzeł `dish` jest już obsługiwany osobno. Kreski (-) przed poszczególnymi potrawami pokazują, że tak jest naprawdę. Białe znaki pojawiają się w związku z domyślnymi regułami obsługi białych znaków w procesorze.

Pobieranie wartości pojedynczego elementu

Powyższy kod jest niewygodny, z tego względu, że dotyczy elementu lub zbioru elementów. W zbiorze elementów (np. elementy `dish`) żaden węzeł nie jest wyróżniony. Jeśli wskażemy węzeł `dish` zamiast dobrać go w regule, otrzymamy wartość jednego takiego węzła — uwidoczniono to na wydruku 3.6.

Wydruk 3.6. *Arkusz stylów pobierający wartość elementu dish*

```
1: <?xml version="1.0" encoding="UTF-8"?>
2: <xsl:stylesheet version="1.0"
3:   xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
4:
5:   <xsl:template match="/">
6:      <xsl:apply-templates />
```

```
 7:   </xsl:template>
 8:
 9:   <xsl:template match="/menu/appetizers" />
10:
11:   <xsl:template match="/menu/entrees">
12:      <xsl:value-of select="dish" />
13:   </xsl:template>
14:
15:   <xsl:template match="/menu/desserts" />
16:
17: </xsl:stylesheet>
```

analiza Reguła z wiersza 11. pasuje do elementu `entrees` i wybiera jedynie wartość węzła `dish` — wiersz 12. Różnica między tym wydrukiem a wydrukiem 3.4 polega na tym, że nie dopasowujemy reguły do elementu `dish`. Wynik działania arkusza 3.6 pokazano na wydruku 3.7.

źródło/wynik **Wydruk 3.7.** *Wynik zastosowania arkusza 3.6 do wydruku 3.1*

```
<?xml version="1.0" encoding="utf-8"?>

  Łosoś z grilla
```

analiza Na wydruku 3.7 widać wyraźnie, że przyjęte założenie dotyczące wydruku 3.6 jest słuszne. Pobieranie wartości węzła `dish` daje wartość dokładnie jednego takiego węzła. Co stało się z pozostałymi węzłami? Przecież `xsl:value-of` pasuje do węzłów `dish`, czyli do zbioru węzłów. To, że pobieramy pojedynczą wartość, jest znów wynikiem domyślnego zachowania XSLT. Jeśli do wyboru z elementu `xsl:value-of` pasuje zbiór węzłów, używana jest jedynie pierwsza wartość z tego zbioru. Pierwszym elementem jest ten element, który znajduje się najwcześniej w źródle XML; w tym wypadku jest to łosoś z grilla.

Pojawia się kolejne pytanie: jak wybrać pojedynczy węzeł ze zbioru, ale nie pierwszy? Na szczęście można po prostu podać numer węzła, który ma być wybrany. Jednak taki sposób wybierania zasadniczo różni się od dobrze nam znanego z adresowania stron sposobu zapisu wyrażeń XPath. Numer żądanego węzła trzeba podać w nawiasie kwadratowym — oto przykład pobrania trzeciego elementu `dish`:

```
<xsl:value-of select="dish[3]"/>
```



W wielu językach programowania listy elementów są numerowane od zera, więc element o indeksie 3 będzie w nich czwartym elementem. W XSLT numeracja zaczyna się od jedynki, więc indeks 3 odpowiada trzeciemu elementowi.

nowy termin Wartość w nawiasie kwadratowym nazywamy *predykatem*. Predykat może być literałem lub całym wyrażeniem, decydującym czy dany węzeł ma zostać włączony do wyboru.

Warto zauważyć, że w przykładzie używamy adresowania względnego, przez co wybór jest zależny od aktualnej pozycji. Jeśli elementem bieżącym jest `entrees`, zostanie wybrany trzeci element `dish`. Jeśli węzeł bieżący nie ma „dzieci” `dish`, wartość pozostanie pusta.

Skoro predykaty mogą być wyrażeniami, mogą to być złożone warunki sprawdzające czy dany węzeł ma być wybrany. Jak zapowiedzieliśmy na początku rozdziału, bardziej złożone wyrażenia będziemy omawiać w dalszej części książki. Teraz mamy zapoznać się z podstawami. Przykład z wydruku 3.8 tworzy menu dnia na podstawie dokumentu z wydruku 3.1.

Wydruk 3.8. *Arkusz tworzący menu na dzisiaj*

```
 1: <?xml version="1.0" encoding="UTF-8"?>
 2: <xsl:stylesheet version="1.0"
 3:   xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
 4:
 5:   <xsl:template match="/">
 6:      Dzisiaj polecamy
 7:      -<xsl:value-of select="/menu/appetizers/dish[2]" />
 8:      -<xsl:value-of select="/menu/entrees/dish[3]" />
 9:      -<xsl:value-of select="/menu/desserts/dish[1]" />
10:   </xsl:template>
11:
12: </xsl:stylesheet>
```

analiza

Wydruk 3.8 zawiera tylko jedną regułą (wiersz 5.) pasującą do elementu głównego. Reguła ta wyświetla wartości różnych elementów z wydruku 3.1, korzystając z adresowania bezwzględnego i predykatu w formie liczby. W wierszu 7. wybieramy drugi element `dish` z elementu `appetizers`, w wierszu 8. trzeci element `dish` z elementu `entrees`, zaś w wierszu 9. pierwszy element `dish` z elementu `desserts`. Na wydruku 3.9. pokazano uzyskany wynik.

źródło/wynik

Wydruk 3.9. *Wynik zastosowania arkusza 3.8 do wydruku 3.1*

```
<?xml version="1.0" encoding="utf-8"?>
    Dzisiaj polecamy
    -Krewetki tygrysie
    -Linguini al Pesto
    -Biała dama
```

analiza

Wydruk 3.8 zawiera jedynie regułę pasującą do elementu głównego. W dalszej części arkusza stylów użyto adresowania bezwzględnego, aby wybrać potrzebne wartości. Jak widać, pobrana została lista dań tworząca menu na dzisiaj.

nowy termin

Wcześniej mówiliśmy o bieżącym węźle. Po uruchomieniu reguły jeden z węzłów staje się bieżącym. Wyrażenie XPath nie zawiera węzła bieżącego, ale zawiera węzeł kontekstu. *Węzeł kontekstu* jest częścią wyrażenia działającego na danym węźle.

Predykaty użyte w wydruku 3.8 działają na węźle kontekstu, w tym wypadku węźle `dish`. Wszystkie części wyrażenia XPath stają się po kolei węzłami kontekstu. Predykatów można używać na różnych etapach dobierania ścieżki — wtedy predykaty są stosowane do odpowiedniego węzła kontekstu.

Trzeba pamiętać, że jeśli nie zostaną znalezione żadne dopasowania, nic się nie stanie. Dzieje się tak również wtedy, kiedy predykat zawiera liczbę, której nie odpowiada

żaden element. W takim wypadku liczba ta jest po prostu pomijana. Nie pojawi się żaden komunikat mówiący, że danego elementu nie ma, ale po prostu nie zostaną wybrane żadne elementy, wobec czego reguła nie zostanie nigdy uruchomiona.

Wybieranie atrybutów

Dotąd omawialiśmy jedynie elementy, a co z atrybutami? Wspomniano, że elementy i atrybuty nie różnią się zbytnio, więc można by się spodziewać, że odwołania mogą mieć taką samą postać — próbę pokazano na wydruku 3.10.

Wydruk 3.10. *Arkusz stylów, w którym próbujemy pobrać atrybuty*

```
 1: <?xml version="1.0" encoding="UTF-8"?>
 2: <xsl:stylesheet version="1.0"
 3:   xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
 4:
 5:   <xsl:template match="/">
 6:      Deser dnia:
 7:      <xsl:value-of select="/menu/desserts/dish[2]" />
 8:      Cena: <xsl:value-of select="/menu/desserts/dish[2]/@price" />
 9:   </xsl:template>
10:
11: </xsl:stylesheet>
```

analiza W wierszu 8. wydruku 3.10 próbujemy pobrać wartość atrybutu `price` elementu `dish`. Jak widać na wydruku 3.11, takie rozwiązanie nie działa.

źródło/wynik **Wydruk 3.11.** *Wyniki zastosowania arkusza 3.10 do wydruku 3.1*

```
<?xml version="1.0" encoding="utf-8"?>
     Deser dnia:
     Mus czekoladowy
     Cena:
```

analiza Element `value-of` z wydruku 3.10 ewidentnie nie zadziałał: zamiast wyniku pojawia się puste miejsce, gdyż żaden element nie pasuje do podanego wyrażenia `select`. Tak właśnie powinno być, bo nie istnieje element `price`. Istnieje natomiast element `dish` i ma on atrybut `price`, który chcemy wybrać. Co zatem należy zmienić?

Trzeba po prostu nakazać procesorowi pobrać atrybut, a nie element. Aby wskazać, że chodzi nam o atrybut, przed nazwą dodajemy znak `@`. Wobec tego, skoro wiersz 8. wydruku 3.10 powinien wskazywać atrybut, wyrażenie `select` powinno mieć postać `/menu/desserts/dish[2]/@price`, co pokazano na wydruku 3.12. Wynik pokazany na wydruku 3.13 zawiera to, czego się spodziewaliśmy.

Wydruk 3.12. *Arkusz stylów wybierający atrybut*

```
1: <?xml version="1.0" encoding="UTF-8"?>
2: <xsl:stylesheet version="1.0"
3:   xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
4:
5:   <xsl:template match="/">
```

```
 6:      Deser dnia:
 7:      <xsl:value-of select="/menu/desserts/dish[2]" />
 8:      Cena: <xsl:value-of select="/menu/desserts/dish[2]/@price" />
 9:    </xsl:template>
10:
11: </xsl:stylesheet>
```

źródło/wynik **Wydruk 3.13.** *Wynik zastosowania arkusza 3.12 do wydruku 3.1*

```
<?xml version="1.0" encoding="utf-8"?>
     Deser dnia:
     Mus czekoladowy
     Cena: 5.95
```

analiza Arkusz z wydruku 3.12 dał już wynik, o jaki nam chodziło. Jako że atrybuty nie mają „dzieci", wiadomo z góry, że nie powstaną żadne efekty uboczne: wartość atrybutu zawsze jest tekstem.



Atrybuty mogą mieć typ, ale wszystkie te typy oparte są na wartościach tekstowych. Typy danych szczegółowo omówimy w rozdziale 10.

Skoro atrybuty nie powodują żadnych efektów ubocznych, praca z nimi jest znacznie prostsza. Poza tym wszystkie atrybuty elementu mają różne nazwy, nie ma problemu z dobieraniem jednocześnie wielu atrybutów (i z pobieraniem wartości pierwszego z nich). Jedynym sposobem uzyskania wielu atrybutów jest zastosowanie metaznaków.

Warto jeszcze zauważyć, że atrybuty zabierają w dokumencie mniej miejsca niż elementy. Element ma znacznik początkowy i końcowy, atrybut tych znaczników nie ma. W związku z tym, że wartością atrybutów zawsze jest tekst, parser lub procesor może je przetwarzać szybciej, gdyż sprawdzanie elementów potomnych nie jest konieczne, co zwykle oznacza zwiększenie wydajności.

Techniki bardziej zaawansowane

Jak dotąd omawialiśmy, o ile tylko było to możliwe, użycie pojedynczego węzła: staraliśmy się unikać wyborów dających w wyniku więcej niż jeden węzeł. Wprawdzie na początek to bardzo dobre rozwiązanie, ale znacząco ogranicza ono możliwości XSLT. Jeśli nie tworzylibyśmy wyrażeń złożonych, to omówiony dotąd zakres XPath byłby wystarczający.

Użycie metaznaków

Metaznaki są bardzo często używane w funkcjach i językach służących do wyszukiwania. XPath ma tylko jeden metaznak, gwiazdkę (*). Metaznak ten zastępuje całą nazwę elementu lub atrybutu, więc wyrażenie `a*` nie pasuje do elementów zaczynających się literą *a* i spowoduje błąd. Metaznaki są przydatne do przeprowadzania dalej idącej analizy dokumentu XML, kiedy nazwy pośrednich węzłów nie mają znaczenia. Na wydruku 3.14 pokazano użycie metaznaku.

Wydruk 3.14. *Arkusz stylów z metaznakiem*

```
1: <?xml version="1.0" encoding="UTF-8"?>
2: <xsl:stylesheet version="1.0"
3:   xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
4:
5:   <xsl:template match="/menu/*/dish">
6:      -<xsl:value-of select="text()" />
7:   </xsl:template>
8:
9: </xsl:stylesheet>
```

analiza W wierszu 5. wydruku 3.14 użyto metaznaku, wobec czego nie ma znaczenia, czy dobrany element `dish` jest „dzieckiem" elementu `appetizers`, `entrees` czy `desserts` — w wierszu 6. pokazywana jest wartość dobranego elementu `dish`. Wyniki pokazano na wydruku 3.15.

źródło/wynik **Wydruk 3.15.** *Wynik zastosowania arkusza 3.14 do pliku 3.1*

```
<?xml version="1.0" encoding="utf-8"?>


     -Ciasteczka krabowe

     -Krewetki tygrysie

     -Wędzony łosoś i quesadilla z awokado

     -Sałatka Cezara


-Łosoś z grilla

     -Frutti di mare

     -Linguini al Pesto

     -Bitki jagnięce

     -Żeberka i skrzydełka


     -Biała dama

     -Mus czekoladowy

     -Lody bananowe
```

analiza Jak widać na wydruku 3.15, wybrane zostały wszystkie węzły `dish`, nie tylko „dzieci" węzłów pewnego rodzaju.

Tej techniki można użyć w wielu różnych sytuacjach. Załóżmy, że tworzymy artykuł lub książkę w formie dokumentu XML. Za pomocą metaznaku można wybrać do spisu treści wszystkie nagłówki rozdziałów i podrozdziałów. Jeśli nie chcemy pobierać

wszystkich węzłów, użycie samego metaznaku sprawy nie załatwi. Jak jednak widzieliśmy wcześniej, do precyzowania ścieżek można użyć predykatów. W ten sposób można dokładniej kontrolować, do czego będą dopasowywane metaznaki. Prosty przykład pokazano na wydruku 3.16.

Wydruk 3.16. *Arkusz stylów z metaznakami i predykatami*

```
 1: <?xml version="1.0" encoding="UTF-8"?>
 2: <xsl:stylesheet version="1.0"
 3:   xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
 4:
 5:   <xsl:template match="/">
 6:     Deser dnia:
 7:     <xsl:value-of select="/menu/*[3]/dish[2]" />
 8:     Cena: <xsl:value-of select="/menu/*[3]/dish[2]/@*[2]" />
 9:   </xsl:template>
10:
11: </xsl:stylesheet>
```

Wydruk 3.16 da takie same wyniki, jak wydruk 3.12 (czyli wyniki z wydruku 3.13). Zamiast jednak adresować węzły według nazwy, w wierszu 8. wydruku 3.16 użyto kilku metaznaków przy predykatach położenia. Element `desserts` jest trzecim elementem „dzieckiem" elementu `menu` — wyrażenie `/menu/*[3]` nakazuje procesorowi pobranie trzeciego „dziecka" elementu `menu`, niezależnie od jego nazwy. Wyrażenie pobierze element `desserts`, zupełnie tak samo, jakbyśmy użyli nazw elementów. Atrybut jest też wybierany według metaznaku; w tym wypadku procesor ma pobrać drugi atrybut elementu `dish`, którym jest atrybut `price`.

Użycie drzewa dokumentu



Pokazane techniki są ściśle powiązane z drzewem dokumentu XML. XPath umożliwia znacznie więcej przez użycie *ścieżek lokalizacji*, które są wyrażeniami wybierającymi węzeł lub zbiór węzłów względem węzła kontekstu. Ścieżka lokalizacji składa się z szeregu części. Omawiane dotąd wyrażenia ścieżek są szczególnymi przypadkami ścieżek lokalizacji, zaczynającymi się zwykle od węzła głównego lub węzła bieżącego. Ścieżka lokalizacji może też pojawiać się w innych instancjach lub być częścią predykatów. Jedną z części ścieżki lokalizacji jest predykat, więc można stworzyć ścieżkę lokalizacji z predykatem, zawierającym ścieżkę lokalizacji z innym predykatem i tak dalej.



Kolejną, znaną już nam częścią ścieżki lokalizacji jest *test węzła* będący częścią ścieżki lokalizacji pasującej do danego węzła lub węzłów. Definicja się uprości, kiedy podamy przykład. Przyjrzyjmy się takiej ścieżce:

```
/menu/*[3]/dish[2]/@*[2]
```

Testami węzła są `menu`, `*`, `dish` i `@*[2]`.



W dotychczas prezentowanych przykładach pokazaliśmy niewiele ścieżek lokalizacji. Nie wszystkie zawierały osie. Osie właściwie w tych ścieżkach były, ale nie zostały podane jawnie. Spójrz na następującą ścieżkę lokalizacji:

```
/menu/desserts/dish
```

Taka ścieżka lokalizacji wybierze wszystkie elementy `dish` będące „dziećmi" węzłów `desserts`, będących z kolei „dziećmi" elementu głównego `menu`. Pełny zapis tej ścieżki wygląda następująco:

```
/chod::menu/chold::desserts/child:dish
```

Oś i test węzła zawsze są rozdzielane podwójnym dwukropkiem. Oś podawana przed testem węzła mówi procesorowi, gdzie szukać węzła lub zbioru węzłów. Test węzła informuje procesor, które węzły mają być faktycznie wybrane. Jawne podawanie ścieżki lokalizacji nie jest zbyt przydatne przy dopasowywaniu reguł. Ścieżka ta staje się niepotrzebnie zbyt długa i mniej czytelna. Rzadko używa się osi `child`, gdyż ta właśnie oś stosowana jest domyślnie.

Inną znaną nam osią jest oś atrybutów. Na razie znamy jej skrótowy zapis za pomocą znaku `@`. W wierszu 8. wydruku 3.12 użyliśmy takiej oto ścieżki:

```
/menu/desserts/dish[2]/@price
```

Możemy też zapisać `@price` podając oś `attribute`:

```
/menu/desserts/dish[2]/attribute::price
```

Osią, której jeszcze nie znamy, ale która jest oczywista, jest oś rodzica, `parent`. Podaje ona oczywiście rodzica bieżącego węzła kontekstu; na wydruku 3.17 pokazano przykład jej użycia.

Wydruk 3.17. *Arkusz, w którym użyto osi parent*

```
 1: <?xml version="1.0" encoding="UTF-8"?>
 2: <xsl:stylesheet version="1.0"
 3:   xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
 4:
 5:   <xsl:template match="/"><xsl:apply-templates /></xsl:template>
 6:
 7:   <xsl:template match="/menu/appetizers/dish">
 8:     <xsl:value-of select="parent::*/@title" />: <xsl:value-of select="." />
 9:   </xsl:template>
10:
11:   <xsl:template match="/menu/entrees" />
12:   <xsl:template match="/menu/desserts" />
13:
14: </xsl:stylesheet>
```

analiza W wierszu 8. powyższego wydruku użyto osi `parent` do wybrania atrybutu `title` elementu „rodzica". Jest to „rodzic" bieżącego węzła — elementu `dish`, czyli jest to element `appetizers`. Na wydruku 3.18 pokazano wynik.

źródło/wynik **Wydruk 3.18.** *Wynik zastosowania arkusza 3.17 do wydruku 3.1*

```
<?xml version="1.0" encoding="utf-8"?>

    Na rozbudzenie apetytu: Ciasteczka krabowe
    Na rozbudzenie apetytu: Krewetki tygrysie
    Na rozbudzenie apetytu: Wędzony łosoś i quesadilla z awokado
    Na rozbudzenie apetytu: Sałatka Cezara
```

analiza Na wydruku 3.18 atrybut `title` jest wstawiany do każdej reguły `dish`. Można byłoby to samo zrobić poprzez zastosowanie adresowania bezwzględnego, ale gdyby istniała osobna reguła dla elementu `dish`, elementy te w `entrees` i `desserts` także pasowałyby do podanego wzorca, a tytuł musiałby być inny. Jedynym rozwiązaniem jest właśnie użycie adresowania względnego za pomocą osi `parent`.

Istnieje jeszcze inny sposób wskazania *dowolnego* węzła „rodzica". Dla osób znających interpreter poleceń DOS lub Unix wyda się on znajomy: ścieżkę

```
parent::*/@title
```

można też zapisać jako

```
../@title
```

Drugi przykład jest znacznie bardziej zwarty i prostszy do zapisania. Jeśli jednak test węzła nie jest metaznakiem, taki zapis się nie sprawdzi i konieczne będzie jawne podanie osi `parent` oraz żądanego testu węzła.

Osie wielopoziomowe

Omawiane dotąd osie dotyczą pojedynczego poziomu zagnieżdżenia w drzewie dokumentu XML: albo powodują zagnieżdżenie jeden poziom niżej, albo jeden poziom wyżej. Niektóre osie działają przez wiele poziomów, choć nie znaczy to oczywiście, że zwracają one fragmenty drzewa, gdyż jest to niepotrzebne; do tego wystarczą osie „rodziców" i „dzieci". Wszystkie osie wielopoziomowe zwracają zbiór węzłów zawierający wszystkie elementy danej osi. Osie wielopoziomowe można traktować jako spłaszczenie drzewa dokumentu do zbioru.

Najlepszym sposobem pokazania działania tych osi jest omówienie kilku przykładów. Najłatwiej będzie zacząć od osi `ancestor` (przodka) użytej na wydruku 3.19.

Wydruk 3.19. *Arkusz stylów, w którym użyto osi ancestor*

```
 1: <?xml version="1.0" encoding="UTF-8"?>
 2: <xsl:stylesheet version="1.0"
 3:   xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
 4:
 5:     <xsl:template match="/menu/appetizers">
 6:       -<xsl:value-of select="dish[1]/ancestor::*[1]/@title" />
 7:       -<xsl:value-of select="dish[1]/ancestor::menu/entrees/@title" />
 8:     </xsl:template>
10:
11:   <xsl:template match="/menu/entrees" />
12:   <xsl:template match="/menu/desserts" />
13:
14: </xsl:stylesheet>
```

analiza Najważniejsza część tego wydruku to wiersze 6. i 7. W obu najpierw wybierany jest pierwszy węzeł „dziecko" `dish` elementu `appetizers`, który staje się węzłem kontekstu. Następnie użycie osi `ancestor` informuje procesor, że chcemy użyć węzłów przodków węzła kontekstu. Przodkami są `appetizers` i `menu`. W wierszu 6. test węzła zawiera

metaznak i predykat; w tym wypadku wybieramy pierwszego przodka — tutaj „rodzic", `appetizers`. W wierszu 7. w teście węzła sprawdzamy, czy element `menu` znajduje się w podanej osi. Wiersz ten można byłoby też zapisać jako `ancestor::*[2]`, gdyż jest to „dziadek" węzła kontekstu, a jako taki, jest drugim węzłem w osi `ancestor`.

źródło/wynik **Wydruk 3.20.** *Wynik zastosowania arkusza 3.19 do wydruku 3.1*

```
<?xml version="1.0" encoding="utf-8"?>

      Na rozbudzenie apetytu
      Czas coś zjeść!
```

analiza Jak widać na wydruku 3.20, kod z wydruku 3.19 najpierw wybiera atrybut `title` elementu `appetizers`, następnie atrybut `title` elementu `entrees`.

Istnieją jeszcze inne osie wielopoziomowe; zestawiono je w tabeli 3.1.

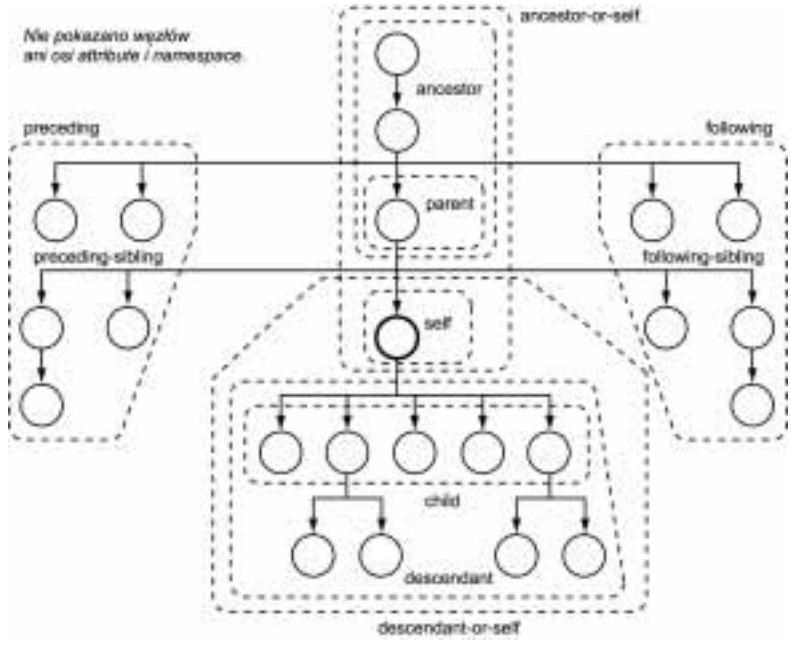
Tabela 3.1. *Zestawienie osi*

Oś	Opis
`self`	Sam węzeł kontekstu.
`child`	Wszystkie „dzieci" węzła kontekstu.
`parent`	Węzeł „rodzic" węzła kontekstu.
`ancestor`	Wszystkie węzły będące przodkami węzła kontekstu.
`ancestor-or-self`	Działa tak samo jak `ancestor`, ale pierwszym węzłem w zbiorze jest sam węzeł kontekstu.
`descendant`	Wszystkie węzły potomkowie węzła kontekstu; kolejność określana najpierw w głąb (np. „dziecko" I, „wnuk" I, „dziecko" II, „wnuk" II i tak dalej).
`descendant-or-self`	Działa tak samo jak `descendant`, ale pierwszym w zbiorze jest sam węzeł kontekstu.
`following-sibling`	Wszystkie węzły sąsiednie znajdujące się za węzłem kontekstu.
`following`	Ma takie samo znaczenie jak `following-sibling`, ale najpierw do wyniku włączane są węzły potomne (na przykład sąsiad I, „dziecko" I sąsiada I, sąsiad II i tak dalej).
`preceding-sibling`	Wszystkie węzły sąsiednie znajdujące się przed węzłem kontekstu.
`preceding`	Ma takie samo znaczenie jak `preceding-sibling`, ale najpierw do wyniku włączane są węzły potomne (na przykład sąsiad I, „dziecko" I sąsiada I, sąsiad II i tak dalej).

Na rysunku 3.5 pokazano graficzną interpretację większości osi z tabeli 3.1, aby lepiej uwidocznić znaczenie poszczególnych osi.

Wyrażenie `//` formalnie osią nie jest, ale dobrze pasuje do tego punktu. Pasuje ono do dowolnej lokalizacji w drzewie dokumentu. Na przykład `//dish` pasuje do dowolnego węzła `dish` występującego w dokumencie. Na wydruku 3.21 pokazano użycie tego wyrażenia.

Rysunek 3.5. *Graficzna interpretacja osi z tabeli 3.1*



Wydruk 3.21. *Arkusz stylów wybierający wszystkie węzły za pomocą //*

```
1: <?xml version="1.0" encoding="UTF-8"?>
2: <xsl:stylesheet version="1.0"
3:   xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
4:
5:   <xsl:template match="/">
6:     -<xsl:value-of select="text()" />
7:   </xsl:template>
8:
9: </xsl:stylesheet>
```

analiza Wydruk 3.21 da taki sam wynik jak wydruk 3.14, gdyż w wierszu 5. użyto wyrażenia // pasującego do wszystkich węzłów dokumentu, z których można dobrać węzły `dish`.

Typową pomyłką jest oczekiwanie, że `//dish[6]` pobierze szósty element `dish` z dokumentu. Niestety, z uwagi na kolejność interpretacji wyrażeń, podane wyrażenie nie zwróci żadnych węzłów. Procesor będzie szukał szóstego elementu kolejnych kontekstów „rodzica". Niestety, `appetizers` zawiera tylko cztery elementy `dish`, `entrees` pięć, a `desserts` trzy. Jak zatem rozwiązać postawiony wcześniej problem? Można po prostu użyć nawiasów: `(//dish)[6]` zwróci szósty element (czyli `frutti di mare`).

Podsumowanie

W tym rozdziale dowiedzieliśmy się, jak odwoływać się do elementów i atrybutów jako węzłów. Wiemy też, że węzeł wraz z jego „dziećmi" (elementami) nazywany

jest fragmentem drzewa. Stosując wyrażenia XPath, możemy dobierać różne części dokumentu XML. Jeśli wyrażenie dopasowujące pasuje do więcej niż jednego węzła, wynikiem jest zbiór węzłów, w którym do poszczególnych węzłów można odwoływać się przez numer w zbiorze. W zbiorze węzłów nie ma żadnej hierarchii elementów.

Węzły wybiera się za pomocą wyrażeń XPath. Wyrażenia te składają się ze ścieżek lokalizacji, które z kolei składają się z osi, testu węzła i predykatu[1]. Oś i predykat są opcjonalne. Jeśli w ścieżce brak osi, przyjmuje się oś domyślną; jeśli brak predykatu, akceptowane są wszystkie węzły pasujące do osi i testu węzła.

W następnym rozdziale będziemy dalej omawiać reguły. Wiele poznanych w tym rozdziale zasad tworzenia wyrażeń XPath zastosujemy w praktyce.

Pytania i odpowiedzi

P. Jak sprawdzić, czy wyrażenie pasuje do węzła, czy do zbioru węzłów?

O. Jeśli predykat nie dotyczy konkretnego, pojedynczego węzła lub wyrażenie nie pasuje do atrybutu, stwierdzenie tego zawczasu jest niemożliwe. Jest to działanie zgodne z oczekiwaniami, gdyż nie ma znaczenia, czy wyrażenie pasuje do węzła, czy do ich zbioru — każdy dobrany węzeł będzie przetworzony. Pod adresem http://www.topxml.com/xpathvisualizer/default.asp można pobrać narzędzie XPath Visualizer, które pomoże sprawdzić, czy podane wyrażenie jest poprawne.

P. Jak sprawdzić, czy wyrażenie w poleceniu `value-of` zwróci tekst, czy fragment drzewa?

O. Nie można tego zawczasu przewidzieć, chyba że używa się funkcji text().

P. Mój arkusz stylów stale zwraca tekst, o który wcale nie prosiłem. Jak rozwiązać ten problem?

O. Powodem może być domyślne zachowanie arkusza stylów. Trzeba sprawdzić, czy istnieje reguła dla wszystkich węzłów. Warto też sprawdzić, czy wyrażenie zwracające wartości nie pasuje do fragmentu drzewa.

Zadania

Zadaniem tej części rozdziału jest sprawdzenie, czy czytelnik opanował materiał z danego rozdziału. Warto to sprawdzić i ewentualnie postarać się zrozumieć odpowiedzi przed rozpoczęciem lektury następnego rozdziału. Odpowiedzi na pytania quizu i do zadań można znaleźć w dodatku A.

[1] W literaturze zwykle przyjmuje się, że ścieżka lokalizacji składa się z kroków lokalizacji, a każdy krok zawiera oś, test i predykat. — *przyp. tłum.*

Quiz

1. Prawda czy fałsz: oś `ancestor-or-self` zawiera węzeł kontekstu.
2. Prawda czy fałsz: predykat może zawierać ścieżkę lokalizacji.
3. Jaki będzie wynik działania elementu `<xsl:value-of select="/menu/entree/dish"/>` na wydruku 3.1?
4. Jaki będzie wynik działania elementu `<xsl:value-of select="//dish[10]/parent::*/@title"/>` na wydruku 3.1?
5. Jaki będzie wynik działania elementu `<xsl:value-of select="(//dish)[10]/parent::*/@title"/>`?

Ćwiczenie

1. Stwórz arkusz stylów, który przekształci wydruk 3.1 w eleganckie menu z nagłówkami poszczególnych grup tworzonymi na podstawie atrybutu `title`.