
XML
за 21 день

Devan Shepherd

SAMS

Teach Yourself XML in 21 Days

Second edition

SAMS

A Division of Macmillan Computer Publishing
201 West 103rd St., Indianapolis, Indiana, 46290 USA

Деван Шеперд

SAMS

Освой самостоятельно XML за 21 день

второе издание

ВИЛЬЯМС

ББК 32.973.26-018.2.75
Ш48
УДК 681.3.07

Издательский дом "Вильямс"
Зав. редакцией В.В. Александров

Перевод с английского и редакция *И.Б. Тараброва*

По общим вопросам обращайтесь в Издательский дом "Вильямс" по адресу:
info@williamspublishing.com, http://www.williamspublishing.com

Шеперд, Деван.

Ш48 Освой самостоятельно XML за 21 день, 2-е издание. : Пер. с англ. — М. : Издательский дом "Вильямс", 2002. — 432 с. : ил. — Парал. тит. англ.

ISBN 5-8459-0272-X (рус.)

Перед вами книга, посвященная одному из самых перспективных языков разметки — XML. Из настоящей книги вы узнаете, что такое языки разметки вообще, что такое правильные и действительные документы XML, а также, схемы, используемые для их проверки. Кроме того, в книге рассмотрены пространства имен XML, языки XPath, XLink, XPointer и XSL, применение DOM и SAX для обработки документов XML, а также использование таблиц стилей. В отдельную часть вынесены различные вопросы, связанные с практическим использованием XML, например, в электронной коммерции.

Книга рассчитана на пользователей с различным уровнем подготовки.

ББК 32.973.26-018.2.75

Все названия программных продуктов являются зарегистрированными торговыми марками соответствующих фирм.

Никакая часть настоящего издания ни в каких целях не может быть воспроизведена в какой бы то ни было форме и какими бы то ни было средствами, будь то электронные или механические, включая фотокопирование и запись на магнитный носитель, если на это нет письменного разрешения издательства Sams Publishing.

Authorized translation from the English language edition published by Sams Publishing
Copyright © 2001
All rights reserved. No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording or by any information storage retrieval system, without permission from the Publisher.

Russian language edition published by Williams Publishing House according to the Agreement with R&I Enterprises International, Copyright © 2001

ISBN 5-8459-0272-X (рус.)
ISBN 0-6723-2093-2 (англ.)

© Издательский дом "Вильямс", 2002
© Sams Publishing, 2001

Оглавление

Введение 16

Неделя I. Основы разметки 19

День 1-й. Обзор языков разметки 20
День 2-й. Подробные сведения о синтаксисе XML 37
День 3-й. Правильные экземпляры XML 51
День 4-й. Определение типа документа 63
День 5-й. Схема XML-Data Reduced 93
День 6-й. Язык определения схем XML 123
День 7-й. Ссылки XML 137

Неделя II. Обработка XML 145

День 8-й. Пространства имен XML 146
День 9-й. Язык XPath 157
День 10-й. Язык XLink 183
День 11-й. Язык XPointer 199
День 12-й. Объектная модель документа XML — DOM 207
День 13-й. Simple API для XML 225
День 14-й. Использование каскадных таблиц стилей CSS 245

Неделя III. Заставьте XML работать 263

День 15-й. Расширяемый язык таблиц стилей и его объекты форматирования 264
День 16-й. Язык преобразований XSLT 285
День 17-й. Связывание данных XML с элементами HTML 315
День 18-й. Технологии XBase и XInclude 331
День 19-й. Интеграция XML с корпоративными бизнес-моделями 337
День 20-й. Электронная коммерция и XML 361
День 21-й. Разработка Web-приложения с помощью XML 379

Часть IV. Приложения 395

Приложение А. Ответы на вопросы 396
Приложение Б. Ресурсы 415

Предметный указатель 421

Содержание

Введение 16

Неделя I. Основы разметки 19

День 1-й. Обзор языков разметки 20

Web — революционный феномен 20
Зачем нужен еще один язык разметки 21
Вам следует знать семь фактов об XML 23
1. XML предлагает метод представления структурированных данных в виде текстового файла 24
2. XML похож на HTML 25
3. XML понятен как компьютеру, так и человеку 29
4. XML образует целое семейство технологий 29
5. XML достаточно гибок 30
6. XML относительно нов, но имеет серьезные корни 30
7. XML свободен от лицензионных отчислений, платформно-независим, а также может похвастаться широкой поддержкой 31
Роль электронного бизнеса 31
Создание вашего первого документа XML 32
Резюме 34
Вопросы и ответы 35
Упражнение 36

День 2-й. Подробные сведения о синтаксисе XML 37

Создание собственных элементов XML 37
Программные средства для работы с XML 37
Зависимость от платформы, программного обеспечения и операционной системы 38
Синтаксические правила XML 38
Правильное вложение элементов 44
Соглашения о присвоении имен в XML 44
Декларации XML 45
Добавление комментариев к коду XML 47
Размещение атрибутов в экземпляре XML 48
Резюме 49
Вопросы и ответы 49
Упражнение 50

День 3-й. Правильные экземпляры XML 51

Зачем беспокоиться о правильности XML 51
Проверка типа данных — еще одна причина для определения правильности 54
Как определить структуру данных 56
Структура рецепта приготовления блюда 56
Вы уже можете использовать структурированные данные 57
Вы можете использовать XML для добавления структуры к неструктурированным данным 58
Парсер проверяет правильность 59

Неужели правильность — это все, что нам нужно? 61
Резюме 61
Вопросы и ответы 61
Упражнение 62

День 4-й. Определение типа документа 63

Откуда появились схемы DTD 63
Знакомство с правилами 66
Синтаксис объявления типов основных элементов в DTD 66
Простой элемент с текстовым содержанием 66
Элемент, содержащий другой элемент 68
Объявление пустого элемента 69
Использование ключевого слова ANY схемы DTD 70
Модели смешанного содержания 71
Подведение итогов по моделям содержания 72
Объявление атрибутов в схеме DTD 73
Объявления атрибутов строкового типа 73
Объявления атрибутов типа #FIXED 74
Объявления атрибутов типа #IMPLIED 75
Атрибуты маркированных типов в схемах DTS 75
Использование в схемах DTD атрибутов типов с перечислением 80
Индикаторы вхождения и объявления последовательностей в схемах DTD 82
Анализ правильных экземпляров XML 84
Объявление внешних схем DTD 88
Резюме 91
Вопросы и ответы 91
Упражнение 92

День 5-й. Схема XML-Data Reduced 93

Зачем нужны другие схемы 93
Подробно о XDR 97
Объявления элементов и атрибутов XDR 98
Объявления ElementType 98
Вложенные объявления ElementType 102
Вложенные элементы 104
Проверка атрибутов в экземпляре XML с помощью схемы XDR 105
Типы данных в схемах XDR 108
Примеры типов данных 109
Индикаторы вхождения в схемах XDR 113
Дополнительные ограничения типов данных в схемах XDR 114
Определение групп содержания 116
Дополнительные преимущества схем XDR 119
Расширяемость в схемах XDR 120
Преобразование схем XDR 120
Резюме 121
Вопросы и ответы 121
Упражнение 121

День 6-й. Язык определения схем XML 123

Схема, предложенная консорциумом W3C 123
Статус языка XSD 124
Рекомендация консорциума W3C 124
Основы XSD 125

Пространства имен XSD 125
Простые типы элементов 126
Типы данных XSD 128
Определения элементов сложных типов 129
Ограничения вхождений в схемах XSD 131
Атрибуты в схемах XSD 133
Три подхода к проверке документов: DTD, XDR и XSD 135
Резюме 136
Вопросы и ответы 136
Упражнение 136

День 7-й. Ссылки XML 137

Два вида ссылок 137
Общие ссылки 138
Параметрические ссылки 141
Ссылки в других языках описания схем 142
Резюме 143
Вопросы и ответы 143
Упражнение 143

Неделя II. Обработка XML 145

День 8-й. Пространства имен XML 146

Зачем необходимы пространства имен 146
Как выглядит конфликт имен 147
Предотвращение конфликтов имен
с помощью объявления пространств имен 149
Синтаксис объявления пространства имен XML 150
Ошибки при объявлении пространств имен 150
Уникальная идентификация пространств имен XML 151
Каким образом приложения используют пространства имен 153
Использование объявлений пространств имен 153
Избранные стандартные идентификаторы URI пространств имен XML 155
Резюме 155
Вопросы и ответы 155
Упражнение 156

День 9-й. Язык XPath 157

Что такое XPath 157
Узлы дерева документа XML 158
Семь узлов документа в XPath 158
Взаимосвязи узлов 159
Знакомство с узлами экземпляра документа XML 159
Сложные отношения узлов 161
Разновидности отношений XPath 163
Концепция self 164
Родительский узел 165
Дочерние отношения 165
Родительские и узлы более высокого уровня 167
Предок самого себя 167
Потомок 169
Потомок самого себя 169
Выбор узлов, следующих после контекстного узла 169

Выбор последующих узлов одного уровня 170
Предыдущие узлы 170
Предыдущие узлы одного уровня 172
Создание и проверка выражений XPath 173
Выбор узлов по имени с помощью косой 174
Резюме 180
Вопросы и ответы 180
Упражнение 181

День 10-й. Язык XLink 183

Связывание гипертекста 183
Ссылки HTML 184
XLink расширяет возможности ссылок HTML 186
Ограничения XLink 187
Связующие элементы 187
Пример использования XLink 188
Проверка документа 189
Описательные атрибуты XLink 191
Поведение простых ссылок 191
Представление 191
Приведение в действие 192
Исследование поведений XLink 192
Расширенные ссылки 194
Элементы-локаторы 195
Элементы arc 196
Пример сложной расширенной ссылки 197
Резюме 198
Вопросы и ответы 198
Упражнение 198

День 11-й. Язык XPointer 199

Именованные фрагменты и гипертекстовые ссылки указывают HTML 199
Ограничения анкерных ссылок HTML 200
Выражения XPointer 201
Синтаксис XPointer 201
Идентификационные ссылки XPointer 202
Абсолютная адресация 203
Выражения отношений 203
Числовой выбор 204
Тип узла 204
Фильтры атрибутов и их значений 205
Резюме 205
Вопросы и ответы 205

День 12-й. Объектная модель документа XML — DOM 207

DOM для XML 207
Объектные модели 209
Создание структур DOM с помощью сценариев 210
Отношения DOM 210
DOM как универсальный прикладной интерфейс API 211
Основные типы API 211
Вызов DOM 211
Свойства, методы и события объектов DOM 213

Загрузка файла XML с указанного адреса URL 215
Отображение результатов на экране 217
Выбор узла по его расположению 218
Выбор элементов XML по имени 219
Сообщения об ошибках DOM 221
Резюме 223
Вопросы и ответы 224
Упражнение 224
Упражнение 2 224

День 13-й. Simple API для XML 225

Что такое SAX 225
Синтаксические анализаторы SAX 226
Сравнение SAX и DOM 226
Выбор SAX вместо DOM 228
Выбор DOM вместо SAX 229
Методы, вызываемые при обработке событий SAX 229
SAX и Java 230
Необходимое программное обеспечение для создания Java-программы 230
Java-приложение для получения списка событий SAX 231
Импортирование классов 231
Расширение класса HandlerBase 232
Метод main 233
Методы SAX 234
Компиляция и выполнение EList 239
SAX и пробелы 240
Обработка ошибок, возникших при работе обработчика SAX 241
Проверка действительности документов с помощью SAX 242
Резюме 242
Вопросы и ответы 242
Упражнение 243

День 14-й. Использование каскадных таблиц стилей CSS 245

Стиль и содержание 245
Каскадные таблицы стилей и HTML 246
Поддержка каскадных таблиц стилей броузерами 247
CSS и XML 248
Создание простой таблицы стилей 249
Определение стилей 250
Связывание с файлом CSS 250
Свойства каскадных таблиц стилей CSS 252
Свойства стилей текста 252
Свойства шрифтов 253
Свойства цвета и фона 254
Свойства рамки 255
Свойства отображения 257
Полученный стиль 258
Использование нескольких элементов в качестве селекторов 259
Ограничения CSS 259
Резюме 259
Вопросы и ответы 260
Упражнение 260

Неделя III. Заставьте XML работать 263

День 15-й. Расширяемый язык таблиц стилей и его объекты форматирования 264

Добавление форматирования к структуре 264
Соглашения об использовании имен XSL 265
Создание и использование объектов форматирования 265
Операционный XSL 266
Обработчик объектов форматирования Apache FOP 267
Объекты форматирования 269
Основная конструкция объекта форматирования 274
Атрибуты шрифта 278
Стили элементов 280
Таблицы XSL-FO 281
Резюме 283
Вопросы и ответы 284
Упражнение 284

День 16-й. Язык преобразований XSLT 285

Преобразование из одной структуры к другой 285
Операционный XSLT 286
Параметры обработки XSLT 287
Правила поведения в организации: используем XSLT 287
Установка инструментов XSLT 288
XSLT-обработчик, работающий из командной строки 289
Редакторы XML, поддерживающие преобразования XSLT 290
Использование XSLT 290
Замечания о пространстве имен XSLT 291
Обработка событий на основе правил 291
Преобразования XML в HTML 293
Итерационная обработка XSLT 296
Создание HTML-страницы с помощью XSLT 301
Сортировка XSLT 304
Добавление атрибутов стилей с помощью XSLT 306
Профессиональное использование преобразований XSLT 309
Элемент xsl:for-each 309
Элемент xsl:if 309
Элементы xsl:choose, xsl:when и xsl:otherwise 310
Резюме 312
Вопросы и ответы 312
Упражнение 313

День 17-й. Связывание данных XML с элементами HTML 315

Создание простых экземпляров связывания данных 315
Структура документа XML 316
Связывание документа XML с HTML-страницей 318
Размещение ссылки XML на HTML-странице 319
Связывание элементов HTML с элементами XML 320
Создание HTML-таблиц, содержащих данные XML 321
Добавление элементов JavaScript для управления протоком данных 323
Просмотр данных одной записи 325
Вызов XMLDSO с помощью Java-аплета 328

Сообщения об ошибках 329
Резюме 330
Вопросы и ответы 330
Упражнение 330

День 18-й. Технологии XBase и XInclude 331

XML Base 331
Атрибут xml:base 331
Несколько атрибутов XBase 332
XML Include 333
Обработка XML во время включения 335
Выражения XPointer 335
Поддержка XInclude 335
Резюме 335
Вопросы и ответы 336

День 19-й. Интеграция XML с корпоративными бизнес-моделями 337

Создание бизнес-моделей с использованием технологий XML 337
Основные положения анализа документов 338
Шаги анализа XML 339
Другие формы анализа информации 341
Бизнес-приложения 341
Приложения для распространения и сбора информации 341
Решения для интеграции приложений 343
Приложения интеграции данных 344
Трехуровневая Web-архитектура 345
Уровень клиента 345
Промежуточный уровень 345
Уровень данных 346
Использование XML на различных уровнях 346
Уровень данных 347
Определение имени источника данных ODBC 349
Настройка Web-сервера 351
Доступ к базе данных с помощью сценария на стороне сервера 351
Создание документа XML с помощью сценария 355
Дальнейшая обработка 357
Резюме 359
Вопросы и ответы 359

День 20-й. Электронная коммерция и XML 361

Применение XML в бизнесе 361
Преимущества ведения электронного бизнеса 361
B2C-приложения 362
Приложения для персонализации в Internet 363
Узлы сбора данных 365
Интеграция звеньев поставок 366
От систематизации к обмену данных 367
Вызовы объекта HTTP в XML 367
XML-RPC и SOAP 369
Web Distributed Data Exchange (WDDX) 373
ebXML 374
Резюме 375
Вопросы и ответы 376

День 21-й. Разработка Web-приложения с помощью XML 379
Проектирование Web-приложения 379
Сохранение записей в документе wine_notes.xml 380
Создание Web-формы wine_notes.html 381
Применение стилей к Web-узлу с помощью таблицы стилей wine_notes.css 384
Конструирование начальной страницы index.html 385
Управление данными на сервере с помощью сценария add_notes.asp 386
Отображение сведений с помощью сценария view_notes.asp 389
Преобразование содержания с помощью документа wine_notes.xsl 390
Создание Web-узла 392
Резюме 394
Вопросы и ответы 394
Упражнение 394

Часть IV. Приложения 395

Приложение А. Ответы на вопросы 396
Возможное решение для упражнения дня 1-го 396
Возможное решение для упражнения дня 2-го 396
Возможное решение для упражнения дня 3-го 397
Возможное решение для упражнения дня 4-го 398
Возможное решение для упражнения дня 5-го 398
Возможное решение для упражнения дня 6-го 399
Возможное решение для упражнения дня 7-го 400
Возможное решение для упражнения дня 8-го 401
Возможное решение для упражнения дня 9-го 401
Возможное решение для упражнения дня 10-го 402
Возможное решение для упражнения дня 12-го 403
Упражнение — часть 2 404
Возможное решение для упражнения дня 13-го 405
Возможное решение для упражнения дня 14-го 406
Возможное решение для упражнения дня 15-го 407
Возможное решение для упражнения дня 16-го 409
Возможное решение для упражнения дня 17-го 409
Возможное решение для упражнения дня 21-го 410

Приложение Б. Ресурсы 415
Некоторые рекомендации W3C 415
Некоторые рабочие черновики консорциума W3C 417
Некоторые ноты консорциума W3C 418
Рекомендуемые Web-узлы, содержащие сведения об XML 418
Полезные редакторы XML 419
Другие программные средства для работы с XML 420

Предметный указатель 421

Об авторе

Деван Шеперд (Devan Shepherd) занимается информационными технологиями уже больше 25 лет; за это время он успел поработать разработчиком, консультантом, инструктором и даже старшим менеджером. Деван разработал учебные курсы для колледжей, посвященные таким дисциплинам, как технологии XML, HTML, языки написания сценариев, разработка Web-страниц, а также широкий спектр разнообразных языков программирования. Сейчас Деван считает технологии XML полноценным индустриальным стандартом и повсеместно использует их в своей практике, включая разработку решений для электронного бизнеса, обучения и системной интеграции. Он занимал ведущие посты во многих компаниях государственного и частного сектора, получив опыт работы в таких сферах, как здравоохранение, оборона, банковское дело, а также юриспруденция. Называя самого себя "е-трудоголиком", Деван старается проводить все свое свободное время вместе с женой и детьми, занимаясь "неэлектронными" делами.

Посвящение

Эта книга посвящается моей жене Кэтрин Шеперд — другу, партнеру, любимой и музе, чьи внимание и поддержка никогда меня не покидали.

Благодарности

Я хотел бы поблагодарить Кэрол Акерман (Carol Ackerman) за ее поддержку и за то, что написание настоящей книги стало действительно простым и полностью управляемым процессом, независимо от того, каким сложным оно казалось в самом начале.

Большое спасибо Хезер Гуделл (Heather Goodell) за то, что она поделилась своим многолетним опытом и улучшила те разделы книги, которые я сам смог написать небезупречно.

Спасибо Фрэнку Нейжебауэру (Frank Neugebauer) за его помощь в освоении всех тонкостей технологий XML, а также за проверку каждой строки кода в настоящей книге.

Спасибо моим друзьям и коллегам, особенно Брайану Тревису (Brian Travis) (президенту компании *Architag.com*), Бобу Гонзалесу (Bob Gonzales) (настоящему гуру XML, который помог мне при описании некоторых технических нюансов), а также Джину Йонгу (Gene Yong) (эксперту по языкам разметки и структуре информационных систем). Спасибо вам за неоценимую помощь при написании книги. А также спасибо за целое море пива.

Я не могу не поблагодарить чрезвычайно профессиональную и эффективно работающую команду сотрудников издательства *Sams*, отвечающую за выполнение всех работ при подготовке этого проекта. Профессионализм этих людей обеспечивает одну из наиболее эффективных моделей ведения бизнеса в издательской сфере. Спасибо вам за то, что придали настоящей книге действительно привлекательный вид.

Спасибо моим родителям, Мерренне и Дугласу Шеперд, за все то, чему они меня научили, а также за обеспечение "пожизненной" моральной поддержки. Спасибо за все, а особенно за слова о том, что я должен написать книгу обо всех тех компьютерных технологиях, с которыми мне приходится работать.

И наконец, я хотел бы поблагодарить своих детей, Грега и Кристен. Спасибо тебе, Грег, за то, что во время моего отсутствия ты взял на себя заботы о ведении домашнего хозяйства. Спасибо тебе, Кристен, за те замечательные компьютерные игры, которые ты присыла мне по электронной почте.

Предисловие

Я уверен, что все мы испытываем некоторое возбуждение, приобретая новое программное обеспечение, и практически всегда испытываем разочарование от прилагаемой к нему документации. Вы никогда не задавали себе вопрос: почему такая замечательная команда разработчиков, создавшая поистине восхитительный новый программный продукт, не в состоянии объяснять людям, как же эту программу следует использовать? В конце концов, ведь никто лучше самих разработчиков не знает всех преимуществ программы, а также всех задач, которые она призвана решать.

Многие профессиональные программисты-разработчики, легко решающие разнообразные проблемы технического характера, далеко не всегда способны объяснить другим, что же они действительно делают; они всегда пропускают некоторые шаги, предполагая, что аудитория обязательно обладает всеми необходимыми знаниями по проблеме, что далеко не всегда именно так. Хотя найти и нанять технически грамотных сотрудников достаточно сложно, наша компания всегда старалась принимать на работу людей, которые обладают не только всеми необходимыми техническими знаниями, *но* и способностью объяснять другим различные тонкости соответствующих информационных технологий.

Для того чтобы выжить в бизнесе, занятом обучением и подготовкой специалистов по информационным технологиям, вы должны взять на себя всю ответственность за обучение студентов. Недостаточно просто изложить материал и основные концепции; вы должны быть уверены в том, что информация подается в том виде, который позволяет студентам эффективно ее усваивать и применять на практике для решения поставленных перед ними проблем. Преподаватели должны обладать некоторым актерским даром, интеллигентностью и умением быстро продемонстрировать, как определенные инструменты и технология применяются в разнообразных ситуациях. Людей, которые бы удовлетворяли подобным требованиям, найти чрезвычайно сложно.

Впервые я встретил Девана Шеперда на выставке информационных технологий Comdex, на которой он выступал. До этого о нем я знал только по положительным отзывам своих знакомых. Немного поработав вместе с Деваном, я был поражен глубиной его технических знаний, его пониманием нюансов процесса обучения, его пониманием места информационных технологий в современном мире. Работая вместо, мы часто встречали людей, которые были неплохо осведомлены о XML и его потенциальных возможностях, но при этом считали его скорее средством создания различных интересных "примочек", а не инструментом бизнеса, позволяющим решать много насущных проблем. Другими словами, эти люди восхищались XML только как собственно технологией, упуская из виду, что XML дает потрясающие решение многих характерных для бизнеса задач, когда на первое место выходят возможности управления, а не различные забавные приемы программирования.

Студенты Девана в значительной мере выигрывали от его способности доступно излагать вопросы, касающиеся технологических тонкостей и эффективного ведения бизнеса. Излагая материал, посвященный XML, Деван предлагал своим читателям подробные пошаговые инструкции, которые можно смело считать настоящим произведением искусства. Он предлагает подробные и всегда предварительно проверенные примеры программного кода, позволяющего решать те или иные проблемы, которые читатели могут легко обобщать для их собственных задач. Очень ценным окажется список Web-узлов, на которых можно найти массу полезного материала, посвященного XML и смежным вопросам. Выполнив упражнения в настоящей книге, читатели смогут действительно разрабатывать решения на базе XML.

- Брайан Снайдер

Брайан Снайдер — бывший президент компании *Architag International*, которая занимается обучением и консультациями по XML. Он обладает значительным опытом в сфере обучения, корпоративных финансов и стратегии ведения бизнеса. Сейчас работает в одной из ведущих компаний высоких технологий.

Введение

XML (Extensible Markup Language — Расширяемый язык разметки) — это *lingua franca* современной Internet. От XML вы получаете полностью расширяемый, простой для изучения и обладающий широкими возможностями формат для структурированных данных и документов, эффективный обмен которыми осуществляется через Internet.

Несмотря на то, что буква M в аббревиатуре XML означает Markup (разметка), XML не является языком разметки в узком значении этого термина. Скорее это метаязык, для описания в высшей степени структурированных и узкоспециализированных словарей разметки.

Независимо от того, занимаетесь вы системами публикации и управления данными, проектированием сложных решений в электронном бизнесе или обеспечением эффективного обмена различными научными данными, XML предоставляет вам возможность создания профессиональных, самоописывающихся и управляемых данными решений. Во многих ситуациях ваши инструменты базируются на словарях, определенных с помощью синтаксиса XML. Овладев XML, вы одновременно изучите целое семейство специализированных технологий, сосуществующих в виде элегантного симбиоза в различных архитектурах и на разных платформах. Семейство технологий XML обеспечивает независимость от конкретных платформ и форматов, что позволяет организовать доставку данных неограниченному кругу пользователей, включая приложения, броузеры, беспроводные устройства с поддержкой Web, а также еще многие, пока неведомые нам инструменты. XML может превратиться в *технологию безграничных возможностей*, без которой невозможна дальнейшая эволюция Internet.

Книга *Освой самостоятельно XML за 21 день, 2-е издание*, содержит весь необходимый материал для освоения работы с XML, а также несколькими смежными с ним стандартами, включая XSL (Extensible Spreadsheet Languages — Расширяемые языки таблиц стилей), Xpath ((Extensible Path Language — Расширяемый язык путей), XLL (Extensible Link Language — Расширяемый язык связей), XPointer, XML-Data Reduced, XML Schema и некоторые другие. Вы увидите массу работающих примеров программного кода, иллюстрирующих все эти технологии, которые вы сможете использовать во время обучения.

Кому предназначается настоящая книга

XML — это одна из самых революционных компьютерных технологий, появившихся в Web в начале 1990-х годов. Вы уже наверняка знакомы с языком HTML (HyperText Markup Language — Язык разметки гипертекста) и использовали его при создании собственных Web-страниц. В настоящей книге предполагается, что вы достаточно опытный разработчик, знакомы со многими технологиями и языками программирования. Некоторые примеры программного кода в настоящей книге используют JavaScript, VBScript, Java, базы данных, Active Server Pages, приемы объектно-ориентированного программирования и т.д. Основная цель, однако, состоит не в том, чтобы научить вас одному из этих языков программирования, а в том, чтобы продемонстрировать вам, насколько элегантно XML внедряется в реальные решения, воплощая все преимущества компонентных технологий. Даже если вы не профессиональный программист, пусть этот факт не отпугивает вас от изучения настоящей книги. Большое количество работающих примеров программного кода и их подробный анализ позволят вам разобраться во всех тонкостях использования данных технологий.

Соглашения, принятые в настоящей книге

В настоящей книге используются различные стили представления информации. Например, примеры кода, команды, инструкции и любой текст, которые вы вводите самостоятельно или видите на экране монитора, отображаются моноширинным шрифтом. Вот пример программного кода:

```
<?xml version="1.0"?>
<mybook>
    <title>Sams Teach Yourself XML in 21 Days, Second Edition</title>
    <author>Devan Shepherd</author>
    <publisher>
        Sams Publishing, a Division of Pearson Edication, Inc.
    </publisher>
    <ISBN>0-672-32993-2</ISBN>
    <year>2001</year>
</mybook>
```

Большая часть примеров подвергается всестороннему анализу, причем чаще всего рассмотрение ведется построчно. Для того чтобы облегчить изучение той или иной строки программного кода, я буду использовать нумерацию строк, как показано ниже:

```
1:  1: <?xml version="1.0"?>
2:  <mybook>
3:   <title>Sams Teach Yourself XML in 21 Days, Second Edition</title>
4:   <author>Devan Shepherd</author>
5:   <publisher>
6:       Sams Publishing, a Division of Pearson Edication, Inc.
7:   </publisher>
8:   <ISBN>0-672-32993-2</ISBN>
9:   <year>2001</year>
10: </mybook>
```

Нумерация строк не используется в неполном коде или фрагментах кода, если только они не слишком длинны.

Если вы создаете собственные документе на основе предоставленных листингов, вам не нужно указывать номера строк, они используются исключительно для обучения.

- Строки программного кода, команды, переменные, а также текст, который вы видите на экране, отображается моноширинным шрифтом, который вы уже видели в примерах кода.
- Подстановочные элементы отображаются моноширинным шрифтом с курсивным начертанием. Замените подстановочный элемент действительным именем файла, параметром или любым другим элементом, который он представляет.

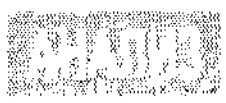
В настоящей книге вы также увидите целый ряд пиктограмм, указывающих на информацию определенного типа.



Здесь содержатся специальные сведения о тестировании, отладке, производительности, а также примеры добротного программирования.

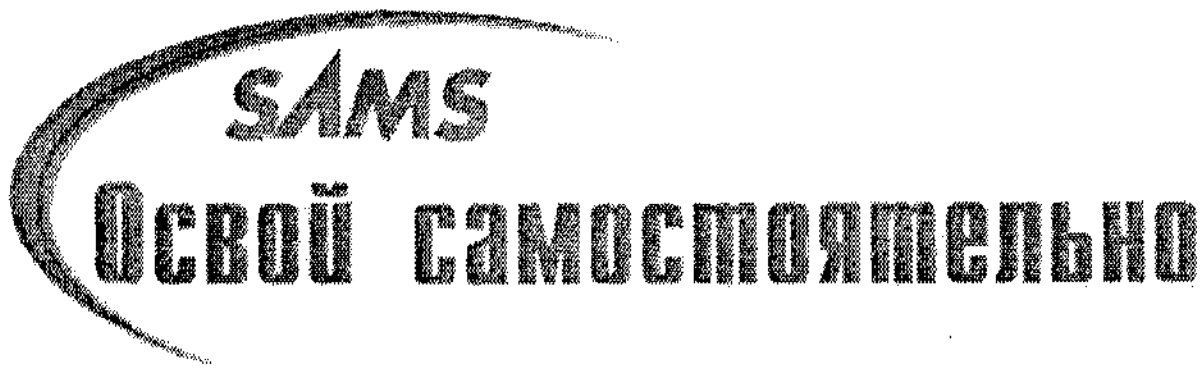


Здесь содержатся справочные сведения, ссылки на ресурсы в Internet и т. д.



Значительная часть примеров программного кода анализируется; здесь подробно рассматриваются новые концепции, средств, функций, грамматика, синтаксис или стиль, впервые представленные в примере.

Неделя I



Основы разметки

1. Обзор языков программирования
2. Подробные сведения о синтаксисе XML
3. Правильные экземпляры XML
4. Определение типа документа
5. Схема XML-Data Reduced
6. Язык определения схем XML
7. Ссылки XML

День 1-й

Обзор языков разметки

Во время первого дня обучения вы познакомитесь с историей Internet, эволюцией World Wide Web (или попросту Web), а также ролью, которую играет в мире Interne консорциум *W3C* (World Wide Web Consortium).

Сегодня вы изучите следующие темы.

- Роль XML в мире электронного бизнеса.
- Ограничения языка HTML, которые не могут не отражаться на всей Web.
- Что такое SGML (Standard Generalized Markup Language — стандартный обобщенный язык разметки) и какова связь между SGML, XML и HTML.
- Семь основных характеристик XML.
- Как приступить к созданию собственных XML-документов, пользуясь элементами разметки с самоописанием.

Web — революционный феномен

За последние несколько лет Web прошла путь от небольшой новеллы до целой мировой библиотеки. В основе этого глобального репозитария данных таится концепция "гипертекста" (hypertext). Этот термин впервые ввел Тэд Нельсон (Ted Nelson) в 1965 году для описания процесса объединения данных, поступающих с различных компьютеров.

ПРИМЕЧАНИЕ

Тэд Нельсон изложил эти идеи в своей работе *A File Structure for the Complex, the Changing, and the Indeterminate* на 20-й национальной конференции, которая проходила в Нью-Йорке под патронатом ассоциации по вычислительной технике (Association for Computing Machinery) в 1965 году. Подробные сведения о Тэде Нельсоне и его проекте Xanadu вы найдете в Internet по адресу `http://jefferson.village.virginia.edu/elab/hf10155.html`.

В конце 1960-х годов Дуг Энгельбарт (Doug Engelbart) создал прототип NLS (сокращение от oNLine System), который объединял средства просмотра и редактирования гипертекста, электронную почту и другие компоненты, которые сегодня в нашем сознании прочно ассоциируются с Web.

Однако границы Web значительно расширились. Огромное количество мультимедийных презентаций, видео, графики, звука и ряда других технологий добавилось к исходному изобретению Тима Бернерса-Ли (Tim Berners-Lee), общепризнанного отца Web, который в 1989 году предложил универсальную систему связанной информации, в которой общность и переносимость играют намного более важную роль, чем графические изображения (подробные сведения вы найдете по адресу `http://www.w3.org/History/1989/proposal.html`).

Начиная с 1980 года, Бернерс-Ли в рамках программы "Enquire-Within-Upon-Everything" объединил огромное количество гиперссылок, обеспечивающих практически незаметное перемещение между сведениями, хранящимися на нескольких компьютерах. Позже (в 1990 году) он назвал свое творение World Wide Web. Несмотря на то, что он начинал с несколько усложненных представлений (в конце концов, Web задумывалась как централизованное хранилище всей информации человечества), организация доступа к подобным сведениям оказалась практичной и достаточно простой.

Методология, разработанная Бернерсом-Ли во время работы в ораганизации CERN (Conseil Europeen pour la Recherche Nucleaire — Европейская организация по ядерным исследованиям), предназначалась для обеспечения обмена документами с помощью протокола TCP/IP (Transmission Control Protocol/Internet Protocol — Протокол управления передачей/Протокол Internet). В частности, Бернерс-Ли предложил следующее.

- *URL* (Uniform Resource Locator — унифицированный локатор ресурса) — новый универсальный метод адресации документов Internet.
- *HTTP* (Hypertext Transfer Protocol — протокол передачи гипертекстовых файлов) — новый протокол семейства TCP/IP.
- *HTML* (HyperText Markup Language — язык разметки гипертекста) — новый язык описания документов

Все это было представлено миру в виде целого ряда программных средств (броузеров), первоначально написанных на Objective C для платформы NeXT, а затем перенесенных на другие платформы с помощью языка C.

Все эти сведения должны были показать, что за разноцветными баннерами, анимацией, потоковыми видео и аудио, которые стали основными характеристиками современной Web, лежат наборы простых текстовых документы (созданных преимущественно с использованием HTML, а также некоторых других технологий), которые легко перенести с одного компьютера на другой с помощью Internet.

Зачем нужен еще один язык разметки

В основе бурного развития Web лежит эффективная разметка Web-страниц с помощью языка HTML. С помощью HTML Web-программист легко создает простые гипертекстовые ссылки, которые надежно выполняют свою работу. С точки зрения HTML-страницы ссылка может активизироваться (после щелчка пользователя на ней мышью) для загрузки содержимого другой страницы.

HTML обеспечивает переносимость, которая в других технологиях просто отсутствует. Переносимость — одна из основных целей, которые преследовались при проектировании Web.

Существует некоторая кривая обучения, связанная с HTML. Практически каждый человек, от школьника с его любопытством до бабушек и дедушек, хранящих фотографии внуков, может без особых усилий публиковать различные сведения в Web. Соответствующие редакторы и инструменты разработки с замечательным графическим интерфейсом пользователя можно приобрести за достаточно скромную сумму или вообще бесплатно загрузить из Internet. Интерфейсы современных броузеров, предлагая широкий спектр возможностей, очень просты в использовании.

Если HTML настолько хорош, зачем вам вообще связываться с изучением XML?

Однако для HTML характерен целый ряд серьезных ограничений. Язык, изначально предназначавшийся для разметки данных, превратился в инструмент презентации в Web. Правда, как инструмент презентации HTML оказывается весьма слабым, обеспечивая недостаточный контроль над расположением и управлением "белым пространством" на странице, например. Для HTML характерны проблемы, связанные с определением расстояния между символами, выравниванием текста, а также расстановкой переносов. Расположение данных в несколько колонок, например как в газетах, в HTML достаточно проблематично.

При создании HTML включение в язык различных стилей даже не планировалось. Дескрипторы событий для таких обычных стилей, как курсив (<I>) или полужирный (<B>), были добавлены в стандарт языка значительно позже. Подобные стили преимущественно используются для выделения определенных фраз или элементов страницы; однако при этом нельзя отделить дескрипторы стилей от заключенных в них сведений. Разделение содержимого и стилей характерно именно для XML. XML — это ориентированный на работу с метаданными язык, которые определяет разметку данных, но не стилей. Вопросы стилей обрабатываются отдельно от данных. Вы научитесь добавлять стили к разметки данных несколькими способами. В уроке дня 14-го "Использование каскадных таблиц стилей CSS" я расскажу вам о добавлении в XML-документ таблиц каскадных стилей CSS (Cascading Style Sheets); это делается практически так же, как и при использовании HTML 4.01. В уроке дня 15-го "Расширяемый язык таблиц стилей и его объекты форматирования" вы познакомитесь с расширяемым языком таблиц стилей XSL (Extensible Stylesheet Language) и такими его компонентами, как объекты форматирования FO (Formatting Object). В уроке дня 16-го "Язык преобразований XSLT" вы узнаете, как преобразовывать к стандарту HTML XML-документы с помощью XSLT (Extensible Stylesheet Language Transformations — преобразования расширяемого языка таблиц стилей).

Чтобы считаться средством разметки данных, языку HTML явно не достает гибкости. Существует строго определенный и достаточно ограниченный набор дескрипторов, который нельзя расширять. По мере выпуска за последние годы новых версий HTML в них появлялись новые функциональные возможности, но всем этим нововведениям свойственна некоторая непоследовательность. Очень часто новые дескрипторы HTML получали поддержку только со стороны броузера одной определенной компании; таким образом нарушается основное правило Web-разметки, которое гласит, что все приложения должны иметь возможность взаимодействовать, а передача данных должна быть независимой.

Консорциум *W3C* намерен прекратить выпуск новых версий HTML. Предполагается, что текущая версия (4.01) будет окончательной. Возможно, что вы уже слышали об XHTML и хотите узнать, что же это такое. XHTML — этот тот же HTML, но переформулированный как приложение XML. Подробно о правилах XML вы узнаете из текущей и последующей глав. В частности, вы увидите, что HTML *может* выступать в роли XML, если будет следовать всем синтаксическим правилам последнего. На самом деле, спецификации XHTML 1.0 и HTML 4.01 подобны, но элементы каждого из них связаны с различными пространствами имен, т.е. наборами допустимых элементов. Подробно о пространствах имен мы поговорим в уроке дня 8-го.

HTML не обеспечивает полной поддержки модульности или повторного использования программного кода, как в других языках. В объектно-ориентированных языках, таких как C#, C++, Visual Basic и Java, существуют классы, которые обеспечивают возможность совместного и повторного использования объектов и методов несколькими программами. В уроке дня 18-го "Технологии XBase и XInclude" вы познакомитесь с новой технологией XInclude, которая должна обеспечить функции повторного использования кода для программистов на XML. В уроке дня 10-го "Язык XLink" вы познакомитесь с подобной концепцией, но базирующейся на приемах внедрения, доступных в приложениях, поддерживающих технологию XLink. К сожалению, HTML ни одной из таких возможностей не обладает.

Для HTML характерен целый ряд непоследовательностей, которые допускают публикацию "плохого" программного кода. XML заставляет использовать "приемы хорошего программирования" при неукоснительном применении определенных синтаксических правил. С большинством этих правил вы познакомитесь в настоящей и последующей главах, но в качестве примера я приведу вам такие сведения: некоторые версии HTML позволяют не использовать закрывающие дескрипторы в некоторых видах разметки. Броузерам приходится поддерживать смешанные наборы правил, которые становятся общей характеристикой HTML. Например, в HTML абзац начинается с дескриптора <P>. Ранние версии HTML не требовали от программистов указывать на конец абзаца с помощью закрывающего дескриптора </P>. Броузерам приходилось эффективно "отключать" инструкции разметки абзацев, чтобы определить, где именно заканчивается тот или иной абзац, вместо того, чтобы просто завершать абзац точно в том месте, которое было указано автором. Последний стандарт, HTML 4.01, требует, чтобы абазац, точно так же как и все дескрипторы, обязательно заканчивался соответствующим закрывающим дескриптором. Однако броузерам все равно приходится иметь дело с миллионами Web-узлов, которые содержат страницы с очень "плохим" HTML-форматированием, а значит, не соответствуют стандарту HTML 4.01.

Вы познакомитесь с прекрасно отформатированными XML-документами в уроке дня 2-го "Подробные сведения о синтаксисе XML", а также узнаете, что синтаксис XML достаточно строг и не допускает каких-либо отклонений. Однако при этом те же самые правила XML обеспечивают такой уровень гибкости для программиста, о котором никогда не могли мечтать разработчики HTML-документов.

Вам следует знать семь фактов об XML

Как вам уже известно, XML — это аббревиатура от eXtensible Markup Language (расширяемый язык разметки). Однако XML можно назвать и расширяемым метаязыком (eXtensible Meta Language), так как он настолько стандартизирован и гибок, что может использоваться при создании других языков. Если говорить точнее, то XML позволяет создавать диалекты языков, которые точно следуют всем правилам, касающимся структуры, синтаксиса и семантики, которые определены консорциумом *W3C*. С некоторыми правилами вы познакомитесь уже в настоящей главе, а важность остальных правил вы поймете, когда прочтете книгу до конца и выполните все представленные в ней упражнения.



Консорциум *W3C* (World Wide Web Consortium) — это официальная организация стандартизации, которая отвечает за разработку различных технологий взаимодействия (спецификаций, руководств, программного обеспечения и инструментов). Эта организация основана Тимом Бернерсом-Ли, признанным изобретателем Web.



Web-узел консорциума *W3C* (`http://www.w3.org`) — один из лучших источников информации по XML и смежных технологиях.

В настоящем разделе вы познакомитесь с семью утверждениями, которые дают полную характеристику XML. На Web-узле консорциума *W3C* вы можете найти описание XML в виде десяти пунктов (`http://www.w3.org/XML/1999/XML-in-10-points`), однако три из этих пунктов относятся к будущим характеристикам. К семи характеристикам XML относятся следующие.

1. XML предлагает метод представления структурированных данных в виде текстового файла.

2. XML похож на HTML.
3. XML понятен как компьютеру, так и человеку.
4. XML образует целое семейство технологий.
5. XML достаточно гибок.
6. XML относительно нов, но у него серьезные корни.
7. XML свободен от лицензионных отчислений, платформно-независим, а также может похвастаться широкой поддержкой.

1. XML предлагает метод представления структурированных данных в виде текстового файла

Если вы подумаете о традиционных программах, используемых для создания, манипулирования и поддержки данных, вы наверняка вспомните, что чаще всего данные хранятся в двоичном формате. Коммерческие версии текстовых процессоров, систем управления базами данных или программы электронных таблиц могут сохранять данные в текстовом формате, однако все они создавались с прицелом на использование бинарного формата, который отличается не только для программных продуктов разного типа, но и для программ одного класса, но от различных компаний. Поэтому обмен данными значительно усложняется, а при использовании собственных форматов данных — вообще невозможен.

Вы, наверняка, уже сталкивались с подобными ситуациями, когда, например, хотели обменяться документом, созданным с помощью определенной версии Microsoft Word, с пользователем Corel WordPerfect. Документ, сохраненный в собственном двоичном формате одного приложения, нельзя прочитать с помощью другого приложения, и наоборот. Для того чтобы обеспечить возможность обмена подобными документами, вам придется выполнить дополнительные действия по сохранению документа в формате, понятном для обоих приложений. Зачастую при этом вам приходится жертвовать частью возможностей форматирования, поддерживаемых одним из продуктов.

XML обеспечивает возможность сохранения данных в обычном текстовом формате. Любой человек или приложение, способное работать с текстовыми документами, способны прочитать XML-документы. Вам не требуется специальное программное обеспечение для того, чтобы получить доступ к подобным данным. Появляется возможность просмотра документа и внесения в него необходимых изменений с помощью простого текстового редактора на любой платформе. В поставку большей части операционных систем входят бесплатные текстовые редакторы. Другие редакторы, предназначенные для выполнения именно такой работы, доступны для загрузки из Internet, причем достаточно часто совершенно бесплатно.

Как я уже сказал, XML позволяет сохранять в текстовых файлах *структурированные* данные. XML — это набор правил для создания текстовых форматов, простых для обработки компьютерами различных типов. Получаемые текстовые файлы структурированы таким образом, что они

- точно выражены;
- расширяемы;
- платформно-независимы.



X в аббревиатуре XML означает Extensible (Расширяемый). Расширяемость означает, что язык можно изменить для обеспечения соответствия определенным требованиям. Поскольку XML не базируется на строго ограниченном наборе дескрипторов, вы можете создавать новые описательные дескрипторы в соответствии со своими требованиями.

Несмотря на то, что для создания XML-документов можно использовать любой самый простой текстовый редактор, вы познакомитесь с несколькими специальными инструментами, изучив материал настоящей книги. Вы сможете использовать эти инструменты при работе над XML-документами.

По общему согласию, для файлов XML характерно расширение `.xml`, например `mydocument.xml`. Специализированные диалекты, созданные в рамках XML, могут использовать другие расширения. Ниже приведено несколько примеров.

- `.xsl`. Файл Extensible Stylesheet Language.
- `.xsd`. Extensible Schema Definition (Определение расширяемой схемы).
- `.xdr`. XML Data Reduced Schema (Сокращенная схема данных XML).
- `.mml`. Файл MathML (Mathematical Markup Language — математический язык разметки).
- `.cdf`. Channel Definition Format (Формат определения каналов).

2. XML похож на HTML

У HTML и XML есть несколько похожих характеристик. Если вы хорошо знакомы с HTML, вы знаете некоторые синтаксические правила, которые в равной мере применимы и к XML. Рассмотрите пример разметки в листинге 1.1. Как вы думаете, это фрагмент кода на HTML или XML?

Листинг 1.1 Пример простой разметки

```
<P>
    Это пример текста, который
    <EM>отмечен</EM> для использования
    в Web-презентации.
</P>
```

Ответ будет положительным в любом случае! На самом деле, это пример хорошо оформленного фрагмента кода на HTML и XML. Он соответствует синтаксическим правилам, общим для обоих языков. Подробно с этими правилами вы познакомитесь в настоящей и следующих главах.

Однако XML разрабатывался для преодоления всех ограничений, свойственных HTML. Автор документа может определить ряд элементов XML; строго фиксированного набора заранее определенных дескрипторов, как в HTML, в XML не существует. Так как вы, как автор документа, можете определять свои собственные дескрипторы, вы может и выбирать для них наиболее описательные имена. Вместо того чтобы размечать абзацы с помощью дескриптора `<P>`, вы можете использовать такие имена дескрипторов, как `<paragraph>` или `<para>`. Либо вы можете вообще использовать для разметки разных абзацев дескрипторы с различными именами. Например, вместо одного имени дескриптора `<paragraph>` вы можете использовать следующие варианты.

- `<introduction>`
- `<summary>`
- `<sales_info>`
- `<address>`
- `<description>`
- `<etc>`

Вы видите, что как в HTML, так и в XML все дескрипторы разметки заключаются в пару знаков < и >. В языках разметки самый простой элемент включает в себя следующее: открывающий дескриптор, содержимое, закрывающий дескриптор, как показано в листинге 1.2.

Листинг 1.2. XML-документ, содержащий всего один элемент

```
1: <title>Освой самостоятельно XML за 21 день</title>
```

Этот элемент образует очень простой, но хорошо оформленный XML-документ. Элемент называется `title`; его имя указывается в открывающем (`<title>`) и закрывающем (`</title>`) дескрипторах. Вся строка целиком, `<title>Освой самостоятельно XML за 21 день</title>`, начиная с первой открывающей угловой скобки и заканчивая последней закрывающей угловой скобкой, называется *элементом*. Текстовые данные, расположенные между дескрипторами, называются содержанием элемента.



Данные, расположенные между дескрипторами, называются *содержанием элемента*, или просто *содержанием*. Однако термин *содержание элемента* также используется и в некоторых языках описаниях схем, о чем вы узнаете дальше в настоящей книге.



Все элементы в XML должны быть завершены. Все непустые элементы должны содержать открывающий и закрывающий дескрипторы. Пустые элементы должны быть должным образом завершены.

Элементы разметки XML выступают в роли строительных блоков языка, являясь аналогами существительных в обычных "человеческих" языках. В XML также существуют и атрибуты, похожие на атрибуты HTML и предназначенные для изменения или классификации элементов, точно так же как прилагательные характеризуют существительные в обычных "человеческих" языках. Атрибуты, если они, конечно же, есть, должны указываться в открывающем дескрипторе элемента. В листинге 1.3 приведен пример немного более сложного элемента, содержащего атрибуты и другие элементы. Если элемент содержит дочерние элементы, те называются *вложенными*.

Листинг 1.3. XML-документ, содержащий вложенные дочерние элементы и атрибуты

```
1: <account type= "checking" currency ="U.S. Dollars">
2:   <name>Smith</name>
3:   <balance>34576.89</balance>
4: </account>
```



Строки с 1 по 4 образуют целый элемент `account`. Два атрибута дополняют элемент `account` точно так же, как прилагательные дополняют существительное, предоставляя новые сведения об элементе. Атрибуты называются `type` и `currency`, а их значения — `checking` и `U.S. Dollars` соответственно. Значения атрибутов всегда заключаются в одинарные или двойные кавычки. Элемент `account` обладает только содержанием (строки 2 и 3). Другими словами, этот элемент не содержит ни одного знака или текстовых данных, а только дочерние элементы, такие как `name` и `balance` (строки 2 и 3). Все элементы, `account`, `name` и `balance`, содержат открывающий и закрывающий дескрипторы. Все элементы XML должны быть завершены.



Атрибуты элементов в XML всегда заключаются в одинарные или двойные кавычки.

Решить, необходимо ли использовать элемент или атрибут для представления определенных сведений, зачастую оказывается не так и просто. Если вы можете представить сведения в качестве прилагательного, которое дополняет значение исходного существительного, скорее всего вы имеете дело с атрибутом. С другой стороны, если сведения обладают собственным значением, их лучше представить в виде элемента. Элементы могут содержать дочерние элементы, а атрибуты — нет. Таким образом, любая единица, которая может содержать другие элементы, является элементом. Вам необходимо тщательно изучать каждый отдельный проект разметки, чтобы определить, какой подход лучше всего в той или иной ситуации.

Элементы XML могут содержать текст, другие элементы, любую комбинацию текста и других элементов или же быть просто пустыми элементами. Пустой элемент — это такой элемент, между открывающим и закрывающим дескрипторами которого отсутствует какое-либо содержание. Синтаксис пустого элемента следующий:

```
<имя_элемента/>
```

Пример пустого элемента XML приведен в листинге 1.4.

Листинг 1.4. Пример пустого элемента XML

```
1: <data month="Сентябрь" year="2002"/>
```

Элемент `data` пустой. Знак завершения (/) указывается прямо в открывающем дескрипторе. Это краткая форма записи следующей инструкции: `<data month="Сентябрь" year="2002"></data>`. Несмотря на то, что обе инструкции функционируют совершенно одинаково, для представления пустого элемента XML принято использовать краткую форму. Даже если такой элемент обладает атрибутами с определенными значениями, он все равно считается пустым, так как между открывающим и закрывающим дескрипторами отсутствуют какие-либо данные.

Обратите внимание на то, что в данном случае элемент обладает атрибутом. У вас может возникнуть вопрос, как считать пустым элемент, если он обладает атрибутами с определенными значениями? Пустым считается любой элемент, который не содержит никаких сведений о разметке. Сведения о разметке (содержимое) должны быть между открывающим и закрывающим дескрипторами. В уроке дня 12-го "Объектная модель документа XML — DOM", вы познакомитесь с моделью DOM (Document Object Model — объектная модель документа), которая обеспечивает программный доступ к отдельным частям XML-документа. Вы узнаете, что данные атрибуты занимают различные части, или "узлы", в структуре, представляемой документом XML.

Вы уже убедились в наличии некоторых синтаксических подобий между HTML и XML; и там, и там речь идет об элементах и атрибутах. Одно из фундаментальных различий состоит в том, что вы как программист можете создавать собственные самоописывающиеся элементы XML. Сравните два примера программного кода, приведенные в листингах 1.5 и 1.6.

Листинг 1.5. Фрагмент программного кода на HTML

```
<HTML>
    <H1>Invoce</H1>
    <P>From: Devan Shepherd</P>
    <P>To: Sally Jones</P>
    <P>Date: 26 July 2001</P>
```

```
    <P>Ammount: $100.00</P>
    <P>Tax: 21%</P
    <P>Total Due: $121.00</P
</HTML>
```

Листинг 1.6. Аналогичный фрагмент программного кода на XML

```
<Invoice>
    <From>Devan Shepherd</From>
    <To>Sally Jones</To>
    <Data year="2001" month="July" day="26" />
    <Amount currency="USD">$100.00</Amount>
    <TextRate>21</TextRate>
    <TotalDue currency="USD">121.00</TotalDue>
</Invoice>
```

Изучив два этих примера, вы должны задать себе следующие вопросы.

- Какой документ имеет более важное значение?
- Какой из примеров содержит больше полезных сведений для приложения обработки?
- Какой из листингов найдет более широкое применение?

Вы можете убедиться в том, что XML с его самоописывающимися элементами и атрибутами, обеспечивает гораздо более широкие возможности разметки. Если приложению потребуются данные из счета-фактуры, фрагмент HTML-кода из листинга 1.5 не предоставит никаких других сведений, кроме нескольких абзацев текста. Для различения содержания одного элемента <P> от другого вам придется добавить некоторую логику программирования, сценариев или сравнения с образцом. С другой стороны, XML-код, приведенный в листинге 1.6, предлагает компоненты данных вместе с их значениями, что-то наподобие несопоставимых полей в одноуровневой базе данных. XML — это язык, ориентированный на метаданные, проявляющий "интеллект" при предоставлении сведений о размечаемых данных.

Давайте рассмотрим структуру разметки документа. Один элемент содержит все остальные элементы в документе. Этот один элемент, находящийся на наивысшем уровне в структуре документа, называется *корневым элементом*. На HTML-странице корневым всегда является элемент `HTML`. В XML-документе вы сами принимаете решение о том, как назвать корневой элемент. В примере программного кода, приведенного в листинге 1.6 , корневой элемент называется `Invoice`.



Хорошо оформленный XML-документ должен содержать один и только один корневой элемент, в котором будут все остальные элементы.

Другое ключевое различие между XML и HTML состоит в том, что HTML смешивает содержание и форматирование в одном потоке разметки. Например, в HTML дескриптор <H1> предназначен для отметки определенной строки данных как заголовка первого уровня, он также сообщает броузеру о том, что все данные, отмеченные этим дескриптором, должны отображаться на экране шрифтом большего размера.

XML основан на предположении, что содержание и внешний вид или стиль должны храниться отдельно от данных разметки. XML полагается исключительно на языки таблиц стилей, такие как CSS (Cascading Style Sheets — каскадные таблицы стилей) или XSL (Extensible Style Sheet Language — расширяемый язык таблиц стилей) при отображении или преобразовании документов из одной структуры к другой. Вы познакомитесь

с использованием технологий CSS и XSL для обработки документов броузерами в уроках дней 14-го "Использование каскадных таблиц стилей CSS" и 15-го "Расширяемый язык таблиц стилей и его объекты форматирования" соответственно. В уроке дня 16-го "Язык преобразований XSLT" вы узнаете об использованиии языка XSL (XSLT) для преобразования XML-документов в HTML для создания сценариев промежуточного уровня для платформно-независимой передачи XML-данных через Internet.

3. XML понятен как компьютеру, так и человеку

Вы уже увидели, что самоописывающиеся элементы обеспечивают интуитивную понятность XML-документов. Сама семантика данных обеспечивает "интеллектуальность", которая представлена в элементах XML наряду с содержанием и значениями атрибутов. Несмотря ни на что, XML остается программным кодом. Он считывается и используется обработчиками XML. В следующей главе вы познакомитесь с одной из разновидностей таких обработчиков, называемой парсером, который интерпретирует XML-документ строка за строкой.

4. XML образует целое семейство технологий

В семейство XML входит целый ряд важных компонентных технологий (табл. 1.1). Здесь вы найдете необходимые пояснения, а также ссылку на день, в котором приводятся подробные сведения о данной технологии.

Таблица 1.1. Технологии XML

День	*Технология*	*Описание*
1-й, 2-й и 3-й	XML Version 1.0	Технические рекомендации об использовании XML
4-й	DTD	Определение типа документа (схема)
5-й	XDR	Формат XML Reduced (схема Microsoft)
6-й	XSD	Определение схемы XML (схема консорциума W3C)
8-й	Пространства имен	Метод определения имен элементов и атрибутов
9-й	XPath	Язык путей XML
10-й	XLink	Язык ссылок XML
11-й	XPointer	Язык указателей XML
12-й	DOM	Программный интерфейс API для объектной модели документов
13-й	SAX	Simple API for XML (Простой программный интерфейс API для XML)
15-й	XSL	Расширяемый язык таблиц стилей
15-й	XSL-FO	Объекты форматирования XSL
16й	XSLT	Язык преобразований XSL
18-й	XInclude	Синтаксис XML Include
18-й	XBase	Синтаксис XML Base URI

По мере того, как вы будете знакомиться с каждой из этих компонентных технологий и экспериментировать со приведенными в книге примерами программного кода, особое внимание следует уделять описанию состояния той или иной технологии, данному консорциумом *W3C*. Некоторые из этих компонентов до сих пор находятся в

черновом варианте, поэтому могут претерпеть значительные изменения в ближайшие месяцы и годы. Обработчики XML поддерживают те или иные члены семейства XML, по крайней мере на момент написания настоящей главы. Должно пройти достаточно много времени, пока пройдет стандартизация основных компонентов XML; кроме того, их должно принять все сообщество программистов на XML. Например, последняя версия языка описания схем, принятая *W3C*, призвана сократить количество описаний схем, которые существуют в настоящее время, что значительно упрощает разработку и поддержку XML-проектов.

5. XML достаточно гибок

Поскольку файлы XML являются простыми текстовыми файлами, содержащими дескрипторы, они практически всегда занимают больше места, чем аналогичные двоичные файлы. Эта характеристика в значительной мере учтена при разработке основных спецификаций XML. Члены консорциума *W3C* приложили немало усилий для того, чтобы сделать XML как можно гибче, что позволило бы обеспечить пространство для расширения и самоописания элементов и их атрибутов.

Разработчики XML прекрасно осознавали, что дисковое пространство, используемое для хранения данных, со временем будет становиться все дешевле и дешевле, поэтому возможные размеры файлов при разработке стандарта особо не учитывались. Кроме того, недорогое, а зачастую и вовсе бесплатное программное обеспечение для сжатия данных широко доступно для использования на многих платформах. Наиболее эффективны такие приложения, как GZIP и ZIP.

И наконец, XML оптимизирован для передачи данных через Internet с помощью протокола HTTP 1.1, который уже по своей природе сжимает текстовые данные буквально на лету, таким образом значительно экономя полосу пропускания.

6. XML относительно нов, но имеет серьезные корни

Стандарт XML впервые представлен в 1996 году и стал стандартом консорциума *W3C* 10 февраля 1998 года. XML основан на языке SGML (Standard Generalized Markup Language — стандартный обобщенный язык разметки), который был разработан на десять лет раньше. SGML — это тоже метаязык, используемый для создания других языков. Одним из приложений SGML и является язык HTML. Вы можете запутаться, пытаясь понять взаимосвязи между этими стандартами, просто запомните, что XML — это подмножество SGML, а HTML — это приложение SGML. XML и HTML — это не стандарты одного уровня. Взаимосвязи между различными стандартами, а также несколько приложений SGML и XML приведены на рис. 1.1.

Вы наверняка уже знаете о существовании языка XHTML (Extended HyperText Markup Language — расширяемый язык разметки гипертекста), а также хотите знать, как он соотносится с другими языками разметки. Строго говоря, XHML — это трактовка HTML как приложения XML, чтобы добиться некоторой гибкости. Консорциум *W3C* предлагает достаточно много подробных сведений на своем Web-узле (`http://www.w3c.org/MarkUp`). В некотором роде XHTML составляет конкуренцию XML как средство представления документов в Web.

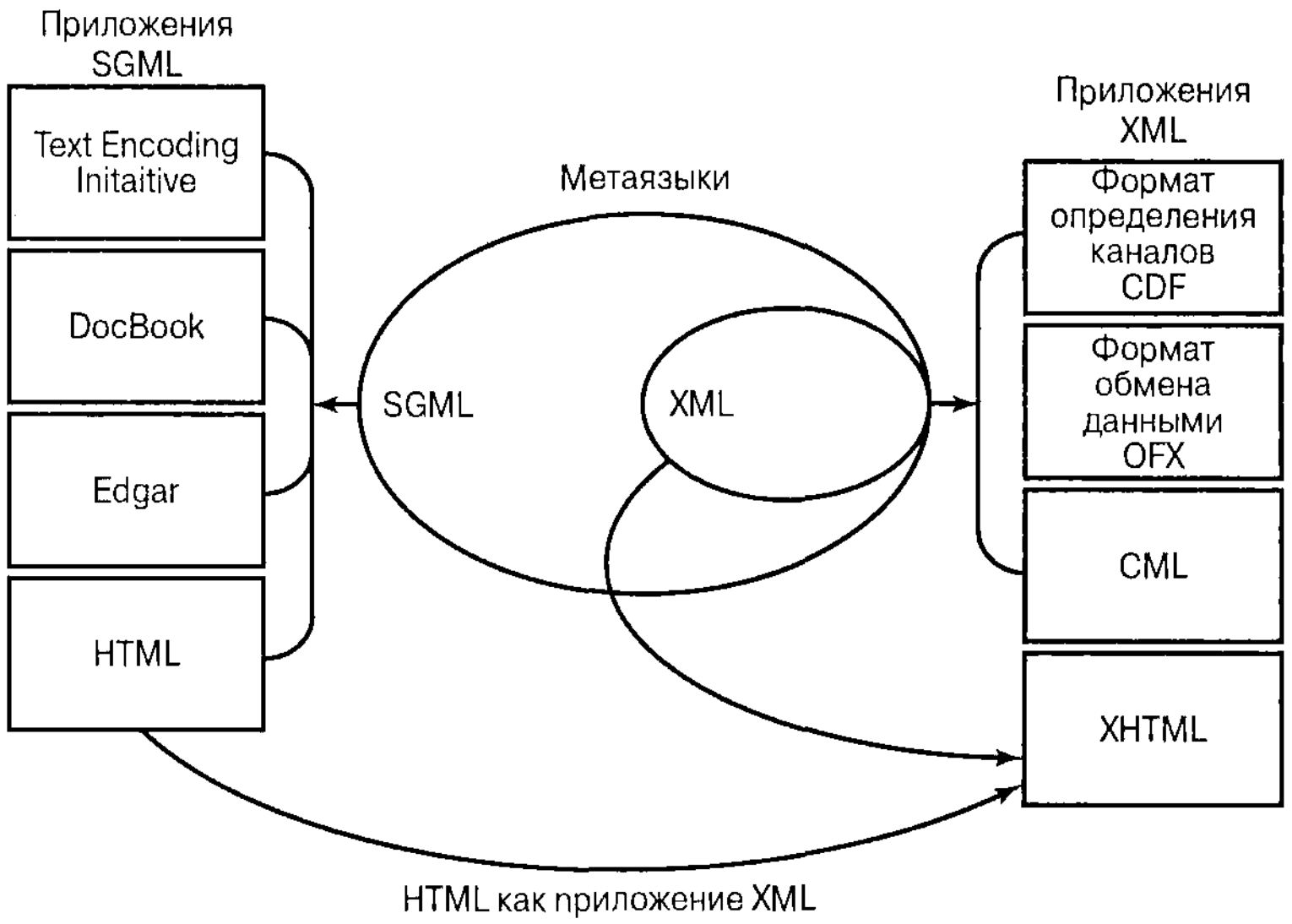


Рис. 1.1. Взаимосвязь между SGML, XML и HTML

7. XML свободен от лицензионных отчислений, платформно-независим, а также может похвастаться широкой поддержкой

Никто не обладает исключительными правами на стандарт XML. Этот стандарт свободен от каких-либо лицензионных отчислений и доступен для всех реализаций. Связанные с ним технологии также доступны на совершенно бесплатной основе.

Платформно-независимая природа XML делает его идеальным вариантом для использования в Web и за ее пределами. Новые модели электронного бизнеса базируются на необходимости "прозрачного" обмена транзакциями через Internet.

Многие из ведущих корпораций, занятых разработкой программного обеспечения, внесли немалый вклад в совместную разработку средств, продвижение стандартов, представление примеров решений, использующих технологии XML. А это означает, что они обеспечивают неплохую поддержку программистов на XML.

Роль электронного бизнеса

Ни у кого не должен вызывать удивления тот факт, что опыт пользователя значительно повышается благодаря мультимедийным методам представления данных, а это особенно актуально в эпоху расцвета Web. Однако решение различных бизнес-задач, связанных с организацией быстрого доступа к финансовым ресурсам и их эффективным представлением, требует ускоренного внедрения средств разработки на базе Web. Действительно, в наши дни способы ведения бизнеса претерпели значительные изменения, в связи с чем необходимость представления данных в электронной форме вышла на первое место.

Когда деловые круги начали поддерживать использование Web в качестве средства ведения торговых операций, требования Web-потребителей стали постоянно расти. Статические Web-страницы уже не в состоянии представлять динамически изменяющиеся данные, уступая место различным моделям представления динамических дан-

ных. Деловому миру необходимы новые инструменты и технологии доставки не только финансовых данных, но и нового способа представления информации все более и более искушенным пользователям Web. Вы уже наверняка знаете о возможностях банковских расчетов или приобретения туристических путевок через Web.

Индивидуализация и персонализация содержимого означает, что Web-узлы способны "узнавать" посетителя и подавать информацию в соответствии с его предпочтениями. Вы наверняка будете неоднократно возвращаться к Web-узлу, который приветствует вас по имени. Вам обязательно понравится Web-узел электронной коммерции, который "помнит", какую именно покупку вы совершили последней, например книгу или музыкальный компакт-диск. Узел может предложить вам список товаров, руководствуясь при его составлении такими критериями, как ваши предыдущие покупки; например, это может быть список книг одного автора или музыкальных альбомов одного жанра или исполнителя. Вы уже могли посещать Web-узлы, предлагающие местные новости, прогноз погоды и другие сведения. Все это примеры персонализации содержимого.

Новые бизнес-модели стараются использовать все доступные технологии доставки данных. Нас интересуют электронные способы доставки данных. Давайте рассмотрим пример с прогнозом погоды. Многие Web-узлы предлагают подобные данные, предварительно запрашивая у вас сведения о индексе или почтовом адресе, а затем сохраняют некоторый идентификационный код в виде так называемого cookie-файла на вашем компьютере. Когда вы загрузите Web-узел во второй раз, программа считывает сведения из cookie-файла и находит соответствующую запись в своей базе данных. После этого на Web-странице представят данные прогноза погоды именно для вашего региона. Все эти данные, как правило, получают от службы, которая предоставляет одни и те же сведения прогноза погоды всем подписчикам.

Модели микроплатежей предлагают еще один пример использования Web-технологий в деловом мире. Например, вы можете посетить online-магазин и приобрести только те главы книги, которые действительно представляют для вас интерес, а не приобретать целиком всю книгу. Обмен информацией, необходимый для проведения подобной транзакции, должен автоматизироваться на Web-узле таким образом, чтобы вы смогли легко сделать выбор. Особенно популярным такой подход стал в музыкальном бизнесе, когда можно создать и приобрести компакт-диск, содержащий только ваши любимые композиции, с помощью всего нескольких щелчков мыши.

Можно привести еще целый ряд примеров электронных транзакций, не имеющих отношения к Web. Давайте рассмотрим электронные транзакции, которые происходят в тот момент, когда вы заправляете автомобиль на заправочной станции и расплачиваетесь с помощью кредитной карточки. В данном случае вы вставляете свою карточку в считывающее устройство которое получает сведения о номере вашего банковского счета и другие сведения с магнитной полосы, после чего все данные передаются специальной службе обработки, которая называется расчетной палатой. Счетная палата проверяет подлинность кредитной карточки и, в случае положительного результата, дает разрешение на проведение покупки. После проверки подлинности карточки происходит транзакция приема, которая дает "приказ" на включение колонки. В то время пока вы заливаете бензин в бак своего автомобиля, между платежными системами продавца и вашего банка продолжается обмен транзакциями, проводятся необходимые расчеты, печатается счет, когда колонка выключается.

Читая настоящую книгу и выполняя примеры, вы быстро поймете, что XML идеально подходит для создания решений таких и других подобных задач бизнеса. Вы научитесь манипулировать технологиями XML, создавать документы, используя свои собственные самоописывающиеся элементы, а также исследуя парсеры и интерфейсы приложений. Вы также узнаете, как можно преобразовать разметку с одного языка на другой, использовать XML совместно с базами данных и обработкой данных на стороне сервера, объединять несколько технологий для создания законченных решений. Книга *Освой самостоятельно XML за 21 день, 2-е издание*, предоставит вам все необходимые сведения для освоения всех этих мощных современных информационных технологий.

Создание вашего первого документа XML

Вы уже увидели, что XML состоит из интуитивно понятных самописывающихся элементов, которые содержатся в одном корневом элементе. Вам необходимо овладеть всего несколькими правилами, о которых мы поговорим в следующей главе. Однако уже сейчас вы обладаете необходимыми сведениями для создания своего первого документа XML. Кроме того, для выполнения упражнения вам потребуются следующие инструменты.

- Текстовый редактор (Блокнот, SimpleText и т.д.) Вы можете использовать любой редактор, который способен сохранять текстовые документы ASCII, но если вы используете текстовый процессор, ни в коем случае не сохраняйте данные в двоичном формате.
- Броузер Microsoft Internet Explorer (версии 5.0 или более поздней). Если на вашем компьютере еще не установлена соответствующая версия этого броузера, загрузите ее с Web-узла `http://www.microsoft.com/downloads/searech.asp`.

Документ, который вы создадите, будет очень простым: в нем будут представлены такие сведения, как ваши имя и фамилия, а также шутка. Пусть имени соответствует элемент `<first>`, а фамилии — `<last>`. Шуткой может быть любая текстовая строка, указанная в элементе `<joke>`. Начните с запуска текстового редактора и приступите к созданию документа XML. Не забывайте о том, что в документе может содержаться всего один корневой элемент и по одному элементу для каждого отдельного компонента данных. Вы должны выбрать наиболее подходящий вариант для имени корневого элемента. Вам также необходимо завершить каждый элемент; в нашем случае для этого используются закрывающие дескрипторы.

Закончив создание документа, сохраните его под именем `me.xml`, после чего запустите Internet Explorer для просмотра результата. Если вы сохраните файл с другим расширением, вы не сможете увидеть нужного результата.

Для того чтобы увидеть результат в Internet Explorer, введите полный путь в адресной строке, расположенной в верхней части окна броузера. При просмотре документа с помощью Internet Explorer документ проходит через специальный обработчик, который называется *парсером*. Парсер проверяет ваш документ на соответствие всем синтаксическим правилам XML. Если разметка вашего документа правильна, парсер преобразует ваш документ таким образом, чтобы его было легко читать, отображая дескрипторы разметки цветом, отличным от цвета заключенного между ними текста. Подобный процесс преобразования называется *преобразованием расширяемого языка таблиц стилей* XSLT. Подробные сведения об XSLT вы найдете в уроке дня 16-го "Язык преобразований XSLT".

В листинге 1.7 приведен пример простого документа; ваш вариант может отличаться другими именами элементов, другой структурой вложений или более удачной шуткой.

Листинг 1.7. Первый документ XML

```
<data>
     <name>
               <first>Devan</first>
               <last>Shepherd</last>
     </name>
     <joke>ASCII silly question, get a silly ANSI</joke>
</data>
```

Результат отображения этого документа XML в окне броузера приведен на рис. 1.2. Обратите внимание, что в случае правильно оформленного документа XML Internet Explorer преобразует документ, применяя к нему определенные стили.

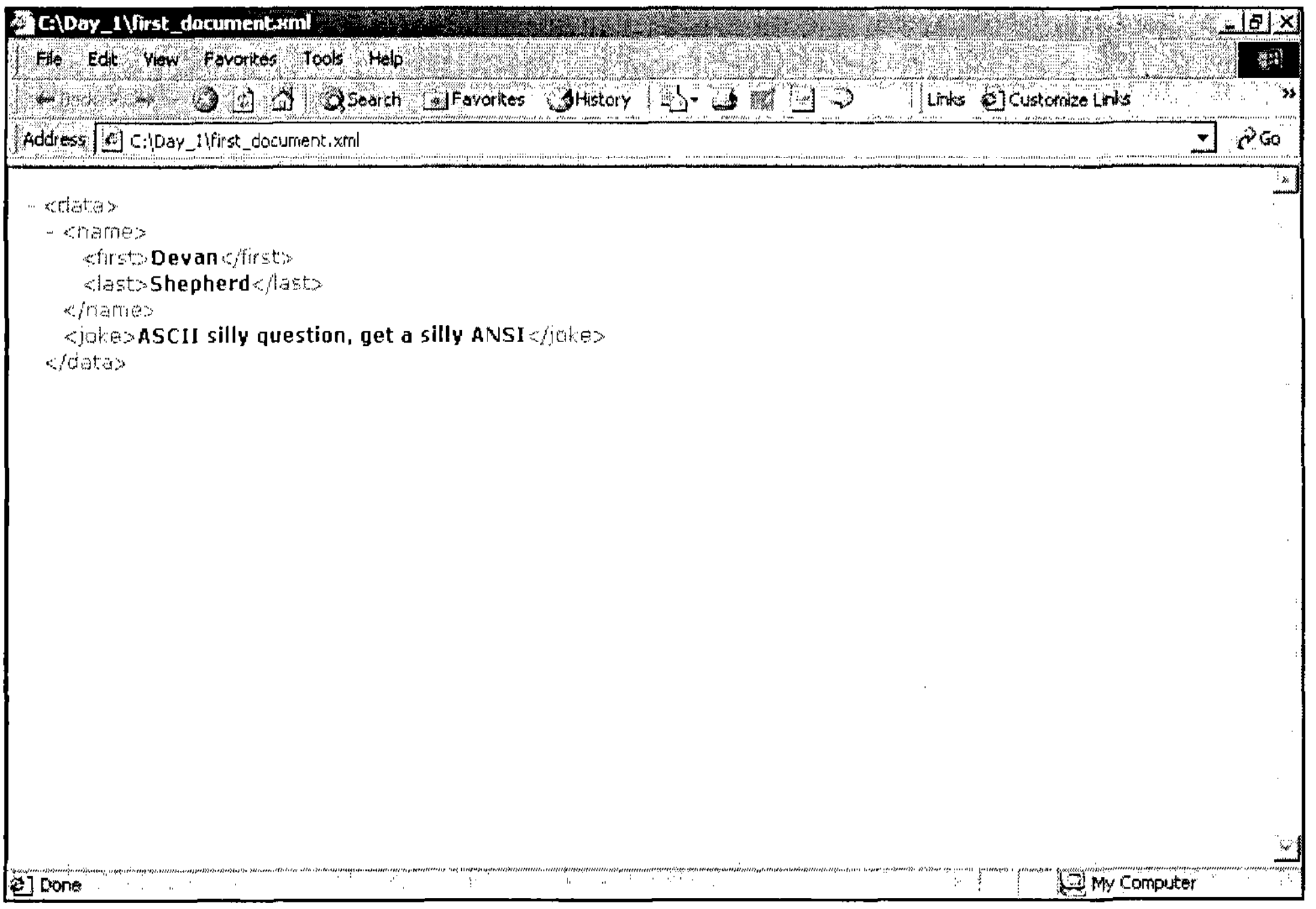


Рис. 1.2. Результат выполнения кода из листинга 1.7

Как уже отмечалось раньше, броузер Internet Explorer отображает имена элементов одним цветом, а их содержание — другим. Это происходит благодаря преобразования XML в HTML-страницу, хранящуюся в памяти с соответствующей таблицей стилей. Таблицы стилей поставляются вместе с Internet Explorer. Кроме того, броузер содержит встроенный обработчик XML-кода, который называется *парсером*. Более подробные сведения о парсерах вы найдете в следующей главе.

Резюме

Сегодня вы познакомились с XML — метаязыком, используемым для создания других языков. Вы узнали о том, насколько важна разметка в мире электронной коммерции, каким образом деловое сообщество содействует быстрому развитию и внедрению Web-технологий. Вы узнали семь основных характеристик XML, а также о том, что всего несколько правил способны образовать стандарт. И закончили вы созданием своего первого документа XML, обработав его с помощью Web-броузера Internet Explorer компании *Microsoft*. Завтра, в следующем уроке, вы подробнее познакомитесь с хорошо оформленным XML и с теми преимуществами, которые дает вам XML как Web-программисту.

Вопросы и ответы

Какова роль делового мира в развитии Web-технологий?

Деловые круги стимулируют развитие новых технологий, не только обеспечивая необходимыми финансовыми средствами, но и создавая новые модели электронного бизнеса и обмена данными.

Зачем нужен еще один язык разметки?

Строго говоря, XML — это не сколько язык разметки, сколько средство, позволяющее программистам выйти за рамки ограничений, накладываемых HTML.

А что же позволяет делать HTML?

Этот язык разметки позволяет создавать замечательные гиперссылки. Он очень прост в изучении и широко используется в Web. HTML поддерживается практически всеми существующими на рынке Web-броузерами.

Можно ли просматривать документы XML с помощью Netscape или другого броузера?

Netscape 6.0 — это броузер, который обладает поддержкой документов XML. Вы можете просматривать документы XML с помощью Netscape, но имейте в виду, что они будут выглядеть несколько иначе, чем при просмотре с помощью Internet Explorer. На самом деле, вам даже следует просматривать свои документы с помощью нескольких броузеров, чтобы увидеть различия между ними. Запомните, что Internet Explorer обеспечивает преобразование с XML на HTML, применяя к вашему документу различные стили и специальные знаки разметки. Преобразование осуществляется с помощью языка XSLT с использованием стандартной таблицы стилей, встроенной в Internet Explorer. Подробные сведения об XSLT вы найдете в уроке дня 16-го "Язык преобразований XSLT".

Почему языка HTML оказывается недостаточно

Языку HTML недостает гибкости, что связано с ограниченным набором заранее определенных дескрипторов. Его нельзя легко расширить, и он страдает от несколько "негибких" стилей.

Назовите семь основных характеристик XML.

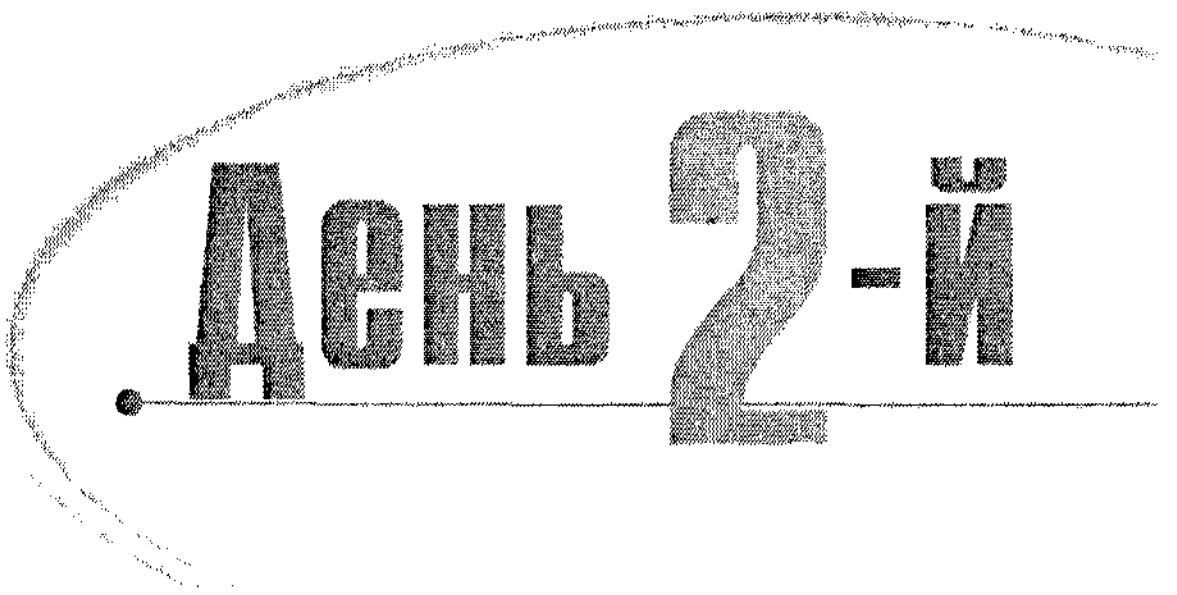
К основным характеристикам XML относятся следующие.

1. XML предлагает метод представления структурированных данных в виде текстового файла.
2. XML похож на HTML.
3. XML понятен как компьютеру, так и человеку.
4. XML образует целое семейство технологий.
5. XML достаточно гибок.
6. XML относительно нов, но у него серьезные корни.
7. XML свободен от лицензионных отчислений, платформно-независим, а также может похвастаться широкой поддержкой.

Упражнение

Проверьте свои знания, полученные в результате изучения настоящего урока. Ответ вы найдете в Приложении А "Ответы на вопросы".

Создайте документ XML, в котором будут представлены такие сведения, как ваши имя, домаший адрес, адрес электронной почты, дата рождения. При указании даты используйте атрибуты. Завершив создание документа, сохраните его и просмотрите с помощью Microsoft Internet Explorer 5.0.



Подробные сведения о синтаксисе XML

В первом уроке вы узнали, что такое XML и на чем он базируется. Сейчас вы познакомитесь с основными правилами построения документов XML.

Сегодня вы изучите следующие темы.

- Синтаксис хорошо оформленного документа XML.
- Методы проверки форматирования документа.

Создание собственных элементов XML

Вы уже видели несколько примеров документов XML. Документы XML состоят из элементов и атрибутов, имена которых вы можете указывать самостоятельно. Сегодня мы рассмотрим все эти конструкции поподробнее, создадим еще несколько документов XML. Вы начнете с создания так называемого "хорошо оформленного" XML.

ПРИМЕЧАНИЕ

Термин *экземпляр XML* является полным синонимом термина *документ XML*. В настоящей книге я буду в равной степени использовать оба эти термина. Действительно, документ — это *экземпляр* используемого языка разметки. Для большей понятности, говоря об HTML-коде, я буду использовать термин *HTML-страница* вместо терминов *HTML-документ* или *экземпляр HTML*.

Документ XML считается хорошо оформленным, если он соответствует всем синтаксическим правилам XML. С некоторыми из этих правил вы познакомились в первый день; с другими правилами вы познакомитесь сейчас. Если хотя бы одно из синтаксических правил нарушается, документ не считается хорошо оформленным, а значит, его не будут использовать обработчики XML. В действительности обработчик XML сообщит об ошибке, что во многих случаях позволяет вам исправить ошибку, указав на ее источник.

Программные средства для работы с XML

Поскольку документы XML — это просто текстовые файлы, вы можете создавать и изменять их с помощью практически любого самого простого текетового редактора. Существует целый ряд более сложных редакторов, которые способны проверять синтаксис документов при вводе кода; некоторые из таких редакторов доступны для бесплатной загрузки

из Internet. К подобным редакторам относятся такие редакторы, как Architag X-Ray editor (`http://www.architag.com/xray`). Существуют и очень хорошие коммерческие редакторы, такие как XML-Spy от компании *Altova* (`http://www.xmlspy.com`) или XML Authority editor от компании *Tibco Extensibility* (`http://www.extensibility.com`). Достаточно обширный список соответствующих средств вы найдете в Internet (например, по адресу `http://www.oasis-open.org/cover/publivSW.html`). Хорошие редакторы XML-кода существуют для любой операционной системы или аппаратной платформы. Однако для изучения материала настоящей книги и выполнения представленных в ней упражнений вам достаточно простого текстового редактора.

Кроме текстового редактора, при выполнении упражнений вам потребуется загрузить полученный документ XML в *парсер* для проверки на наличие ошибок. *Парсер* — это программное обеспечение, которое проверяет соблюдение синтаксических правил XML и сообщает обо всех обнаруженных ошибках при загрузке документа в память. Кроме того, желательно, чтобы на вашем компьютере был установлен броузер Microsoft Internet Explorer версии 5.0 или более поздней. Этот броузер содержит встроенный парсер XML. Версии Microsoft Internet Explorer существуют для таких операционных систем, как Windows, UNIX и Mac OS. Загрузить копию броузера Microsoft Internet Explorer вы сможете с Web-узла компании *Microsoft* `http://www.microsoft.com`.

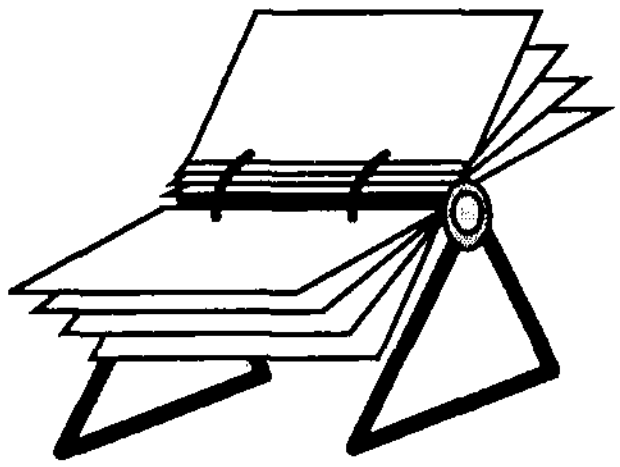
Зависимость от платформы, программного обеспечения и операционной системы

К сожалению, несмотря на то, что при создании XML стремились к независимости от платформы, программного обеспечения и операционной системы, многие из полезных инструментов и обработчиков доступны далеко не для всех платформ или операционных систем. Например, некоторые из приведенных в настоящей книге примеров программного кода будут работать только на платформе *Microsoft* при использовании броузера Netscape или конкретной реализации парсера XML. Это означает, что вам придется кое-что выбирать при создании решений. Например, Web-разработчикам не обойтись без учета специфических черт того или иного броузера, что значительно ограничивает выбор инструментов, которые могут использоваться. Ведь пользователи, независимо от того, с каким броузерам они работают, должны получить доступ к данным Web-узла. В настоящей книге я буду по возможности учитывать особенности различных решений, платформ и операционных систем. Иногда приведенные примеры будут рассчитаны на одну платформу, например платформу *Microsoft*, хотя основные концепции применимы и к другим платформам. Например, в уроке дня 12-го вы узнаете, как создать программу на JavaScript для доступа к такому интерфейсу API компании *Microsoft*, как DOM (Document Object Model — объектная модель документа). Эта модель компании *Microsoft* обладает очень широкой поддержкой и наибольшей инсталлированной базой. Однако, если вам необходимо использовать другое решение, вы найдете варианты DOM, выполненные на PERL, Java, Python и некоторых других популярных языках для поддержки широкого спектра платформ. В подобных случаях я привожу ссылки на Web-узлы, на которых вы найдете сведения о соответствующих инструментах.

Синтаксические правила XML

В этом разделе вы познакомитесь с некоторыми синтаксическими правилами XML, а также увидите, каким образом парсер обрабатывает документы XML. Для того чтобы извлечь максимум пользы из этих сведений, вы можете создать несколько документом и обработать их с помощью парсера.

Запустите свой любимый текстовый редактор и создайте в нем документ XML. В этом документе XML будет рецепт приготовления одного очень интересного блюда. Начните с определения структуры типичного рецепта. Пример рецепта приведен на рис. 2.1. В рецепте обязательно должен быть заголовок, по которому его легко отличить от других рецептов. Список ингредиентов, а также количество каждого из них — это очень важные сведения, точно так же как инструкции по приготовлению. Также включите в рецепт несколько важных рекомендаций.



Смешать говяжий фарш, соль, сухой измельченный эстрагон, измельченный в порошок сухой лук и 3 столовые ложки томатного соуса.
Полученную массу разделить на 4 лепешки.
Уложить в посуду с бортами и готовить при *полной мощности 2 минуты. Затем* положить сверху грибы, сыр и залить оставшимся соусом.
Довести до готовности за 2-3 минуты.

Рис. 2.1. Рецепт приготовления блюда

Если вы создаете файл XML на основе существующего документа, вам необходимо очень внимательно изучить оригинал, чтобы понять структуру экземпляра для разметки. Этот процесс называется анализом документа и является важным компонентом процесса разметки. Для разметки рецепта приготовления блюда анализ рецепта будет совсем несложным. Начните с идентификации компонентов рецепта. Вы уже знаете, что основные строительные блоки называются элементами. Элементы чем-то сродни существительным в обычных языках; это контейнеры информации, которые содержат данные или другие элементы. Дополнительные сведения об элементах вы найдете дальше в настоящем уроке.

Для того чтобы разобраться, какие именно элементы должны быть в рецепте, вы должны разобраться со списком необходимых дескрипторов элементов. Для каждого основного компонента рецепта вам необходимо определить следующее.

- Как должен называться элемент?
- Где должен начинаться элемент (а значит, где должен находиться его открывающий дескриптор)?
- Где должен заканчиваться элемент (а значит, где должен находиться его закрывающий дескриптор)?
- Какие именно сведения должен содержать элемент?
- Какова взаимосвязь между элементом и другими элементами рецепта?

На рис. 2.2 приведен типичный рецепт в виде обычной блок-схемы. Как видно из этой блок-схемы, элемент `recipe` содержит такие элементы, как `title`, `ingredients`, `preparation` и `serving`. Элемент `ingredients` содержит вложенные элементы `item` и соответствующие им дочерние элементы `quantity`.

Еще один полезный пример документа XML приведен на рис. 2.3. Здесь рецепт представлен в виде древовидной структуры.

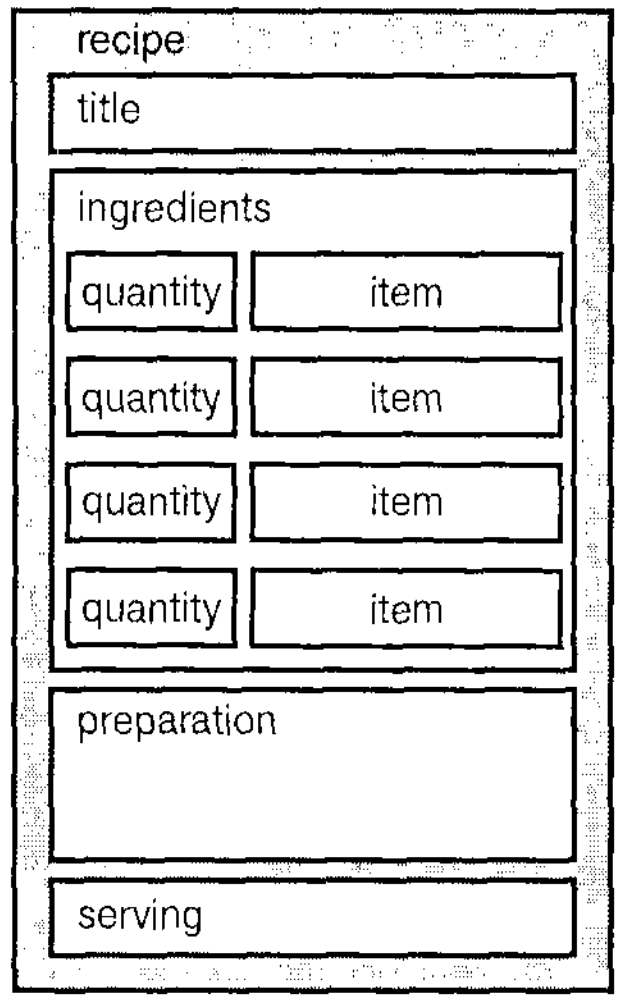


Рис. 2.2. Блок-схема рецепта приготовления блюда

ПРИМЕЧАНИЕ

В уроке дня 12-го "Объектная модель документа XML — DOM" вы познакомитесь с таким интерфейсом API, как DOM for XML (Объектная модель документа для XML). Этот прикладной интерфейс взаимодействует со всеми экземплярами XML как с узлами древовидной структуры, позволяя программисту легко добавлять, удалять и изменять узлы и их содержимое.

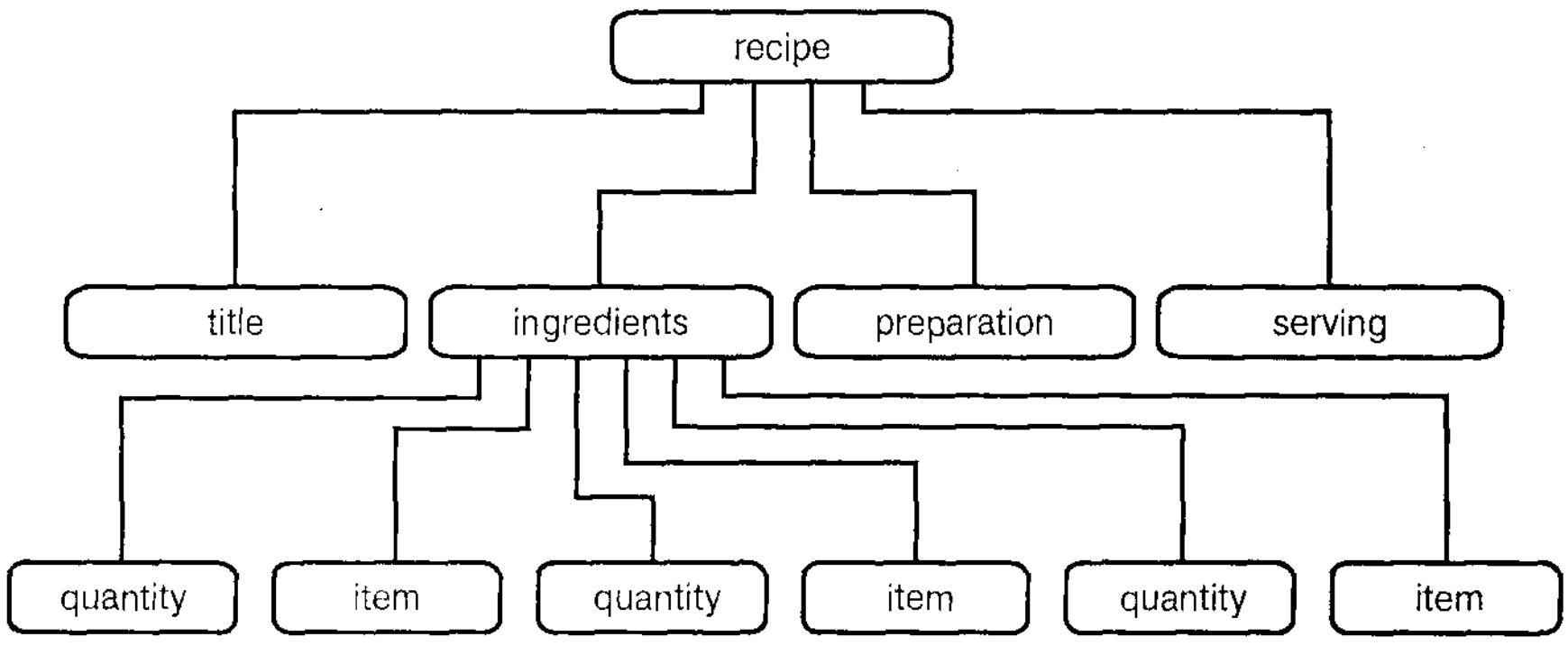


Рис. 2.3. Древовидная структура рецепта приготовления блюда

Из показанной на рисунке древовидной структуры можно увидеть взаимосвязи между элементами. В любом случае мы имеем дело только с одним корневым элементом `recipe`, который выступает в роли контейнера для всех остальных элементов. Близкое рассмотрение элемента `ingredients` позволяет увидеть, что он содержит вложенные элементы `item` и соответствующие им дочерние элементы `quantity`. Для того чтобы экземпляр XML считался хорошо оформленным, дочерние элементы и подэлементы корневого элементы должны быть вложены без взаимного "перекрытия" дескрипторов. Сегодня вы познакомитесь с очень простым способом проверки правильной организации вложения элементов, однако сначала вам следует создать документ XML, введя приведенный код с помощью текстового редактора.



> Хорошо оформленный экземпляр XML должен содержать один и только один корневой элемент, содержащий все остальные элементы. Элементы в корневом элементе могут содержать дочерние элементы, которые должны быть правильно вложены. Все элементы, содержащиеся в корневом элементе, являются дочерними по отношению к нему.

1. Начните с создания корневого элемента, назвав его `recipe`. Он должен начинаться с открывающего дескриптора `<recipe>` и заканчиваться закрывающим дескриптором `</recipe>`.
2. Корневой элемент `recipe` будет содержать все остальные элементы. Укажите открывающий и закрывающий дескрипторы для всех дочерних элементов корневого элемента `recipe`, которые называются `title`, `ingredients`, `preparation` и `serving`.
3. Затем укажите открывающий и закрывающий дескрипторы для всех пар дочерних элементов `item` и `quantity` в рамках открывающего и закрывающего дескрипторов элемента `ingredients`.
4. Расставив по местам все необходимые открывающие и закрывающие дескрипторы, заполните их необходимыми данными, приведенными раньше на рис. 2.1.
5. Сохраните полученный документ под именем `bean_dip.xml`. Он должен иметь такой вид, как показано в листинге 2.1. При просмотре документа с помощью Internet Explorer вы должны увидеть результат, приведенный на рис. 2.4.

Листинг 2.1. Документ XML с рецептом `bean_dip.xml`

```
1: <recipe>
2:  <title>Devan's Bean Dip</title>
3:  <ingredients>
4:      <quantity>1 can</quantity>
5:      <item>refried beans</item>
6:      <quantity>1 can</quantity>
7:      <item>burrito sauce</item>
8:      <quantity>8 oz.</quantity>
9:      <item>cubed Jalapeno cheese</item>
10:     <quantity>1/2 cup</quantity>
11:     <item>sour cream</item>
12: </ingredients>
13: <preparation>In medium sauce pan, heat bean
14:     and burrito sauce until bubbly. Add cubed
15:     cheese and stir until melted. Remove from
16:     heat and stir in sour cream.</preparation>
17: <serving>Serve as a dip with corn
18: tortilla chips</serving>
19: </recipe>
```

В первом уроке настоящей книги вы познакомились с составными частями элемента: открывающим и закрывающим дескрипторами, а также содержанием. Дескрипторы XML чувствительны к регистру символов, в отличие от дескрипторов HTML; таким образом, открывающий и закрывающий дескрипторы должны быть указаны с использованием одного и того же регистра символов. В некоторых экземплярах элемент пустой, а значит, между его открывающим и закрывающим дескрипторами нет никакого содержания. Угловая скобка (<) в начале дескриптора в языках разметки называется MDO (Markup Declaration Open — Открытие декларации разметки). Угловая скобка (>) в конце дескриптора в языках разметки называется MDC (Markup Declaration Close — Закрытие декларации разметки). В качестве примера давайте рассмотрим следующий элемент:

```
<item>sour cream</item>
```

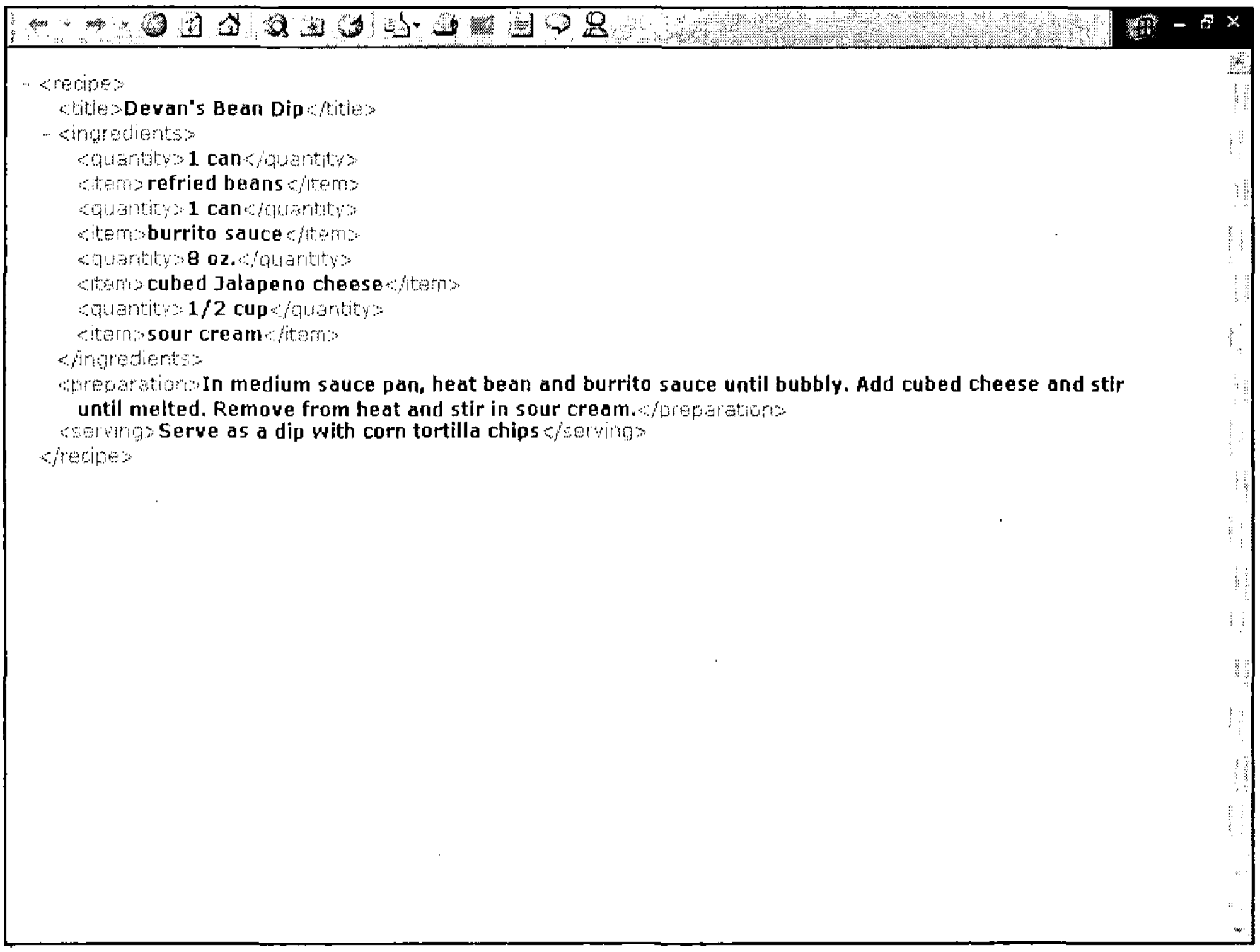


Рис. 2.4. Рецепт приготовления блюда, представленный в виде хорошо оформленного документа XML, создан с помощью простого текстового редактора и отображен в окне Internet Explorer

Отдельные части этого элемента описаны в табл. 2.1.

Таблица 2.1. Анатомия элемента `item`

Элемент разметки	*Описание*
<	Открытие декларации разметки (MDO)
item	Имя элемента
sour cream	Содержание элемента
</	Завершение элемента
item	Имя элемента, который должен быть завершен
>	Закрытие декларации разметки (MDC)

Отступы перед строками, как вы могли видеть в листинге 2.1, считаются среди программистов правилом хорошего тона. Используя отступы перед дочерними элементами, вы упрощаете определение уровней вложения в документе XML. Добавление свободного пространства значительная часть обработчиков XML игнорирует, но это не приводит к возникновению каких-либо проблем.



Использование свободного пространства в документе XML нормализовано. Однако, в отличие от HTML, XML передает знаки "свободного пространства" обработчику, который должен принять решение о том, как именно они должны обрабатываться.

Как вы уже узнали из урока дня 1-го, броузер Microsoft Internet Explorer (версии 5.0 или более поздней) позволяет просматривать документ XML на экране, используя разные цвета для представления имен элементов и их содержания. Это объясняется тем, что в состав Internet Explorer входит специальный обработчик XML, *парсер*; он проверяет XML-код, чтобы определить, правильно ли оформлен последний. Если все правильно, парсер связывает с экземпляром XML встроенную таблицу стилей, а потом преобразует документ с помощью XSLT. Подробные сведения об XSLT вы найдете в уроке дня 16-го "Язык преобразований XSLT".

Запустите Internet Explorer и укажите ему месторасположение созданного раньше файла `bean_dip.xml`. Вы увидите результат, приведенный на рис. 2.5.

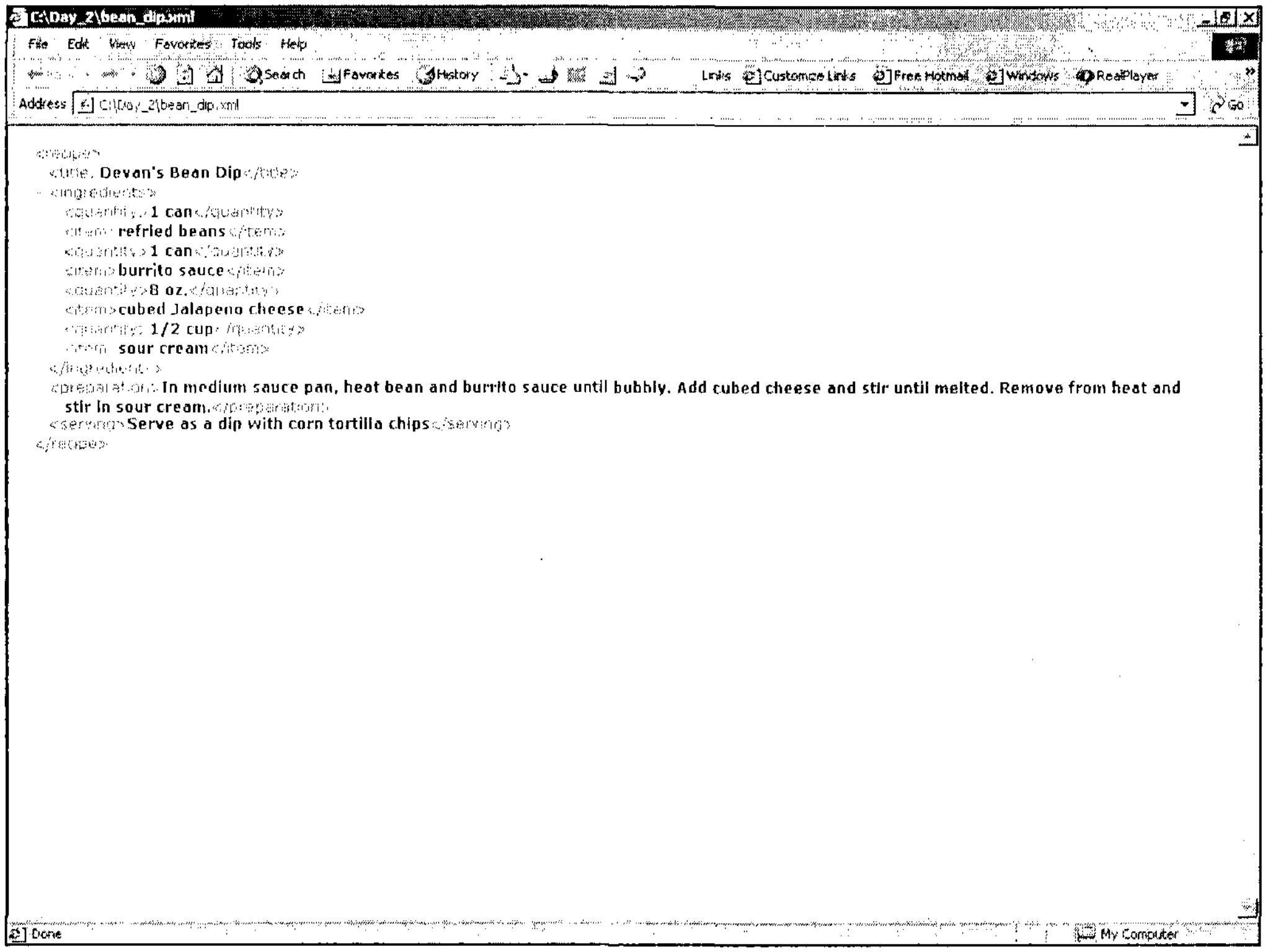


Рис. 2.5. Обработчик MSXML, встроенный в броузер Microsoft Internet Explorer, создал древовидную структуру для экземпляра `bean_dip.xml`

Установив броузер Microsoft Internet Explorer на компьютер, вы сможете использовать реализацию парсера XML, выполненную компанией *Microsoft*. Парсер XML — это программа, которая отслеживает выполнение синтаксических правил, определенных версией 1.0 стандарта XML, принятого консорциумом *W3C*. Парсер проверяет документ XML на предмет его соответствия всем этим стандартам. При обнаружении несоответствия отображается сообщение об ошибке, после чего обработка документа прекращается.

Все парсеры XML способны определить, хорошо ли оформлен документ. Завтра, в следующем уроке, вы узнаете, что парсеры, кроме того, способны определить, является ли документ *действительным*, а не только хорошо оформленным. Проверка действительности означает дополнительный набор действий, которые позволяют определить, что программист создал схему, задающую явную структуру документа XML. Таким образом, программист получает возможность определить порядок следования элементов в документе XML, наличие необходимых элементов и, в некоторых случаях, подтвердить тип данных для содержания элементов. Дополнительные сведения вы найдете в следующем уроке!

Правильное вложение элементов

Вы должны обязательно проверять все свои документы XML на предмет того, что при использовании вложенных элементов не было путаницы с порядком следования открывающих и закрывающих дескрипторов. В противном случае документ не будет иметь смысла с точки зрения парсера XML. HTML подобных недостатков лишен. Например, с точки зрения современных броузеров приведенные ниже инструкции приведут к одному и тому же результату:

```
<B><I>Этот текст отображается с использованием полужирного курсивного
начертания</I></B>

<B><I>Этот текст отображается с использованием полужирного курсивного
начертания</B></I>
```

Первый фрагмент кода сконструирован совершенно правильно. Элемент <I> ... </I> является дочерним по отношению к элементу <B>...</B>. В втором фрагменте дескриптор </B> указан перед дескриптором </I>, а это неправильно. Если вы поместите одну строку между дескрипторами <B> и </B>, а вторую — между дескрипторами <I> и </I>, они будут пересекаться, а значит, правила вложения нарушены.



> Всегда ставьте строки между открывающим и закрывающим дескрипторами элемента, тогда никогда не получите наложения строк.

На рис. 2.6 показан экземпляр `bean_dip.xml`, в котором строки заключены между открывающим и закрывающим дескрипторами для всех элементов. Ни одна из строк не пересекается с другой, что свидетельствует о правильном вложении.

Соглашения о присвоении имен в XML

Имена, используемые в документе `recipe`, просты и достаточно описательны. Несмотря на то, что XML считывается компьютером, он должен быть понятен и человеку. Поэтому вам следует выполнять несколько правил при определении имен элементов документе XML.

- Имя элемента XML может начинаться с буквы, знака подчеркивания (_) или двоеточия (:).
- После первого символа в имени элемента могут быть буквы, цифры, знаки переноса (-), знаки подчеркивания (_), точка (.) или двоеточие (:).

- Имена элементов не могут начинаться с буквосочетания XML или вариаций на эту тему, поскольку все подобные имена защищены правами на интеллектуальную собственность консорциума *W3C*.

```
<recipe>
        <title>Devan's Bean Dip</title>
        <ingredients>
                <quantity>1 can</quantity>
                <item>refried beans</item>
                <quantity>1 can</quantity>
                <item>burrito sauce</item>
                <quantity>8 oz.</quantity>
                <item>cubed Jalapeno cheese</item>
                <quantity>1/2 cup</quantity>
                <item>sour cream</item>
        </ingredients>
        <preparation>In medium sauce pan, heat bean
                and burrito sauce until bubbly.  Add cubed
                cheese and stir until melted. Remove from
                heat and stir in sour cream.</preparation>
        <serving>Serve as a dip with corn
                tortilla chips</serving>
</recipe>
```

Рис. 2.6. Проверка на правильность вложения, примененная к экземпляру `bean_dip.xml`

Декларации XML

Вы уже создали несколько простых документов XML, которые хорошо оформлены, а значит, их может обработать парсер из состава Internet Explorer 5.0. Хотя Internet Explorer прекрасно определяет XML, хорошим тоном считается явно указывать, что это документ XML. Некоторые парсеры *требуют* наличия в документе соответствующей инструкции, которая называется декларацией XML.

Добавьте декларацию XML в начало документа `bean_dip.xml` прямо перед открывающим дескриптором элемента `recipe`. Она выглядит следующим образом:

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

Полный код `bean_dip.xml` должен выглядеть так, как показано в листинге 2.2. Обратите внимание на то, что первой строкой в документе является декларация XML. Внесите необходимые исправления в разметку рецепта и сохраните полученный результат под именем `bean_dip2.xml`.

Листинг 2.2. Декларация XML bean_dip.xml

```
1:  <?xml version="1.0" ?>
2:  <recipe>
3:   <title>Devan's Bean Dip</title>
4:   <ingredients>
5:        <quantity>1 can</quantity>
6:        <item>refried beans</item>
7:        <quantity>1 can</quantity>
8:        <item>burrito sauce</item>
9:        <quantity>8 oz.</quantity>
10:       <item>cubed Jalapeno cheese</item>
11:       <quantity>1/2 cup</quantity>
12:       <item>sour cream</item>
13:  </ingredients>
14:  <preparation>In medium sauce pan, heat bean
15:       and burrito sauce until bubbly. Add cubed
16:       cheese and stir until melted. Remove from
17:       heat and stir in sour cream.</preparation>
18:  <serving>Serve as a dip with corn
19:  tortilla chips</serving>
20:  </recipe>
```

Декларация XML в первой строке указывает обработчику на то, что документ XML. Несмотря на то, что некоторые обработчики не требуют наличия такой декларации, ее включение в код документа считается хорошим тоном.

Синтаксис декларации XML несколько отличается от синтаксиса элемента. Декларация — это инструкция обработки (Processing Instruction — PI), которая передает специальные сведения обработчику, сообщая о том, что документ составлен в соответствии с правилами, определяемыми версией 1.0 стандарта XML. Подобная инструкция обработки всегда указывается в первой строке документа. С другими инструкциями обработки вы познакомитесь дальше в настоящей книге. Основные части декларации XML описаны в табл. 2.2.

Таблица 2.2. Анатомия декларации XML

Элемент разметки	*Описание*
<?	Открытие декларации разметки (MDO)
version="1.0"	Сведения о требуемой версии стандарта. На сегодня существует только одна версия XML — 1.0. Поэтому значение атрибута всегда равно `1.0`
encoding=""	Необязательное указание кодировки символов
standalone=""	Необязательный атрибут `standalone`, значение которого может быть `"yes"` или `"no"`
?>	Закрытие декларации разметки (MDC)

Для того чтобы инструкции обработки были понятны парсерам XML, в них должны использоваться только символы нижнего регистра. Кроме того, в любой декларации XML указано, что должна использоваться версия 1.0 стандарта XML. Это связано с тем, что на сегодня это единственная существующая версия стандарта. Возможно, в будущем эта ситуация изменится, поэтому указывать версию стандарта нужно обязательно. Атрибут `encoding`, используемый для указания нужной кодировки символов, не может принимать значения `UTF-8` или `UTF-16`.

СОВЕТ

UTF-8 и UTF-16 — это стандартные схемы кодировки символов, используемые при представлении данных. Обсуждение различных схем кодировки выходит за рамки рассмотрения настоящей книги. Подробные сведения о кодировках вы найдете на следующих Web-узлах: `http://www.w3.org` и `http://msdn.microsoft.com/xml/articles/xmlencodings.asp`.

Атрибут `standalone` указывает обработчику XML, необходимо ли использовать внешнее определение типа документа DTD при проверке экземпляра. Подробно о проверке экземпляров XML мы поговорим в уроке дня 3-го "Правильные экземпляры XML". В большинстве случае хорошо оформленные документы XML не требуют использования определений DTD.

Добавление комментариев к коду XML

Как и другие языки программирования, XML позволяет добавлять к разметке документа различные комментарии. Однако в XML комментарии могут предназначаться и для обработчиков XML. Другими словами, как вы узнаете из урока дня 12-го "Объектная модель документа XML — DOM", в котором будет идти речь об объектной модели документов, вы сможете (под управлением программы) манипулировать комментариями в документе XML или использовать их в процессе обработки.

Комментарии в экземплярах XML имеют практически такой же вид, как и в HTML:

```
<!--Это комментарий-->
```

Воспользовавшись текстовым редактором, добавьте в документ `bean_dip2.xml` комментарий, который будет напоминать вам о остром соусе при приготовлении блюда. Добавьте комментарий перед элементом `serving`, после чего просмотрите полученный результат с помощью Microsoft Internet Explorer 5.0. Ваш комментарий может выглядеть так:

```
<!--При желании добавить к полученной смеси острый соус-->
```

Результат просмотра полученного кода с помощью Internet Explorer приведен на рис. 2.7. Обратите внимание, парсер в Internet Explorer отображает текст комментария другим цветом после выполнения преобразования XSLT. В уроках дней 3-го и 12-го вы узнаете, что текст комментариев находится в "узле" комментариев, являющемся частью экземпляра XML.

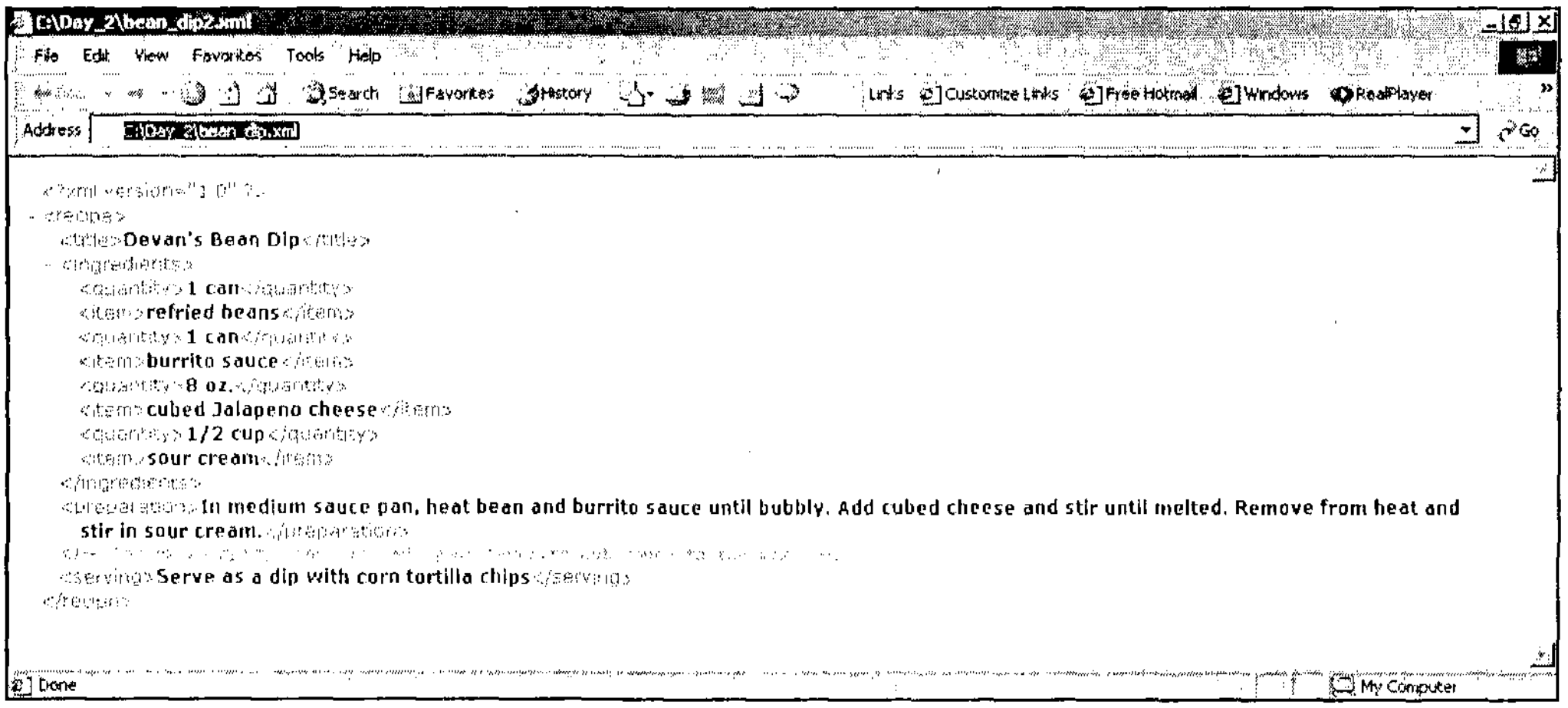


Рис. 2.7. Добавление комментария в документ `bean_dip2.xml`

Размещение атрибутов в экземпляре XML

Вы уже знаете, что атрибуты напоминают прилагательные в обычном "человеческом" языке, поскольку они дополняют элементы, которые мы считаем аналогами существительных. В XML атрибуты всегда заключены в открывающий дескриптор элемента, о котором они предоставляют дополнительную информацию.

Используя текстовый редактор, измените документ `bean_dip2.xml` таким образом, чтобы у элемента `recipe` появился атрибут `title`, а элементы `quantity` превратились в *атрибуты соответствующих элементов* `item`. *После этого ваш документ примет более* профессиональный вид. Синтаксис определения атрибута для элемента следующий:

```
<имя_элемента
имя_атрибута="значение">содержание_элемента</имя_элемента>
```

Атрибуты всегда обьявляются в открывающем дескрипторе соответствующего элемента. Вы можете указать любое количество атрибутов для одного элемента, но все они должны быть заключены в его открывающий дескриптор. Ниже приведен соответствующий пример:

```
<имя_элемента
 имя_атрибута1="значение"
 имя_атрибута2="значение"
 имя_атрибута3="значение">содержание_элемента</имя_элемента>
```

Иногда, когда разметка становится слишком длинной, для упрощения прочтения кода полезно использовать одинаковые отступы при указании подобных элементов. Все четыре строки, в которых определены атрибуты, все равно относятся к одному элементу, несмотря на то, что они занимают четыре строки в документе.



Не забывайте, что вы можете проверить, хорошо ли оформлен ваш документ, воспользовавшись Microsoft Internet Explorer 5.0.

Пример кода XML с использованием атрибутов приведен в листинге 2.3.

Листинг 2.3. Документ XML с использованием атрибутов `bean_dip.xml`

```
<?xml version="1.0" ?>
<recipe title="Devan's Bean Dip">
 <ingredients>
      <item quantity="1 can">refried beans</item>
      <item quantity="1 can">burrito sauce</item>
      <item quantity="8 oz.">cubed Jalapeno cheese</item>
      <item quantity="1/2 cup">sour cream</item>
 </ingredients>
 <preparation>In medium sauce pan, heat bean
      and burrito sauce until bubbly. Add cubed
      cheese and stir until melted. Remove from
      heat and stir in sour cream.</preparation>
 <!--  Дополнительные сведения
      о рецепте  -->
 <serving>Serve as a dip with corn
      tortilla chips</serving>
 </recipe>
```

Обратите внимание, что все атрибуты определены в рамках открывающего дескриптора элемента. Кроме того, обратите внимание и на то, что значения атрибутов указываются в кавычках. Причем могут быть как одинарные, так и двойные кавычки.

Парсеры XML очень "болезненно" относятся к любым нарушениям синтаксических правил. Для того чтобы убедиться в этом, удалите открывающую двойную кавычку из определения первого атрибута в документе. После внесения подобных изменений сохраните документ и попытайтесь обработать его с помощью броузера Internet Explorer. В результате этого вы увидите сообщение об ошибке следующего содержания:

```
The XML page cannot be displayed
Cannot view XML input using XSL style sheet. Please correct the error
and then click the Refresh button, or try again later.
A string literal was expected, but no opening quote character was
found. Line 4, Position 18
  item quantity="1 can">refried beans</item>
-----------------------------------^
```

Затем восстановите открывающую и закрывающую двойные кавычки. Это приведет к отображению сообщения об ошибке совершенно иного содержания. Поскольку парсер не нашел закрывающей кавычки до того, как обнаружил следующий символ открытия объявления разметки (<), он определит, что проблема связана со значением атрибута. Для того чтобы исправить ошибку, вам необходимо изменить строку так, чтобы она выглядела следующим образом:

```
<item quantity="1 can">refried beans</item>
```

Резюме

Сегодня вы познакомились с синтаксическими правилами, характерными для хорошо оформленных документов XML. Вы узнали, что Microsoft Internet Explorer 5.0 или более поздней версии содержит встроенный парсер XML, способный проверять ваши документы на наличие ошибок.

Завтра, в следующем уроке, вы узнаете, каким образом программист может контролировать структуру документа XML, а не только придерживаться определенных правил.

Вопросы и ответы

В чем состоит отличие между XML и HTML по отношению к основным синтаксическим правилам?

Для того чтобы документ XML считался хорошо оформленным, он должен соответствовать всем синтаксическим правилам, определенным консорциумом *W3C* в рамках спецификации XML версии 1.0. Документ XML, в котором хоть раз нарушено одно из таких правил, считается неправильным и его не может обработать парсер или броузер. В случае HTML, несмотря на наличие подобных правил, броузеры не обязаны неукоснительно все их выполнять.

Какие инструменты необходимы для создания хорошо оформленных документов XML?

Вам достаточно иметь под рукой текстовый редактор и парсер, который сможет проверить документ на предмет его соответствия определенным правилам.

Почему анализ очень важен для программистов на XML?

Этот шаг очень важен для понимания элементов структуры документа, что необходимо для правильной разметки документов. Разработчик должен знать, где именно начинается и заканчивается тот или иной элемент, каково его назначение, его содержание, каким образом он связан с другими элементами в данном экземпляре XML.

Каковы основные синтаксические правила, характеризующие хорошо оформленный документ XML?

К основным правилам относятся следующие.

1. Документ XML включает в себя только один корневой элемент, который содержит все остальные элементы.
2. Имена элементов XML чувствительны к регистру символов, поэтому регистр символов, используемых в открывающем и закрывающем дескрипторах элемента, должен быть идентичен.
3. Значения атрибутов в XML должны заключаться в одинарные или двойные кавычки.

Каков синтаксис комментариев в документе XML?

```
<--  Это комментарий!  -->
```

Упражнение

Проверьте свои знания, полученные в результате изучения материала настоящего урока. Ответ вы найдете в Приложении А, "Ответы на вопросы".

В листинге 2.4 приведен пример плохо оформленного экземпляра XML. В этом документе перечисляются два музыкальных альбома и предоставлены некоторые сведения о них. Зная синтаксис XML, воспользуйтесь текстовым редактором для внесения необходимых исправлений. Проверьте работоспособность полученного результата, прогнав его через парсер из состава Microsoft Internet Explorer 5.0.

Листинг 2.4. Неправильно оформленный документ XML

```
<?xml version=1.0?>
    <cd number="432>
    <title>The Best of Van Morrison</Title>
    <artist>Van Morison</artist
    <tracks total=20">
    <cd number=97>
    <title>HeartBreaker</title>
    <Subtitle>Sixteen Classic Performances</subtitle>
    <artistPat Benatar</Artist>
    <tracks total=16>
    </CD>
```

День 3-й

Правильные экземпляры XML

Вы уже познакомились с синтаксическими правилами, которые характеризуют хорошо оформленные экземпляры XML. В настоящей главе вы познакомитесь с концепцией действительного XML.

Сегодня рассматриваются следующие темы.

- Каково значение правильности для XML.
- Почему правильность важна при обмене данными.
- Каким образом XML обращается со структурами данных и документов.
- Некоторые противоречия, связанные с правильностью XML.

Зачем беспокоиться о правильности XML

Вы уже знаете, что использование XML требует строгого следования синтаксическим правилам, так как парсеры XML очень "болезненно" относятся к их нарушению. Действительно, ваш документ XML будет считаться неправильным при наличии в нем хотя бы самой незначительной ошибки. Зачем же вам нужна такая сложная система правил?

Правильность в XML относится к структуре документа. Из первого урока настоящей книги вы уже знаете, что XML позволяет сохранять структурированные данные в обычных текстовых файлах. Однако структура — это несколько больше, чем просто синтаксис. Например, документ XML синтаксически может быть просто безупречным, но содержать элементы в неправильном порядке, или в нем могут отсутствовать элементы, необходимые обработчику для выполнения транзакции электронной коммерции или решения другой задачи.

В качестве примера представим, что разметка настоящей книги выполнена с помощью XML. Вы наверняка будете иметь дело с такими элементами, как `title`, `author`, `publisher`, `ISBN`, `toc` (table of contents — содержание), которые относятся к титульным элементам книги. При разметке материала книги вам не обойтись без таких элементов, как `chapter`, `section`, `paragraph`, `note`, `tip`, `table` и многих других. Вы убедились в том, что разметка может быть достаточно сложной. В качестве примера рассмотрим разметку, которая относится к титульным элементам книги (листинг 3.1).

Листинг 3.1. Пример разметки для титульных элементов книги `MyXMLBook.xml`

```
1: <?xml version="1.0"?>
2: <!-- листинг 3.1 - myXMLBook.xml -->
3:
```

```
4:  <book>
5:     <title>SAMS Teach Yourself XML in 21 Days, Second
Edition</title>
6:     <author>Devan Shepherd</author>
7:     <publisher>SAMS Publishing</publisher>
8:     <isbn>0-672-32093-2</isbn>
9:     <date>August 2001</date>
10:     <toc>
11:       <day number="1">An Overview of the Internet and Markup Lan-
guages</day>
12:       <day number="2">XML Syntax Explored in Detail</day>
13:       <day number="3">Valid XML Instances</day>
14:     </toc>
15: </book>
```

АНАЛИЗ Корневой элемент book (строки 2 —13) содержит такие дочерние элементы, как `title`, `author`, `publisher`, `isbn` и `toc`. Все они хорошо отформатированы, с использованием символов одного регистра в открывающем и закрывающем дескрипторах. Дочерние элементы должным образом вложены в родительские безо всякого пересечения дескрипторов. Элемент `toc` включает в себя три дочерних элемента `day` с содержанием в текстовом формате. Значения атрибутов `number` дополняют элементы `day` точно так же, как прилагательные дополняют существительные в обычных языках.

Документ XML, код которого приведен в листинге 3.1, не только без проблем считывает компьютер, но и полностью понятен человеку. Структура документа может быть представлена в виде простой древовидной диаграммы, подобной той, что показана на рис. 3.1.

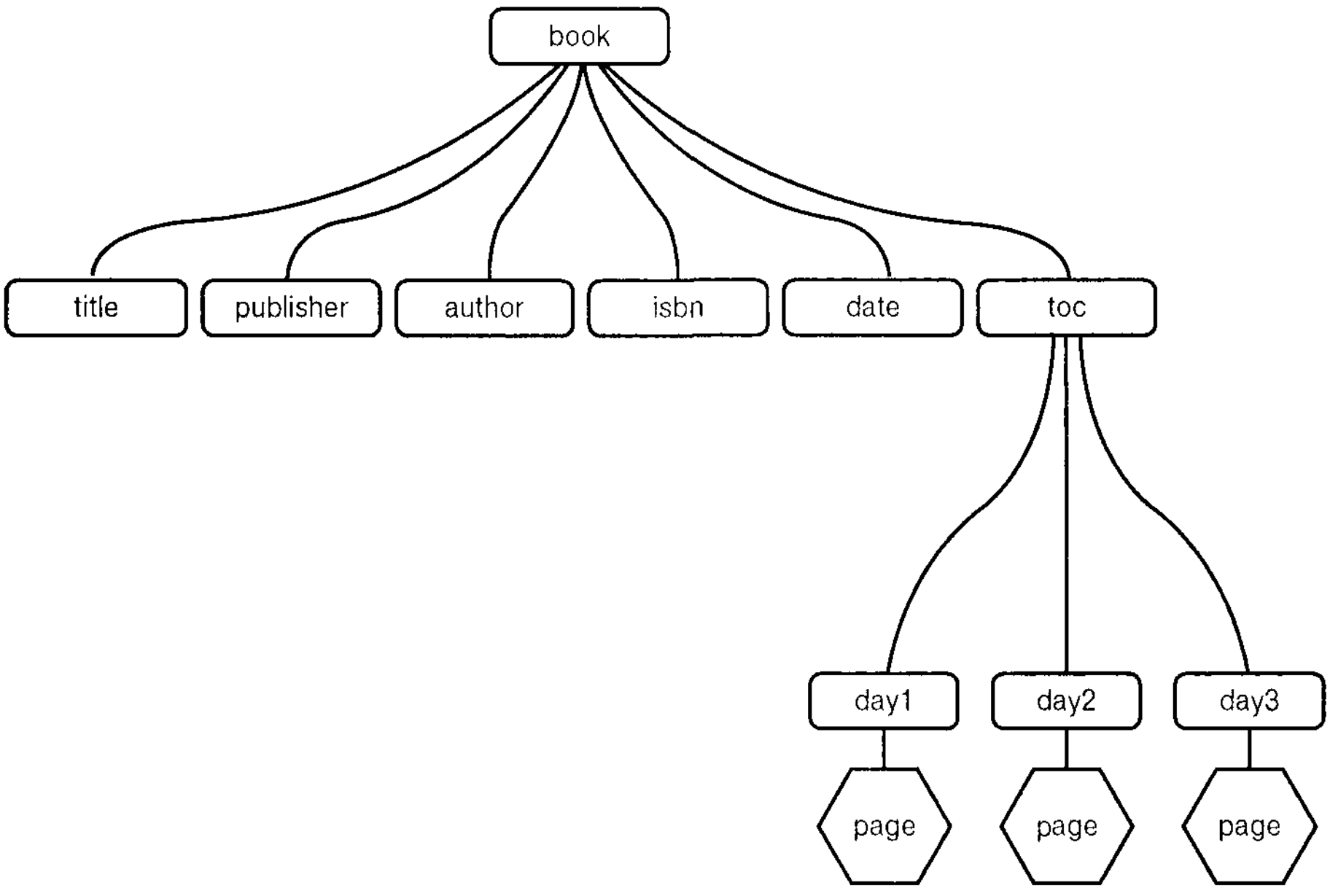


Рис. 3.1. Древовидная диаграмма для документа XML

Теперь изучите документ, приведенный в листинге 3.2. Этот документ также очень хорошо оформлен. Вы можете убедиться в этом, "прогнав" его через парсер XML. Обратите внимание, что Microsoft Internet Explorer не сообщит об ошибках и отобра-

зит экземпляр XML для кода из листинга 3.2. Имейте в виду, что это связано с наличием в Internet Explorer встроенного парсера XML. Несмотря на то, что документ очень хорошо оформлен, он не отражает структуру, показанную раньше на рис. 3.1, а значит, его нельзя использовать для представления титульных данных книги.

Листинг 3.2. Пример хорошо оформленного, но плохо структурированного документа `mixed_front_matter.xml`

```
1: <?xml version="1.0"?>
2: <!-- Листинг 3.2 - mixed_front_matter.xml -->
3:
4: <book>
5:   <toc>
6:     <day number="1">An Overview of the Internet and Markup Languages</day>
7:     <title>Everything You Ever Wanted to Know About XML</title>
8:     <title>XML an Why Cats Chase Laser Pointers</title>
9:     <date>July 2001</date>
10:     <date>June 2001</date>
11:     <date>August 1908</date>
12:     <author>Devan Shepherd</author>
13:     <publisher>Sams Publishing</publisher>
14:     <title>Sams Teach Yourself XML in 21 Days, Second Edition</title>
15:     <day number="1">An Overview of the Internet and Markup Languages</day>
16:     <title>Sams Teach Yourself XML in 21 Days, Second Edition</title>
17:     <date>June 2001</date>
18:     <day number="2">XML Syntax Explored in Detail</day>
19:     <isbn>0-672-32093-2</isbn>
20:     <day number="3">Valid XML Instances</day>
21:   </toc>
22: </book>
```

Очень хорошо оформленный экземпляр XML содержит несколько данных и заголовков, а также ряд неупорядоченных элементов, из-за чего его нельзя считать интуитивно понятным. Например, элемент `toc` содержит даты и другие данные, которые обычно не включаются в оглавление книги. Подобные данные вряд ли включаются в титульные данные при работе с настоящими книгами.

Вам необходим механизм, который бы проверял правильность XML-кода, а не только его форматирование. Если вы описываете настоящую книгу с помощью XML, имеет смысл использовать механизм, который будет проверять определенный набор условий. В большинстве случаев необходимо проверять правила содержания документа, структуру, определяющую порядок следования элементов, наличие необходимых элементов и атрибутов и т.д. Например, в случае с титульными данными книги вы можете определить следующие положения.

- Существует только один корневой элемент `book`, который содержит все остальные элементы.
- Элемент `book` содержит по одному элементу `title`, `author`, `publisher`, `isbn` и `toc` в строго указанном порядке.
- Элемент `toc` содержит строго определенное количество элементов `day`.
- Для каждого элемента `day` задано значение атрибута `number`, соответствующее номеру главы книги.

Все эти простые правила содержания можно объединить в виде схемы, которая определяет структуру экземпляра XML. Программист создает схему для применения к документу, который предназначен для решения определенной проблемы; кроме того, возможен вариант выбора схемы из заранее созданной общедоступной библиотеки.

Большая часть схем, как и другие части семейства XML, расширяема по своей природе. Таким образом, если раньше существовавшая схема только на 80% соответствует вашей конкретной задаче, вы легко сможете расширить ее соответствующим образом. После этого, когда вы будете обмениваться данными с другими пользователями, вы сможете передавать и расширенную схему, чтобы гарантировать, что все пользователи будут видеть полученные от вас данные в соответствующем формате.



Создавать уникальную схему для каждого конкретного случая совсем необязательно. В Web содержатся обширные библиотеки схем для использования в той или иной сфере бизнеса.

В программировании на XML используется несколько разновидностей схем. В уроке дня 4-го "Определение типа документа" вы научитесь создавать, использовать и интерпретировать схемы определения типов документов DTD. Подход с использованием схем DTD представлен еще в SGML, после чего его перенесли и в XML; этот подход характеризуется некоторыми ограничениями, касающимися типов данных и синтаксиса, не совсем совместимыми со стандартами XML. В уроках дней 5-го и 6-го вы познакомитесь с очень мощными схемой XML Data Reduced (XDR) и языком определения схем XML (XSD), предложенными консорциумом *W3C*. Все эти технологии позволяют контролировать бизнес-логику, определяющую структуру ваших документов. Каждый подход обладает своими достоинствами и недостатками, однако со временем все они должны объединиться в один общий стандарт консорциума *W3C*, после чего получат широкую поддержку со стороны отрасли и разработчиков программного обеспечения.



Проверка типа данных означает классификацию значений данных. В результате этой проверки компьютер получает возможность различать такие типы данных, как целые числа, проценты, числа с плавающей запятой, даты и строковые значения и т.д.

Проверка типа данных – еще одна причина для определения правильности

Необходимость определения правильности документов уже была проиллюстрирована на нескольких примерах разметки раньше в настоящей главе. Представьте себе, что вы осуществляете транзакции B2B (Business-to-Business) через Internet со своим деловым партнером. Одно из основных преимуществ XML в данной ситуации заключается в его независимости от конкретной платформы. Это означает, что вы сможете вести бизнес с любым человеком или организацией через Web, независимо от используемых ими операционных систем и языков программирования и баз данных. Однако такой подход эффективен только в том случае, если вы гарантируете структуру документов, которыми обмениваетесь. Вам необходимо проверять структуру транзакции на предмет не только порядка следования и конструкцию отдельных элементов, но и на соответствие типов данных.

Такие элементы и знания атрибутов, которые относятся к простым типам данных, например датам, могут передаваться без какой-либо сложной проверки. Однако, если ваш деловой партнер находится в другой стране, он может использовать формат представления дат, отличный от принятого у вас. В табл. 3.1 приведены все существующие форматы представления даты на примере 9 октября 2001 года. Несмотря на, что сегодня общепринято представление года в виде четырех цифр, в таблице приведены и варианты представления года в виде всего двух цифр, чтобы только проиллюстрировать различные возможные проблемы.

Таблица 3.1. Пример представления даты 9 пктября 2001 года с использованием различных форматов

Формат	*Значение*
mmddyy (мм.дд.гг)	100901
ddmmyy (дд.мм.гг)	091001
yyddmm (гг.дд.мм)	010910
yymmdd (гг.мм.дд.)	011009

Для подтверждения целостности транзакций данных с деловыми партерами вам следует использовать в схеме только один формат данных. Вы уже увидели, что числовые значения содержат очень мало информации и могут легко привести к путанице, результатом чего будет неправильная обработка данных.

В этой книге вы познакомитесь с несколькими подходами к контролю типов данных, например, с помощью XDR или DTD. Однако следует иметь в виду, что схема DTD не позволяет проводить проверку правильности типов данных. Это одно из ключевых отличий между схемами DTD и всеми остальными подходами. Подробные сведения вы найдете в уроках 5 и 6. В дальнейшем, при создании собственной схемы, вы сможете выбрать методологию, которая наилучшим образом соответствует вашим потребностям.



Можно описывать типы данных, используя представления совместно со схемами DTD; однако такой подход оказывается далеко не идеальным.

В табл. 3.2 представлено несколько примеров данных, для которых может потребоваться проверка правильности, что зависит от того, обмен какой именно информацией осуществляется. Вы уже могли применять проверку типов данных при использовании других языков программирования.

Таблица 3.2. Примеры типов данных

Тип данных	*Описание*
Boolean	0 (ложь) или 1 (истина)
Char	Один символ, например "C"
String	Последовательность символов, например "CIST 1463"
Float	Действительное число, например 123.4567890
Int	Целое число, например 5
Date	Дата в формате YYYY-MM-DD (например, 2001-10-09)
Time	Время в формате HH-MM-SS (например, 18:45:00)
Id	Текст, уникальным образом идентифицирующий элемент или атрибут
Idref	Ссылка на идентификатор
Enumeration	Последовательность значений, любое из которых можно выбрать

Указание типа данных используется для определения природы тех или иных данных. Структура документа также требует отдельного рассмотрения. Структура — это схема организации связанных элементов информации. Порядок и расположение элементов в XML часто является частью структуры всего документа. В следующем разделе мы подробно рассмотрим вопросы, связанные со структурами в XML.

Как определить структуру данных

В листингах документов XML, с которыми вы уже познакомились в настоящем уроке, много общих элементов, но их структуры значительно различаются. Порядок расположения элементов не идентичен, хотя имена элементов полностью совпадают. В результате один из документов (см. листинг 3.1) интуитивно понятен, а второй документ (см. листинг 3.2) явно страдает от ряда погрешностей в разметке.

Структура рецепта приготовления блюда

В XML вам необходимо очень внимательно исследовать документы и понимать структуры данных. При изучении урока дня 2-го вы познакомились со структурой рецепта и создали две модели для ее описания. Одна из них представляла собой блок-схему (см. рис. 2.5), а вторая — рецепт в виде древовидной структуры (см. рис. 2.6). В результате выделения определенных элементов рецепта в отдельные элементы данных вы получаете файл `bean_dip3.xml`, текст которого приведен в листинге 3.3.

Листинг 3.3. Файл `bean_dip3.xml`

```
1:  <?xml version="1.0"?>
2:  <!-- листинг 3.3 - bean_dip3.xml -->
3:
4:  <recipe title="Devan's Bean Dip">
5:   <ingredients>
6:        <item quantity="1 can">refried beans</item>
7:        <item quantity="1 can">burrito sauce</item>
8:        <item quantity="8 oz.">cubed Jalapeno cheese</item>
9:        <item quantity="1/2 cup">sour cream</item>
10:  </ingredients>
11:  <preparation>In medium sauce pan, heat bean
12:       and burrito sauce until bubbly.  Add cubed
13:       cheese and stir until melted. Remove from
14:       heat and stir in sour cream.</preparation>
15:        <!-- Добавьте свои любимые
16:             ингредиенты  -->
17:  <serving>Serve as a dip with corn
18:       tortilla chips</serving>
19:  </recipe>
```

Сначала вы определили структуру документа с рецептом в соответствии с выбранной моделью. После этого вы определили имена элементов и атрибутов, которые интуитивно понятны и являются самоописывающимися. После этого вы передаете все эти сведения парсеру, а это гарантирует, что созданная вами структура понятна компьютеру, после чего просматриваете структуру, чтобы убедиться в том, что она понятна и человеку. Это очень неплохой пример подхода, который вам следует использовать при последующем создании нового экземпляра XML, базирующегося на документе с известной структурой. Для проверки правильности вам необходимо рассмотреть правила, связанные со структурой, после чего создать схему, которая будет следить за выполнением правил при "прогонке" документа через парсер. Например, в случае рецепта приготовления блюда вам необходимо обеспечить выполнение следующих правил.

- Существует один и только один корневой элемент `recipe`, содержащий все остальные элементы.
- Элемент `recipe` содержит необходимый атрибут `title`.

- Элемент `recipe` должен содержать по одному экземпляру элементов `ingredients` и `preparation`, причем в строго указанном порядке, а также необязательный элемент `serving`.
- Элемент `ingredients` должен содержать хотя бы один, но может содержать и любое количество элементов `item`.
- Элементы `item` содержат необходимый атрибут `quantity`, определяющий нужное количество определенного ингредиента блюда.

Вы уже можете использовать структурированные данные

Ведя свои дела, вы уже могли *сталкиваться* со структурами данных. Во многих компаниях существуют правила, касающиеся работы с корреспонденцией, форматов счетов-фактур и т.д. Хорошая структура документа обычно очевиднее, чем хорошая структура данных. Думая о структуре данных, вспомните четкую организацию баз данных. Это очень хорошая аналогия, поскольку правила, определяющие структуру баз данных, точно так же как и правила в XML, в большей степени касаются классификации данных, чем того, каким образом эти данные можно представить в форме или отчете. Получение результатов из базы данных требует использования генератора отчетов или другого приложения, функции или запроса, которые не связаны с собственно с хранением или проверкой правильности данных. Все это справедливо и в отношению XML — XML имеет дело только с данными, а для получения определенных результатов требуется использование каких-то внешних средств.

Если речь идет о проверке правильности XML, на самом деле подразумевается структура данных, а не их представление. Web-разработчикам приходится тратить намного больше времени на структуру данных, на вопросы хранения, извлечения и поддержания, чем на их представление в окне броузера. Специалисты по составлению технической документации или всевозможных официальных отчетов на протяжении многих лет имеют дело со сложными наборами заранее определенных структур документов, отклонение от которых не допускается. XML также предлагает возможности создание структур и их распространение среди деловых партнеров и других пользователей информации.

Однако, как уже отмечалось, XML прежде всего работает с данными. XML практически не представляет никаких средств для представления данных. Поэтому, в отличие от структуры документа, которая связывается с устройством вывода, предназначенным для создания бумажных отчетов, структуры данных XML касаются только собственно данных. Когда дело доходит до отображения данных, вы используете различные подходы, в том числе и каскадные таблицы стилей, расширяемый язык таблиц стилей XSL, такую часть XSL, как преобразования XSLT, а также различные сценарии. Со всеми эти подходами вы познакомитесь по мере освоения материала настоящей книги.



Тот факт, что XML разделяет структуры данных от стилей и содержания, и является одной из причин потрясающей гибкости этого языка. Один и тот же экземпляр XML можно использовать для получения бумажных отчетов, создания библиотек на компакт-дисках или представления данных в окнах броузеров, на экранах сотовых телефонов с поддержкой Web-технологий, персональных цифровых помощников (PDA), а также других *агентов пользователя.*

Каково же основное различие между структурой документа и структурой данных? Структура документа позволяет читателю быстро понять, в каком именно виде представил информацию автор документа. Когда вы пишете записку своему сотруднику, вы обычно указываете, кому эта записка адресована, свое имя, дату, тему и степень важности, после чего следует собственно содержание записки. Тогда получатель может быстро просмотреть все сведения о записке, а затем перейти к прочтению ее содержания. Структуры данных

указывают компьютерным приложениям путь к данным, которые содержатся в различных контейнерах целого документа. В структуре данных отсутствует как таковое определение важности одного компонента документа по отношению к другим компонентам. Все компоненты "равны", а уже приложение или обработчик XML должны определить, каким образом использовать данные, полученные из того или иного контейнера.

Вы можете использовать XML для добавления структуры к неструктурированным данным

XML предоставляет вам замечательный механизм использования структурированных данных при передаче информации. Действительно, используя XML, вы сможете легко смешивать и сопоставлять структурированные и менее структурированные данные, в том числе текст, числовые и строковые данные для получения потоков вывода для решения определенных задач. Вы можете получить один и тот же результат, используя широкий диапазон подходов, и здесь роль XML переоценить невозможно.

В табл. 3.3 приведен фрагмент базы данных, содержащей сведения о различных учебных курсах по XML. В таблице базы данных присутствуют такие поля, как `Curse_Number`, `Curse_Name` и `Instructor`.

Таблица 3.3. Примеры курсов из таблицы базы данных

Номер курса	*Название курса*	*Инструктор*
XML111	Introductory XML	Bob Gonzales
XML333	Advanced XML	Devan Shepherd
XMT222	XMetal Core Configuration	Gene Yong

Если данные из этой таблицы будут сохранены в текстовом файле с разделителями-запятыми, результат будет иметь такой вид, как показано в листинге 3.4.

Листинг 3.4. Представление данных из таблицы базы данных в текстовом формате

```
Course_Number, Course_Name, Instructor
XML111,Introductory XML,Bob Gonzales
XML333,Advanced XML,Devan Shepherd
XMT222,XMetal Core Configuration,Gene Yong
```

XML предоставляет возможность добавлять к подобным данным дополнительные сведения, имеющие значение для людей или компьютеров. Например, в листинге 3.5 показаны те же самые данные, но к ним добавлены дополнительные сведения, используемые при подготовке пресс-релиза о предстоящих учебных курсах.

Листинг 3.5. Структурированные данные из таблицы базы данных, а также дополнительные сведения, представленные в формате XML

```
1: <?xml version="1.0"?>
2: <!-- листинг 3.5 - courses.xml -->
3:
4: <announcement>
5:   <dist>For Immediate Release</dist>
6:   <to>All Potential Students</to>
7:   <from>Devan Shepherd</from>
8:   <subject>Public Course Offerings in August</subject>
```

```
 9:    <notice>ACME Training is pleased to announce the following
10:     public courses, which are offered on a monthly basis.</notice>
11:    <contact>For more information, or to register for any of these courses,
12:     please visit
13:  <website>http://ACME-Train.com/university</website></contact>
14:    <courses>
15:      <course id="XML111" instructor="Bob Gonzales">
16:       Introduction to XML</course>
17:      <course id="XML333" instructor="Devan Shepherd">
18:       Advanced XML</course>
19:      <course id="XMT222" instructor="Gene Yong">
20:       XMetal Core Configuration</course>
21:    </courses>
22:  </announcement>
```

Теперь полученный результат имеет более удобную форму, чем текстовый файл с разделителями-запятыми, не только благодаря наличию дополнительных сведений, но и благодаря использованию атрибутов. Источник XML можно использовать для создания сообщения электронной почты, газеты или другого печатного документа. Те же самые данные можно преобразовать с помощью специальной программы к другому языку разметки. Используя технологии XML, вы сможете получать результаты, пригодные для отображения с помощью различных беспроводных устройств с поддержкой Web, таких как сотовые телефоны, персональные цифровые ассистенты, пейджеры и т.д.

В уроке дня 14-го "Использование каскадных таблиц стилей CSS" вы познакомитесь с использованием каскадных таблиц стилей CSS. В уроках 15 и 16 вы познакомитесь с расширяемым языком таблиц стилей XSL, а также его подмножеством, преобразованиями XSL, которые позволяют преобразовывать результаты вывода для их использования броузерами или другими агентами пользователей, устройствами и обработчиками.



Правильный XML гарантирует целостность структуры данных. Благодаря этому значительно упрощается доставка данных, параметры которой корректируются с помощью XSLT.

Парсер проверяет правильность

Вы уже неоднократно использовали парсер, выполняя упражнения. В состав броузера Microsoft Internet Explorer входит парсер MSXML, начиная с версии 5.0 или более поздней. Таким образом, вы проверяли форматирование своих документов именно с помощью парсера MSXML. Однако вы действительно использовали его только для проверки форматирования документов. Парсеры также способны проверять правильность вашего документа XML с помощью таких схем, как DTD, XDR или XSD, а также ряда других. Подобная проверка проиллюстрирована на рис. 3.2.



Правильный документ XML также и хорошо оформлен; однако нет никакой гарантии, что хорошо оформленный документ XML является правильным.

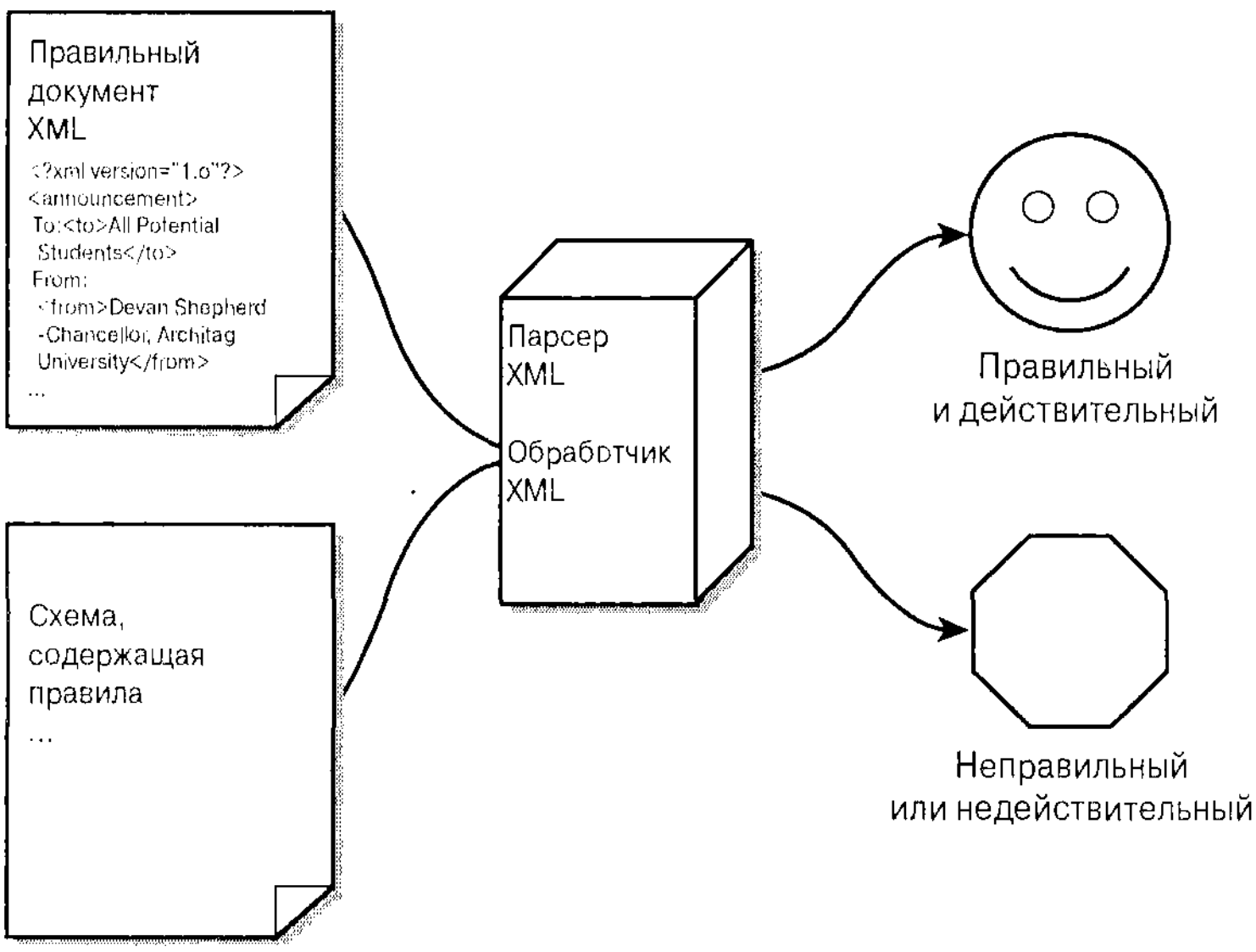


Рис. 3.2. Документ XML и соответствующая структура, переданные парсеру для проверки правильности

При проверке правильности парсер сначала определяет, соответствует ли экземпляр XML основным синтаксическим правилам, т.е. является ли он хорошо оформленным. После этого он проверяет, удовлетворяют ли все правила, применяемые к содержанию документа, связанной с документом схеме (или схемам, поскольку с одним документом могут быть связаны несколько схем), что гарантирует правильность документа. Другими словами, при проверке правильности парсер должен проверить не только синтаксис, но и структуру данных.

Парсер — это всего лишь приложение, которое интерпретирует текст символ за символом до тех пор, пока его не проинструктирует программист или автор схемы пропускать определенные последовательности данных. XML позволяет использовать парсеры для проверки семантики разметки, примененной к тексту.

ПРИМЕЧАНИЕ

В официальной спецификации XML 1.0 консорциум *W3C* называет парсер обработчиком XML: "Программный модуль, который называется обработчиком XML, предназначен для чтения документов XML и предоставления доступа к их содержанию и структуре." (Подробные сведения вы найдете в Web по адресу `http://www.w3.org/TR/1998/REC-xml-19980210`.)

После того как парсер интерпретирует ваш документ XML и подтвердит его форматирование и правильность, он передаст данные в форме структуры документа другим приложениям для дальнейшей обработки. Для этого вы уже использовали Internet Explorer 5.0, который с помощью древовидной структуры, возвращенной парсером MSXML, отображает структурированный документ в окне броузера. Вы уже помните, что Internet Explorer использует преобразования XSLT для назначения документу XML стандартной таблицы стилей при его преставлении в окне броузера. Именно по этой причине различные элементы в окне Internet Explorer отображаются разным цветом. Либо Internet Explorer выявляет ошибку, полученную парсером, и отображает ее на экране вместе с фрагментом кода, в котором она содержится, вместо того, чтобы отображать целиком весь экземпляр XML.

Подобная концепция документов, представленных в виде древовидной структуры, является причиной того, что вы начали с представления документов именно таким

образом при изучении урока дня 2-го (см. рис. 2.6.). Вы неоднократно будете возвращаться к использованию подобной структуры при изучении материала книги. В уроке дня 12-го "Объектная модель документа XML — DOM" вы научитесь использовать прикладной интерфейс API объектной модели документа DOM для манипулирования отдельными узлами дерева XML.

Неужели правильность — это все, что нам нужно?

Вы уже увидели, что правильность означает непротиворечивость структуры данных, но гарантирует ли она, что люди, с которыми вы будете обмениваться данными, знают, как эти данные следует использовать? Конечно, нет! Правильность лишь гарантирует, что структура данных в экземпляре XML позволит приложению определить, каким образом эти данные должны быть обработаны в дальнейшем. Даже если вы опубликуете вместе с экземпляром XML соответствующую ему схему, этого все равно будет недостаточно для решения определенной задачи.

Еще одно заблуждение состоит в том, что без схемы документа невозможно вести бизнес через Web с использованием XML. Схема — это хороший, но далеко не единственный способ проверки целостности данных, предназначенных для использования XML. Основное преимущество использования схем совместно с документами XML — упрощение контроля над полученным с помощью компьютера содержанием, улучшение поиска, а также проверка правильности структуры данных. В последующих главах вы научитесь использовать приемы программирования для манипулирования данными без построения логических способов проверки правильности, если с вашими документами XML связаны определенные схемы.

Резюме

Сегодня вы узнали, что схемы могут очень эффективно использоваться для того, чтобы убедиться, что структуры данных размечены именно так, как вы того и ожидали. В частности, это означает, что для проверки правильности вам необходимо выполнить следующие действия.

- Определить использование только заданного набора дескрипторов.
- Проверить, чтобы порядок следования элементов и их атрибутов полностью соответствовал содержанию документа или определенным правилам.
- Контроль типов данных (это достигается благодаря использовании соответствующей схемы).
- Контроль целостности данных для обеспечения оптимального обмена информацией через Web с помощью транзакций.

Вопросы и ответы

Каково различие между хорошо оформленным и правильным XML?

Хорошо оформленный XML безупречен с точки зрения синтаксиса — он соответствует всем основным синтаксическим правилам, определенным для документов XML. Правильный XML обязательно является хорошо оформленным, но также соответствует всем ограничениям, которые налагает связанная с ним структурная схема.

Какие ограничения может использовать программист при создании схемы для документа XML?

Схема четко определяет имя и структуру корневого элемента, включая спецификацию всех его дочерних элементов. Программист может задать, какие элементы и в каком количестве обязательны, а какие — необязательны. Схема определяет, какие элементы содержат атрибуты, а также допустимые значения этих атрибутов, в том числе и значения по умолчанию. Как и в случае элементов, атрибуты могут быть обязательными или необязательными.

Какие языки описания схем могут использовать авторы документов XML?

На сегодня используется несколько видов языков описания схем. Язык определения типов документов DTD, который существует еще со времен языка SGML, будет рассмотрен в следующем уроке. Компания *Microsoft* разработала такой диалект XML, как XML Data Reduced (XDR), о котором мы поговорим в уроке 5. Консорциум *W3C* работает над превращением различных диалектов схем в один язык описания схем, который называется XSD; он рассматривается в уроке 6.

Можно ли использовать схемы для контроля над типами данных в потоке данных XML?

Такие языки описания схем, как XDR и XSD, уже могут ограничивать типы данных. Подход с использованием DTD не позволяет легко создавать правила использования типов данных. Программисты еще на заре развития XML определили, что проверка типов данных оказывается чрезвычайно важной для экземпляров XML, используемых в моделях электронной коммерции и транзакций электронного бизнеса.

Каково различие между структурой документа и структурой данных?

Структура документа позволяет читателю быстро понять, в каком именно виде представил информацию автор документа. Структуры данных указывают компьютерным приложениям путь к данным, которые содержатся в различных контейнерах в рамках целого документа. В структуре данных отсутствует как таковое определение важности одного компонента документа по отношению к другим компонентам.

Что такое парсер?

Парсер -- это программа (которая в спецификациях консорциума *W3C* называется обработчиком XML), которая интерпретирует текст символ за символом. Существует два общих типа парсеров, используемых при работе с XML. Парсеры первого типа способны проверять только форматирование документов, т.е. их соответствие основным синтаксическим правилам XML. Парсеры второго типа сначала проверяют форматирование документов, после чего проверяют их соответствие всем ограничениям, указанным в связанных с ними схемах. Прошедший подобную проверку документ XML считается правильным.

Упражнение

Проверьте свои знания, полученные в результате изучения настоящего урока. Ответы вы найдете в Приложении А, "Ответы на вопросы".

В упражнении из предыдущей главы вам предлагалось исправить плохо оформленный документ XML. Документ описывал два музыкальных альбома и содержал такие элементы, как `cd`, `title`, `artist` и `tracks`. Вы можете создать новый специализированный язык разметки, который будет называться MCML. Создайте древовидную диаграмму для этого языка разметки, а также определите ряд правил для проверки правильности.

День 4-й

Определение типа документа

Вы уже узнали, что документ XML действителен, если он соответствует всем ограничениям, изложенным в связанной с ним схемой. Сегодня вы познакомитесь с одной из таких схем, которая называется DTD (Document Type Definition — Определение типа документа).

В частности, сегодня рассматриваются следующие темы.

- Что такое схема DTD и каким образом она связывается с документами XML.
- Как написать внутреннюю или внешнюю схему DTD.
- Объявление элементов и атрибутов в DTD.
- Использование ссылок в схемах DTD.
- Различие между парсерами с проверкой правильности и без нее.
- Некоторые недостатки схем DTD и почему необходимо создание новых схем.

Откуда появились схемы DTD

Как вы уже знаете, связывание схемы с документом XML означает возможность распространения документа независимо от приложения. Схема добавляет ряд ограничений, которые определяют список необходимых элементов и атрибутов, порядок их следования, а также, при необходимости, их допустимые значения. Используя подобные ограничения, вы сможете обмениваться данными с другими пользователями, а для проверки использовать обычную схему. Как вы узнаете, схему можно встроить в приложение для проверки правильности документа XML, предоставляющего данные для решения других задач. В роли такого приложения выступает парсер. Парсер проверяет правильность документа XML, определяя, что он хорошо оформлен, а также проверяет его соответствие всем ограничениям, налагаемым связанной внешней (или внутренней) схемой.

DTD — это схема, разработанная еще для языка SGML. Когда миру впервые представляли язык XML, он предполагался чем-то наподобие "SGML-без-DTD". С некоторой точки зрения оно так и есть: вы можете получить хорошо оформленный документ XML, с которым не будет связано никакой схемы; но, как вы уже знаете, схема DTD — это мощное средство добавления ограничений на обмен данных, позволяющее контролировать использование в документах должных структур данных. Вы можете использовать XML без DTD или любой другой схемы; ведь именно таким и яв-

ляется хорошо оформленный XML, однако схемы позволяют проверять правильность содержания документа на предмет соблюдения некоторых определенных правил.



> Схема DTD (как и любая другая схема) предоставляет шаблон разметки документа, в котором указываются наличие, порядок следования и расположение элементов и их атрибутов в документе XML.

Вы уже узнали, что документ XML можно представить в виде древовидной структуры элементов, содержащей данные, другие элементы и атрибуты. Схему DTD также можно представить в виде структуры, определяющей дерево, однако между деревом DTD и деревом документа XML существует ряд различий даже в том случае, если документ XML полностью соответствует схеме DTD.

Дерево DTD само по себе не повторяет элементы или структуру. Однако его структуры дублируют элементы в соответствующем правильном экземпляре XML. Изучите документ XML, приведенный в листинге 4.1. (Нечто подобное вы уже видели при изучении урока дня 3-го.)

Листинг 4.1. Объявление учебных курсов в виде экземпляра XML `courses.xml`

```
1:  <?xml version="1.0"?>
2:  <!-- листинг 4.1 - courses.xml -->
3:
4:  <announcement>
5:  <dist>For Immediate Release</dist>
6:  <to>All Potential Students</to>
7:  <from>Devan Shepherd</from>
8:   <subject>Public Course Offerings in August</subject>
9:    <notice>ACME Training is pleased to announce the following
10:    public courses, which are offered on a monthly basis at its
11: headquaters.</notice>
12:   <more_info>For more information, or to register for any of these
13:   courses, please visit <website>http://ACME-
       Train.com/university</website></ more_info>
14:    <courses>
15:      <course id="XML111" instructor="Bob Gonzales">Introduction to
         XML</course>
16:      <course id="XML333" instructor="Devan Shepherd">Advanced
         XML</course>
17:      <course id="XMT222" instructor="Gene Yong">XMetal Core
         Configuration</course>
18:    </courses>
19: </announcement>
```

В строке 14 этого листинга содержится элемент `courses`, который содержит дочерние элементы `course`. С учетом этого, диаграмма дерева документа для этого экземпляра XML может выглядеть так, как показано на рис. 4.1.

Дерево документа, показанное на рис. 4.1, содержит несколько элементов `course`, каждый из которых обладает атрибутами `id` и `instructor`. В дереве DTD элемент `course` будет указан всего один раз, но при этом он содержит специальный оператор +, указывающий на наличие нескольких однотипных элементов. Подробно с этим и некоторыми другими операторами схемы DTD вы познакомитесь дальше, в настоящем уроке. Пример дерева DTD для файла `courses.xml` приведен на рис. 4.2.

В настоящей главе вы научитесь создавать и "читать" простые схемы DTD, которые предоставляют ограничения, отражаемые соответствующими деревьями DTD. Существует

целый ряд специальных инструментов, которые позволяют создавать схемы DTD для проверки правильности хорошо оформленных документов XML на основе нескольких подобных документов. (Например, вы сможете найти одну из подобных программ в Internet, по адресу `http://www.pault.com/pault/dtdgenerator/`. Если вы предпочитаете запускать DTD-генератор локально, вы можете загрузить набор средств SAXON с Web-узла `http://users.iclway.co.uk/mhkay/saxon/saxon6.0.2/index.html`.) Однако сначала вам необходимо научиться создавать схемы DTD вручную.

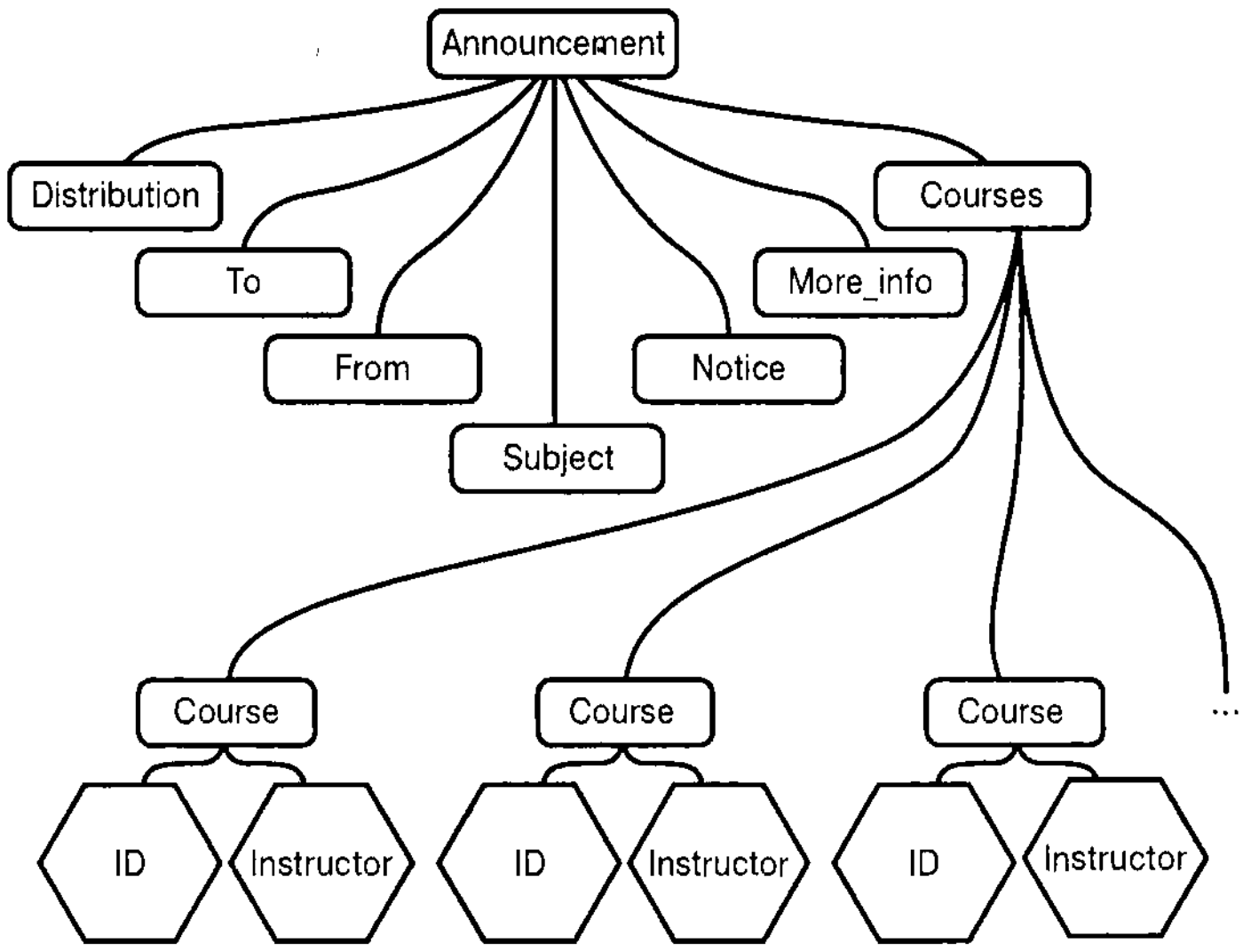


Рис. 4.1. Дерево документа для файла `courses.xml`

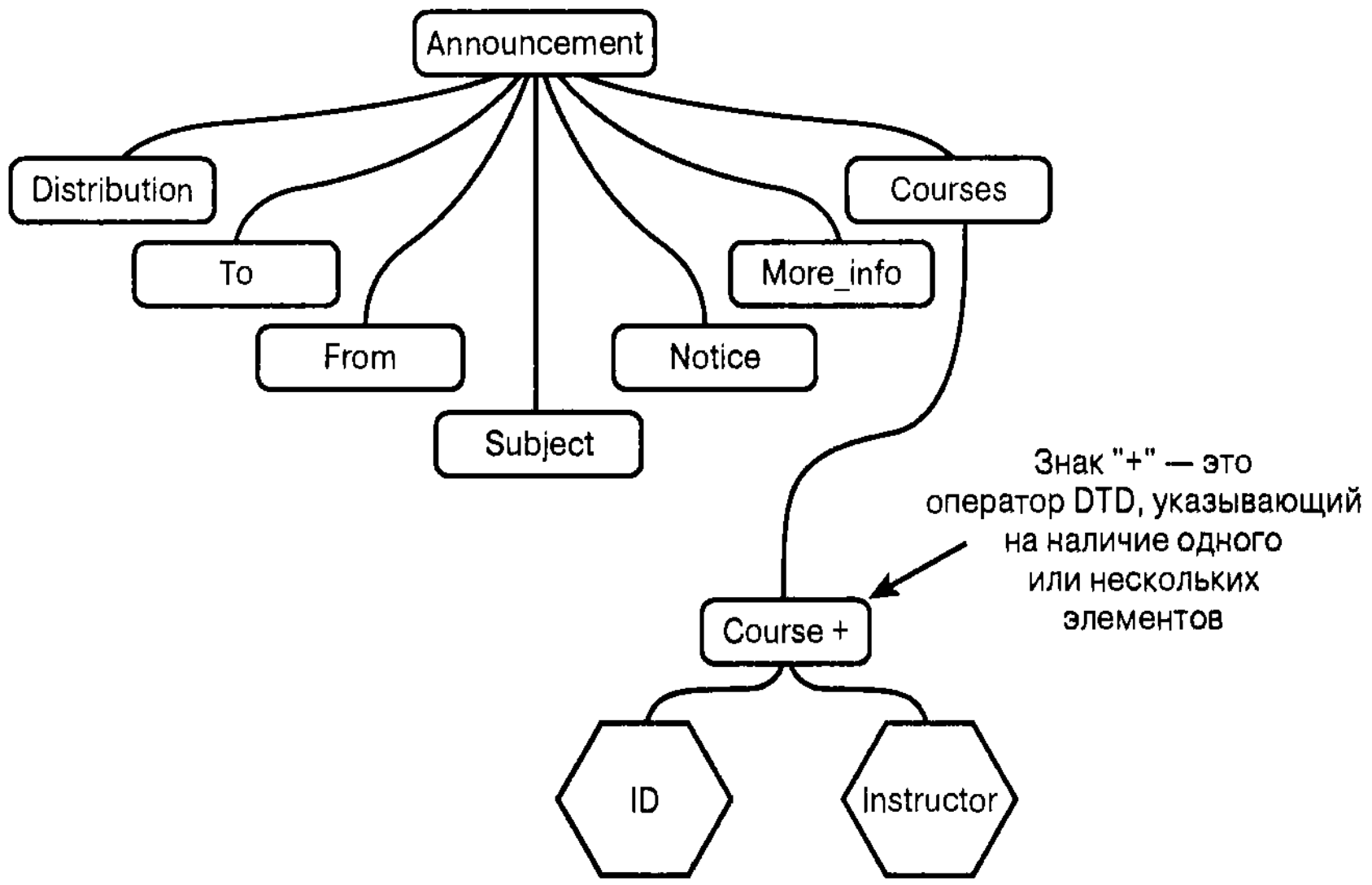


Рис. 4.2. Дерево DTD для файла `courses.xml`

Знакомство с правилами

Схемы DTD имеют дело с элементами документа XML или как с элементами-контейнерами, или как с пустыми элементами (т. е. подстановочными элементами в структуре документа). Элементы-контейнеры могут содержать данные (например, текст), дочерние элементы или и то, и другое. Схема DTD предоставляет синтаксические правила для представления ограничений для этих обеих моделей содержания.



Объявление содержания элемента или атрибута в схеме DTD называется моделью содержания для этого элемента или атрибута.

Синтаксис объявления типов основных элементов в DTD

Элементы в любом документе XML — это фундаментальные структуры, которые объединены для представления экземпляра XML. Каждый элемент должен быть объявлен в схеме DTD вместе с объявлением его типа.

Объявления типов элементов имеют следующий вид:

```
<!ELEMENT    имя_элемента (модель содержания)>
```



Все приведенные ниже примеры связаны между собой. Я предлагаю вам создавать каждый из этих примеров, вводя программный код для каждого листинга в текстовом редакторе. Затем, дальше в настоящем уроке, вы проанализируете некоторые из этих примеров, чтобы выяснить, представляют ли они правильный XML или нет. Сохраняйте каждую новую версию документа в виде отдельного файла, присваивая такие имена, как `message01.xml`, `message02.xml` и т. д.

Простой элемент с текстовым содержанием

Изучите небольшой документ XML, приведенный в листинге 4.2 в качестве простого элемента. Действительно, это очень простой документ XML, который содержит всего один элемент `note`. Создайте этот документ и сохраните под именем `message01.xml`.

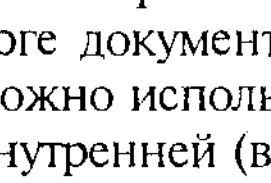
Листинг 4.2. Небольшой хорошо оформленный экземпляр XML `message01.xml`

```
1:  <?xml version = "1.0"?>
2:  <note>Remember to buy milk on the way home from work</note>
```

Элемент `note` содержит только текст и ничего больше. Вы можете проверить этот документ, создав схему DTD с объявлением типа элемента, указывающим на то, что элемент содержит только зарезервированное ключевое слово `#PCDATA` или текст. `PCDATA` — это обычные текстовые данные, но которые считываются парсером XML и обрабатываются должным образом. Итак, если в разметке вашего документа содержится ключевое слово `#PCDATA`, это повлияет на обработку всего документа. Это может быть как желательным, так и нежелательным эффектом. Дальше вы узнаете о способах предотвращения интерпретации текста парсером благодаря использованию типа данных `CDATA`. Данные этого типа — это тоже только текст, но он не обрабаты-

вается парсером; символы разметки, такие как угловые скобки, игнорируются в сегментах CDATA и обрабатываются в сегментах PCDATA.

Поскольку схема DTD обеспечивает проверку правильности структуры документов, она содержит правила для содержания. С каждым элементом или атрибутом связана определенная модель содержания. Например, элемент может содержать текст или другие элементы, а может быть просто пустым элементом. Все эти модели содержания представляются в схемах DTD по-разному. Схемы DTD могут быть указаны в различных документах и быть связанными с определенными инструкциями в документах XML, а могут содержаться прямо в этих документах. В настоящей главе вы сначала познакомитесь с внутренними схемами, а затем, несколько позже, — со внешними.

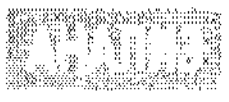


Зарезервированное ключевое слово #PCDATA в схемах DTD всегда указывается только знаками верхнего регистра.

В листинге 4.3 показан документ XML со внедренной схемой DTD. Создайте этот документ и сохраните под именем message02.xml. В строке 3 содержится объявление типа элемента, которое будет рассмотрено нами более подробно.

Листинг 4.3. Небольшой хорошо оформленный и правильный экземпляр XML message02.xml

```
1:  <?xml version = "1.0"?>
2:  <!DOCTYPE note [
3:  <!ELEMENT note          ( #PCDATA)>
4:                  ]>
5:  <note>Remember to buy milk on the way home from work</note>
```



Строка 1 — это обычное объявление XML, которое вы будете использовать во всех остальных примерах. В этой строке указано, что необходимо использовать версию 1.0 спецификации XML, и не больше того.

В строке 2 содержится объявление определенного типа документа, которое дается в прологе документа XML, который связывает с документы схему DTD. Вы скоро узнаете, как можно использовать определение типа документа для указания того, что схема DTD является внутренней (в этом случае она называется внутренним подмножеством) или внешней (в этом случае она называется внешним подмножеством). Синтаксис этих двух различных подходов несколько различается и будет рассмотрен дальше в настоящей главе. Определение типа документа всегда начинается с <!DOCTYPE и заканчивается символом >. Ключевое слово DOCTYPE всегда указывается только символами верхнего регистра. Первые несколько строк в документе XML, которые следуют до строки, в которой содержится корневой элемент, часто называют *прологом* документа. Пролог содержит инструкции обработки документа, такие как объявление XML (<?xml version = "1.0"?>) и объявление DOCTYPE в случае использования схемы DTD. В уроках 5 и 6 вы познакомитесь с другими видами схем, которые могут связываться с экземплярами документов XML, используя различные сведения в прологе документа или атрибуты в открывающем дескрипторе корневого элемента.

Объявление типа документа в строке 2 сообщает обработчику XML о том, что существует объявление элемента note, а все сведения, заключенные в квадратные скобки, являются внутренней схемой DTD. Другими словами, строка 2 начинает внутреннее подмножество DTD под названием note. В данном случае, note — это корневой элемент.

В строке 3 содержится объявление типа элемента для элемента note. В частности, здесь указано, что элемент note может содержать только текстовые данные. В скобках указывается такая часть объявления типа элемента, как модель содержания или спецификация содержания. Именно с помощью модели содержания вы сообщаете парсеру

XML, что следует ожидать от каждого элемента XML в вашем документе. В некоторых случаях вам необходимо указать, что элемент пуст. В других случаях вы объявляете элемент-контейнер, содержащий другие элементы. Некоторые элементы содержат данные, а другие — смешанное содержание, включая текст и другие элементы. Примеры каждого вида моделей содержания вы найдете дальше в настоящей главе.



Такие ключевые слова схемы DTD, как `DOCTYPE` и `ELEMENT`, должны быть указаны с помощью символов верхнего регистра.

Строка 4 означает закрытие внутренней схемы DTD и указывает обработчику на то, что дальше следует документ XML.

Строка 5 содержит элемент `note` вместе с его текстовым содержанием (`#PCDATA`).

Элемент, содержащий другой элемент

Элементы быть являться контейнерами для других элементов. Таким элементом обычно выступает корневой элемент документа XML. В листинге 4.4 приведен элемент `note`, который содержит элемент `message`. Создайте этот документ и сохраните под именем `message03.xml`.

Листинг 4.4. Элемент, содержащий только один дочерний экземпляр `message03.xml`

```
1: <?xml version = "1.0"?>
2: <note>
3:     <message>Remember to buy milk on the way home from work</message>
4: </note>
```

В данном случае, для того, чтобы создать схему DTD для этого документа XML, вам необходимо указать, что элемент `note` содержит другой элемент, который называется `message`, а не текстовые данные. Однако, после этого вам необходимо объявить элемент `message` вместе с его текстовым содержанием (`#PCDATA`), так как в экземпляре XML должны быть объявлены все его элементы — именно эта часть структуры и подкрепляется с помощью схемы DTD. В листинге 4.5 показан документ XML, дополненный внутренней схемой DTD. Создайте этот документ и сохраните под именем `message04.xml`.

Листинг 4.5. Определение типа документа для элемента, содержащего только один дочерний экземпляр `message04.xml`

```
1: <?xml version = "1.0"?>
2: <!DOCTYPE note      [
3: <!ELEMENT note      ( message )>
4: <!ELEMENT message   ( #PCDATA)>
5:                     ]>
6: <note>
7:     <message>Remember to buy milk on the way home from work</message>
8: </note>
```



В строке 3 объявляется о том, что элемент `note` содержит другой элемент, который называется `message`. В строке 4 объявляется о том, что элемент `message` содержит текстовые данные (`#PCDATA`). Строки с 6 по 8 образуют хорошо оформленный и правильный экземпляр XML. Как вы видите из строки 7, элемент `message` должным образом вложен в элемент `note` (другими словами, элемент `message` является дочерним по отношению к

элементу note). Отступы перед объявлением элемента message используются только для придания документу более аккуратного вида. Использование отступов в программном коде — одно из правил хорошего тона в программировании. Однако добавлять свободное пространство в XML необязательно. Дальше из настоящем уроке вы узнаете, как обработчики XML работают с дополнительными знаками пробелов.

Объявление пустого элемента

Пустые элементы обычно используются для подстановки или же для предоставления требуемых значений атрибутов, которые должным образом не изменяют другие элементы. Подробно об объявлении атрибутов в схемах DTD мы поговорим дальше в настоящем уроке. В строке 3 из листинга 4.6 к документу XML добавлен пустой элемент number. Создайте этот документ и сохраните под именем message05.xml.

Листинг 4.6. Пустой элемент в документе XML message05.xml

```
1:  <?xml version = "1.0"?>
2:  <note>
3:      <number />
4:      <message>Remember to buy milk on the way home from work</message>
5:  </note>
```

В листинге 4.7 приведена внутренняя схема DTD, которая должным образом объявляет пустой элемент. Создайте этот документ и сохраните под именем message06.xml.

Листинг 4.7. Определение типа документа для пустого элемента number-message06.xml

```
1:  <?xml version = "1.0"?>
2:  <!DOCTYPE note [
3:  <!ELEMENT note ( number , message ) >
4:  <!ELEMENT number EMPTY>
5:  <!ELEMENT message ( #PCDATA)>
6:  ]>
7:  <note>
8:      <number />
9:      <message>Remember to buy milk on the way home from work</message>
10: </note>
```

АНАЛИЗ В строке 3 объявляется о том, что элемент note содержит другой элемент, который называется number, за которым следует элемент message. Запятая (,) в модели содержания указывает на то, что элемент следует за другим элементом в порядке их объявления. Обработчик XML интерпретирует строку 3 таким образом, что элемент note содержит другой элемент, который называется number, за которым следует элемент message, причем именно в таком порядке. Экземпляр XML, в котором элемент message указан перед элементом number, в соответствии с подобной схемой DTD считается неправильным.

В строке 4 ключевое слово EMPTY используется в модели содержания элемента number для того, чтобы объявить, что этот элемент пуст. В соответствии со схемой DTD элемент number не может содержать ни текстовые данные, ни другие элементы.



Такое ключевое слово схемы DTD, как EMPTY, должно быть указано с помощью символов верхнего регистра.

Использование ключевого слова ANY схемы DTD

Иногда вы знаете, что определенный элемент не пустой (а значит, содержит текст и другие элементы или и то, и другое), но вы не уверены в том, какая именно модель содержания используется. Ключевое слово ANY схемы DTD можно использовать для того, чтобы объявить содержание подобных элементов. Для того чтобы понять, каким образом ключевое слово ANY влияет на определение структуры XML, добавьте несколько элементов, чтобы сделать структуру немного сложнее. Затем, определив модель содержания с помощью ключевого слова ANY, вы снимите с нее целый ряд ограничений.

Изучите экземпляр XML, приведенный в листинге 4.8, в котором к элементу-контейнеру note добавлен еще один дочерний элемент date. Создайте этот документ и сохраните под именем message07.xml.

Листинг 4.8. Добавление еще одного дочернего элемента к элементу note — message07.xml

```
1: <?xml version = "1.0"?>
2: <note>
3:     <number />
4:     <message>Remember to buy milk on the way home from work</message>
5:     <date />
6: </note>
```

Вы можете расширить объявление элемента note в строке 3 листинга 4.7 для включения нового элемента date. В результате вы получите следующее выражение:

```
3: <!ELEMENT note(number, message, date)>
```

Несмотря на то, что такое представление не очень точное и дает меньше возможностей для проверки правильности, вы также можете использовать ключевое слово ANY для указания того, что элемент note может содержать данные любого типа, в том числе текст и другие элементы, независимо от их конкретных имен и порядка. В листинге 4.9 приведена схема DTD, содержащая подобное объявление. Создайте этот документ и сохраните под именем message08.xml.

Листинг 4.9. Определение содержания для элемента note — message08.xml

```
1: <?xml version = "1.0"?>
2: <!DOCTYPE note [
3: <!ELEMENT note ANY>
4: <!ELEMENT number EMPTY>
5: <!ELEMENT message ( #PCDATA)>
6: <!ELEMENT date EMPTY>
7: ]>
8: <note>
9:     <number />
10:    <message>Remember to buy milk on the way home from work</message>
11:    <date />
12: </note>
```

В строке 3 содержится объявление элемента `note` с использованием ключевого слова `ANY`. Однако для того, чтобы документ считался правильным, также необходимо добавить строку 6, в которой содержится объявление элемента `date` с использованием ключевого слова `EMPTY`. Без этой строки документ XML не будет правильным. Строка 4 налагает те же ограничения на элемент `number`. Вы не можете включать в экземпляр XML элементы, которые не были объявлены в схеме DTD.



Такое ключевое слово схемы DTD, как `ANY`, должно быть указано с помощью символов верхнего регистра.

Модели смешанного содержания

Возможны такие ситуации, когда вам необходимо установить правило, которое разрешало бы элементу содержать некоторую комбинацию текстовых данных и других элементов. Подобные возможности обеспечивает модель смешанного содержания. В листинге 4.10 представлен экземпляр XML, корневой элемент которого содержит текстовые данные, а также определенные вами раньше дочерние элементы. Создайте этот документ и сохраните под именем `message09.xml`.

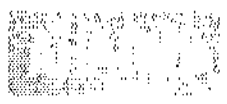
Листинг 4.10. Смешанное содержание элемента `note` — `message09.xml`

```
1: <?xml version = "1.0"?>
2: <note>This is an important note
3:      <number />
4:      <message>Remember to buy milk on the way home from work</message>
5:      <date />
6: </note>
```

Корневой элемент `note`, указанный в строке 2, содержит текст `This is an important note` (Это важная заметка). В листинге 4.11 представлена схема DTD для проверки экземпляра XML. Создайте этот документ и сохраните под именем `message10.xml`.

Листинг 4.11. Проверка правильности элемента с помощью модели смешанного содержания `message10.xml`

```
1: <?xml version = "1.0"?>
2: <!DOCTYPE note [
3: <!ELEMENT note (#PCDATA | number | message | date)*>
4: <!ELEMENT number EMPTY>
5: <!ELEMENT message ( #PCDATA)>
6: <!ELEMENT date EMPTY>
7: ]>
8: <note>This is an important note
9:      <number />
10:      <message>Remember to buy milk on the way home from
         work</message>
11:      <date />
12: </note>
```



В строке 3 объявляется смешанное содержание корневого элемента `note`. Обработчик XML интерпретирует эту строку таким образом, чтобы показать, что элемент `note` может содержать любую комбинацию текстовых данных и перечисленных дочерних элементов. Это действительно достаточно сложная ин-

струкция. Знак "|" на языке схемы DTD означает "или". Спецификация содержания гласит, что элемент `note` может содержать текстовые данные или дочерний элемент `number`, `message` или `date`. Звездочка (*) в конце спецификации содержания означает, что указанные в скобках элементы могут использоваться любое количество раз, а могут не использоваться вообще. Таким образом, использование той же схемы DTD означает, что документ XML, представленный в листинге 4.12, также правильный.

Листинг 4.12. Еще один правильный экземпляр XML с использованием модели смешанного содержания — `message11.xml`

```
1:  <?xml version = "1.0"?>
2:  <!DOCTYPE note [
3:  <!ELEMENT note (#PCDATA | number | message | date)*>
4:  <!ELEMENT number EMPTY>
5:  <!ELEMENT message ( #PCDATA)>
6:  <!ELEMENT date EMPTY>
7:  ]>
8:  <note>This is an important note
9:      <number />now we can no longer control the occurrence of each
       content type
10:     <number />
11:     <message>stuff</message>to remember
12:     <message>Remember to buy milk on the way home from
       work</message>
13: </note>
```

Если вы изучите различия между листингами 4.11 и 4.12, вы увидите, что в них содержится одна и та же внутренняя схема DTD. Однако в листинге 4.12 отсутствует элемент `date`, поскольку модель смешанного содержания требует использования оператора "*" для указания необязательных элементов.

Используя модель смешанного содержания, вы теряете значительную часть контроля над объявляемой структурой документа. Вам придется время от времени использовать модели смешанного содержания, но все-таки их желательно по возможности избегать, особенно если вам необходим контроль над структурой данных в документе.

Подведение итогов по моделям содержания

Вы уже знаете об использовании схемы DTD для объявления содержания элементов. В табл. 4.1 приведены основные сведения о разных способах указания содержания, а также используемый в каждом из этих случаев синтаксис.

Таблица 4.1. Указание содержимого элементов в схемах DTD

Содержание	*Синтаксис*	*Интерпретация*
Элемент(ы)	`<!ELEMENT имя (дочерний_элемент1, дочерний_элемент1)>`	Этот элемент может содержать только дочерние элементы
Смешанное	`<!ELEMENT имя (#PCDATA, дочерний_элемент)*>`	Этот элемент содержит комбинацию текстовых данных и дочерних элементов
EMPTY	`<!ELEMENT имя EMPTY>`	Этот элемент ничего не содержит
ANY	`<!ELEMENT имя ANY>`	Этот элемент может содержать текстовые данные или дочерние элементы

Объявление атрибутов в схеме DTD

Предположим, что вы решили, что пустые элементы `number` и `date` лучше превратить в атрибуты элемента `message`, поскольку они дополняют именно его. Вы помните, что атрибуты модифицируют элементы точно так же, как прилагательные модифицируют существительные. В следующем примере элементы `number` и `date` из листингов 4.8 и 4.9 превращены в атрибуты элемента `message`, как показано в листинге 4.13.

Листинг 4.13. Экземпляр XML с атрибутами `message12.xml`

```
1: <?xml version = "1.0"?>
2: <note>
3:     <message number="10" date="073001">
4:        Remember to buy milk on the way home from work
5:     </message>
6: </note>
```

Теперь в строке 3 содержатся атрибуты `number` и `date` элемента `message`. В схеме DTD для объявления атрибутов используется специальный механизм. Для этого предназначено ключевое слово `ATTLIST`.

Объявления атрибутов представляются в следующем виде:

```
<!ATTList    имя_элемента имя_атрибута1 (тип) значение по умолчанию
                          имя_атрибута2 (тип) значение по умолчанию
```

Существует три фундаментальных типа атрибутов, объявляемых в рамках схемы DTD. К ним относятся следующие:

- строки, указываемые с помощью ключевого слова `CDATA`;
- маркированные атрибуты, указываемые с помощью определенных раньше маркеров;
- атрибуты с перечислением, предлагающие целый список допустимых значений.

Значения атрибутов по умолчанию объявляются для того, чтобы разрешить авторам документов контролировать допустимые значения атрибутов. Соответствующие примеры вы увидите дальше в настоящей главе. Все три стандартных типа атрибутов реализуются с помощью схем DTD, как показано в табл. 4.2.

Таблица 4.2. Стандартные значения атрибутов схемы DTD

Значение	*Интерпретация*
#REQUIRED	Указывает на то, что атрибут должен быть указан
#FIXED	Указывает на фиксированное значение атрибута. Если значение атрибута отличается от объявленного, документ не считается правильным
#IMPLIED	Атрибут необязателен. Это означает, что если для элемента атрибут не указан, приложение при обработке может использовать любое значение (если в этом есть необходимость)

Объявления атрибутов строкового типа

В листинге 4.14 приведена схема DTD, которая используется для проверки экземпляра XML из листинга 4.13.

Листинг 4.14. Определение типа документа с помощью ключевого слова ATTLIST — message13.xml

```
1: <?xml version = "1.0"?>
2: <!DOCTYPE note    [
3: <!ELEMENT note          ( message ) >
4: <!ELEMENT message       ( #PCDATA)>
5: <!ATTLIST message
6:           number   CDATA    #REQUIRED
7:           dateCDATA    #REQUIRED>
8:                  ]>
9: <note>
10:     <message number="10" date="073001">
11:        Remember to buy milk on the way home from work
12:     </message>
13: </note>
```

Анализ В строке 5 указано, что для элемента `message` требуются атрибуты `number` и `date`. Оба атрибута типа `CDATA`, что позволяет использовать в качестве их значения любую строку. Ключевое слово `CDATA` допускает использование любых символов, за исключением <, >, & или ". Ключевое слово `#REQUIRED` свидетельствует о том, что атрибуты `number` и `date` обязательно должны быть предоставлены для элемента `message`.

В строке 10 атрибуту `number` присваивается значение `"10"`, а атрибуту `number` присваивается значение `"073001"`.

Объявления атрибутов типа #FIXED

В листинге 4.15 показан новый атрибут `from` со значением `"Kathy Shepherd"`. Благодаря использованию ключевого слова `#FIXED` вы можете быть уверены, что значение атрибута будет именно таким, как вы того ожидаете. В этом упражнении все сообщения (элемент `message`) поступили от `Kathy Shepherd`.

Листинг 4.15. Использование атрибута from для изменения элемента message — message14.xml

```
1: <?xml version = "1.0"?>
2: <note>
3:     <message number="10" date="073001" from="Kathy Shepherd">
4:        Remember to buy milk on the way home from work
5:     </message>
6: </note>
```

Схема DTD, объявляющая фиксированное значение атрибута, приведена в листинге 4.16.

Листинг 4.16. Правильный экземпляр XML со значением атрибута #FIXED — message15.xml

```
1: <?xml version = "1.0"?>
2: <!DOCTYPE note    [
3: <!ELEMENT note          ( message ) >
4: <!ELEMENT message       ( #PCDATA)>
5: <!ATTLIST message
6:           number   CDATA    #REQUIRED
```

```
7:            dateCDATA    #REQUIRED
8:            fromCDATA    #FIXED   "Kathy Shepherd">
9:                    ]>
10: <note>
11:     <message number="10" date="073001" from="Kathy Shepherd">
12:        Remember to buy milk on the way home from work
13:     </message>
14: </note>
```

В строке 8 объявляется, что атрибут `from` может содержать значение `"Kathy Shepherd"` и ничего больше. Например, если строка 11 будет иметь вид `<message number="10" date="073001" from="Kathy Shepherd">`, обработчик XML будет считать документ неправильным, что приведет к отображению сообщения об ошибке.

Объявления атрибутов типа #IMPLIED

В листинге 4.17 показан атрибут типа `#IMPLIED`. Подобные атрибуты считаются необязательными, поэтому их наличие или отсутствие не влияет на результаты проверки правильности документа.

Листинг 4.17. Правильный экземпляр XML с атрибутом `#IMPLIED` — `message16.xml`.

```
1:  <?xml version = "1.0"?>
2:  <!DOCTYPE note    [
3:  <!ELEMENT note          ( message ) >
4:  <!ELEMENT message       ( #PCDATA)>
5:  <!ATTLIST message
6:            number   CDATA    #REQUIRED
7:            dateCDATA    #REQUIRED
8:            fromCDATA    #FIXED   "Kathy Shepherd"
9:            status   CDATA    #IMPLIED>
10:                   ]>
11: <note>
12:     <message number="10" date="073001" from="Kathy Shepherd"
         status="urgent">
13:        Remember to buy milk on the way home from work
14:     </message>
15: </note>
```

В строке 9 объявлен атрибут `status` типа `#IMPLIED`. Поскольку этот атрибут предполагаемый (от англ. *implied*) или необязательный, документ считается правильным, независимо от наличия этого атрибута в элементе `message`. Например, если строка 12 будет иметь вид `<message number="10" date="073001" from="Kathy Shepherd" status="urgent">`, документ будет считаться правильным.

Атрибуты маркированных типов в схемах DTS

При использовании атрибутов маркированных типов вы можете наложить определенные ограничения на значения атрибутов. Например, вы можете использовать уникальные идентификаторы для каждого элемента или разрешить атрибутам принимать только одно из двух допустимых значений. В табл. 4.3 перечислены четыре различных маркированных типов атрибутов, которые могут использоваться в схемах DTD.

Таблица 4.3. Маркированные типы атрибутов для схем DTD

Значение	*Интерпретация*
ID	Уникальным образом идентифицирует элемент
IDREF	Указывает на элементы, содержащие атрибуты `ID`
ENTITIES	Ссылка на внешний элемент
NMTOKEN	Содержит буквы, цифры, точки, знаки подчеркивания, переносы и двоеточия, но не пробелы

Использование атрибутов типов ID и IDREF

В листинге 4.18 приведен пример экземпляра XML, который можно проверить с использованием атрибутов `ID` и `IDREF`. В этом примере документ XML `note` расширен таким образом, что в него включены несколько новых элементов `message` с уникальными атрибутами `number` и `date`, а также элементы `outcome`, связанные с элементами `message` с помощью атрибута `msg`. Этот экземпляр XML очень хорошо оформлен.

Листинг 4.18. Расширенный документ XML note — message17.xml

```
1:  <?xml version = "1.0"?>
2:  <note>
3:      <message number="a1" from="Kathy Shepherd">
4:         Remember to buy milk on the way home from work
5:      </message>
6:      <message number="a2" from="Greg Shepherd">
7:         I need some help with my homework
8:      </message>
9:      <message number="a3" from="Kristen Shepherd">
10:         Please play Scrabble with me tonight
11:     </message>
12:
13:     <outcome msg="a1">
14:     milk was past its best before date
15:     </outcome>
16:     <outcome msg="a1">
17:     went to a different store
18:     </outcome>
19:     <outcome msg="a2">
20:     homework finished early
21:     </outcome>
22: </note>
```

Для прояснения назначения этого документа XML представьте себе такую ситуацию: приложение сохраняет короткие сообщения, получаемые на протяжении дня с помощью сотового телефона, пейджера или электронной почты. Каждому сообщению присваивается уникальный идентификатор (например, атрибут `number`). Это же приложение сохраняет элементы `outcome`, которые могут сохраняться, а могут и не сохраняться (если для них не указаны атрибуты типа `#REQUIRED`). Если эти элементы должны записываться, их нужно связать с определенным элементом `message` (с помощью атрибута `msg`), чтобы их обработать в дальнейшем.

Тогда возникает необходимость достаточно сложной проверки, которая, однако, легко реализуется благодаря использованию атрибутов `ID` и `IDREF`. Соответствующая схема DTD приведена в листинге 4.19.

Листинг 4.19. Проверка правильности с использованием атрибутов ID и IDREF — message18.xml

```
1:  <?xml version = "1.0"?>
2:  <!DOCTYPE note     [
3:  <!ELEMENT note          ( message+, outcome+ ) >
4:  <!ELEMENT message  ( #PCDATA)>
5:  <!ATTLIST message
6:                  number   ID       #REQUIRED
7:                  from CDATA    #REQUIRED>
8:  <!ELEMENT outcome  (#PCDATA)>
9:  <!ATTLIST outcome
10:                 msg  IDREF    #IMPLIED>
11:                 ]>
12:  <note>
13:      <message number="a1" from="Kathy Shepherd">
14:         Remember to buy milk on the way home from work
15:      </message>
16:      <message number="a2" from="Greg Shepherd">
17:         I need some help with my homework
18:      </message>
19:      <message number="a3" from="Kristen Shepherd">
20:         Please play Scrabble with me tonight
21:      </message>
22:
23:      <outcome msg="a1">
24:      milk was past its best before date
25:      </outcome>
26:      <outcome msg="a1">
27:      went to a different store
28:      </outcome>
29:      <outcome msg="a2">
30:      homework finished early
31:      </outcome>
32:  </note>
```

АНАЛИЗ В строке 3 представлена иная модель содержания, чем использовалась раньше в настоящей главе. В частности, после элементов `message` и дочерних элементов `outcome` в определении содержания для элемента `note` используется оператор +. Это оператор вхождения, который означает, что дочерний элемент должен содержаться в элементе-контейнере один или несколько раз. В данном случае строка 3 интерпретируется таким образом, что элемент `note` содержит один или несколько элементов `message`, за которыми (вы помните, что запятая всегда указывает должный порядок) следует один или несколько элементов `outcome`.

Строка 4 указывает на то, что элемент `message` содержит текстовые данные.

Объявление списка атрибутов для элемента `message` начинается со строки 5. В строке 6 указывается, что необходимый атрибут `number` относится к типу `ID`. Такой тип атрибута указывает на то, что каждый атрибут `number` должен быть уникальным. Если один атрибут `number` будет использоваться несколько раз, документ будет считаться неправильным.

СОВЕТ

> Атрибут типа `ID` должен начинаться с буквы, двоеточия (:) или знака подчеркивания (_). В каждой инструкции может содержаться только один атрибут типа `ID`.

Строка 7 заимствована из предыдущих примеров и указывает, что элемент message должен содержать строковый атрибут from.

В строке 8 объявляется элемент outcome, содержащий строковые данные.

В строке 10 атрибут msg объявляется как атрибут типа IDREF для элемента outcome. Это означает, что значение атрибута msg элемента outcome связано со значением соответствующего атрибута number элемента message. В нашем примере элемент message с атрибутом number, равным "a1", Remember to buy milk on the way home from work (Не забудь приобрести молоко по пути с работы домой), связано с элементом outcome, равным milk was past its best before date (молоко уже прокисло). Точно так же, элемент outcome с атрибутом msg="a2", равный homework finished early (домашняя работа выполнена рано), связан с элемент message с атрибутом number, равным "a2", I need some help with my homework (мне нужна помощью для выполнения домашней работы). Вы могли заметить, что с элементом message с атрибутом number, равным "a3", не связан ни один элемент outcome. Это вполне приемлемая ситуация, поскольку в строке 10 атрибут msg был объявлен как атрибут типа #IMPLIED, а значит, необязательный.

Использование элементов ENTITY в схемах DTD

С точки зрения разметки элементы считаются контейнерами. Элементы ENTITY — это ссылки, которые могут заменить другими. Читатели, знакомые с HTML, знают, что в этом языке также существуют несколько видов ссылок, такие как   , которая означает для броузера, что надо поставить один знак неразрывного пробела. В XML используется несколько видов ссылок. Первый тип, с которым вы познакомитесь, позволяет реализовать в документе XML подстановку строк. Объявление и определение ссылок, или подстановочных элементов, осуществляются в схеме. Когда обработчик XML доходит до подстановочного элемента, он заменяется соответствующими, заранее определенными данными. На наличие подстановочного элемента обработчику указывает специальный знак — &. Все подстановки символов представляются в следующей форме:

&ссылка;

Указанные в начале знак & и в конце двоеточие ограничивают строку, которая соответствует ссылке. При нахождении ссылки в схеме DTD парсер считывает строку, которой она соответствует, и должным образом раскрывает или разрешает ссылку.

В уроке 7 вы подробно познакомитесь с использованием подстановочных элементов в XML. Подстановочные элементы можно использовать несколькими способами. Вы узнаете, как использовать подстановочный элемент для ссылки на сохраненную переменную в схеме DTD. Объявление подставочного элемента для представления строковых данных приведено в листинге 4.20.

Листинг 4.20. Подстановка элементов в XML с использованием определений типов документов DTD message19.xml

```
1:  <?xml version = "1.0"?>
2:  <!DOCTYPE note     [
3:  <!ELEMENT note          ( message+, outcome+ ) >
4:  <!ELEMENT message  ( #PCDATA)>
5:  <!ATTLIST message
6:                  number   ID       #REQUIRED
7:                  from CDATA    #REQUIRED>
8:  <!ELEMENT outcome  (#PCDATA)>
9:  <!ATTLIST outcome
10:                 msg  IDREF    #IMPLIED>
11:  <!ENTITY today    "073001">
```

```
12:                     ]>
13: <note>
14:     <message number="a1" from="Kathy Shepherd">
15:        &today; - Remember to buy milk on the way home from work
16:     </message>
17:     <message number="a2" from="Greg Shepherd">
18:        &today; - I need some help with my homework
19:     </message>
20:     <message number="a3" from="Kristen Shepherd">
21:        &today; - Please play Scrabble with me tonight
22:     </message>
23:     <outcome msg="a1">
24:     milk was past its best before date
25:     </outcome>
26:     <outcome msg="a1">
27:     went to a different store
28:     </outcome>
29:     <outcome msg="a2">
30:     homework finished early
31:     </outcome>
32: </note>
```



В строке 11 содержится объявление подстановочного элемента `today`, содержащего строку `"073001"`. При обнаружении ссылки `&today;` в строках 15, 18 и 21 парсер выполняет соответствующую подстановку.

На рис. 4.3 показана обработки подстановочного элемента в Microsoft Internet Explorer.

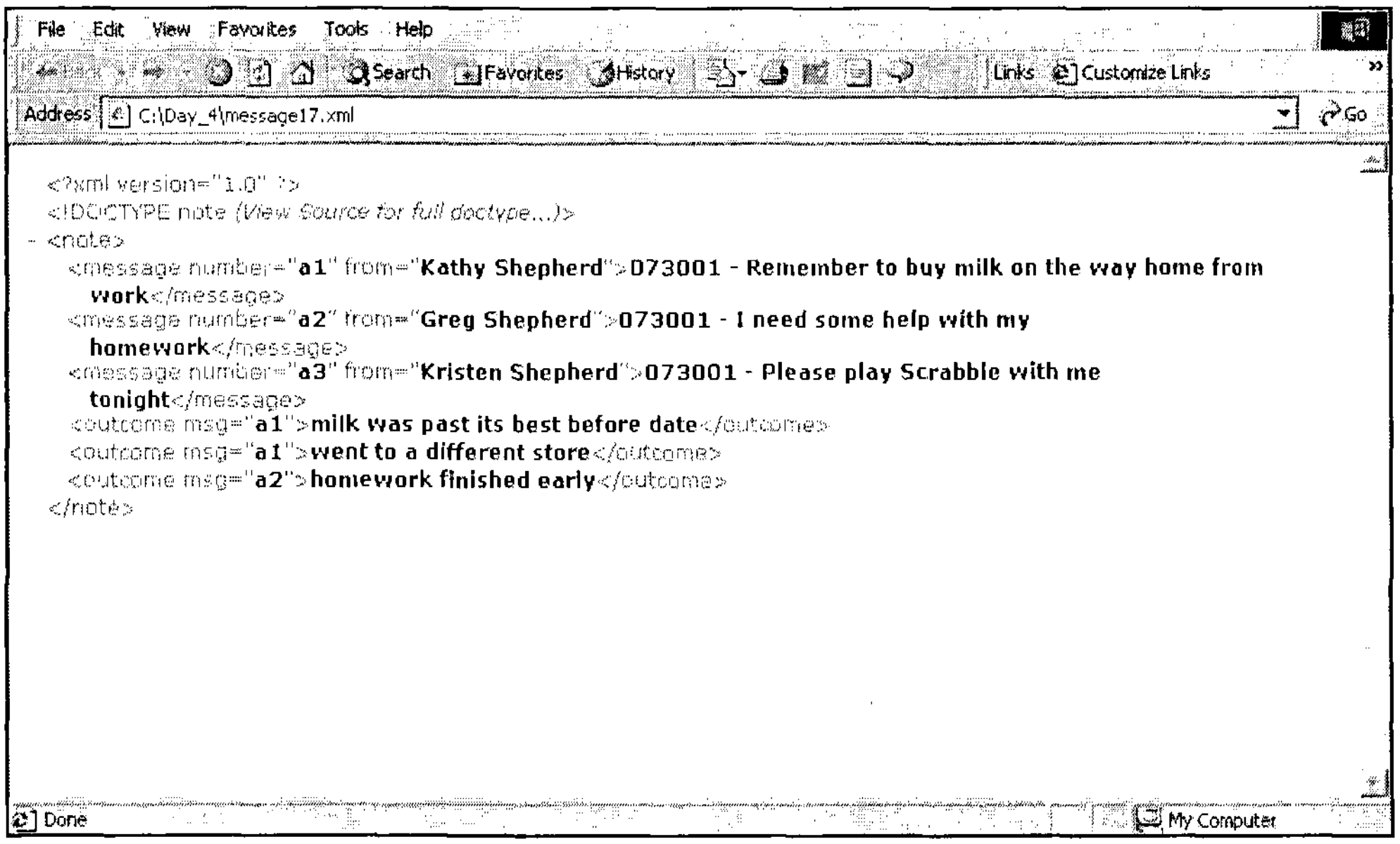


Рис. 4.3. Обработанный документ, содержащий подстановку строк

Использование атрибутов типа NMTOKEN

В XML используется несколько видов ограничений типа данных. Тип `NMTOKEN` позволяет наложить ограничение на допустимые символы в содержании XML. В частно-

сти, тип NMTOKEN ограничивает данные до тех же правил, которые используются и в соглашениях о допустимых именах элементов XML, которые накладывают незначительные ограничения. Несмотря ни на что, тип NMTOKEN разрешает использовать только буквы, цифры, точки, переносы, двоеточия и знаки подчеркивания. Использование атрибута типа NMTOKEN проиллюстрировано в листинге 4.21.

Листинг 4.21. Использование атрибутов типа NMTOKEN в схеме DTD — message20.xml

```
1:  <?xml version = "1.0"?>
2:  <!DOCTYPE note     [
3:  <!ELEMENT note          ( message+, outcome+ ) >
4:  <!ELEMENT message  ( #PCDATA)>
5:  <!ATTLIST message
6:                  number   ID           #REQUIRED
7:                  from CDATA         #REQUIRED
8:                  phone    NMTOKEN  #REQUIRED>
9:  <!ELEMENT outcome  (#PCDATA)>
10:  <!ATTLIST    outcome
11:                 msg  IDREF    #IMPLIED>
12: <!ENTITY today    "073001">
13:                   ]>
14: <note>
15:     <message number="a1" from="Kathy Shepherd" phone="720-555-1212">
16:        &today; - Remember to buy milk on the way home from work
17:     </message>
18:     <message number="a2" from="Greg Shepherd" phone="720-555-1234">
19:        &today; - I need some help with my homework
20:     </message>
21:     <message number="a3" from="Kristen Shepherd" phone="720-555-4321">
22:        &today; - Please play Scrabble with me tonight
23:     </message>
24:     <outcome msg="a1">
25:     milk was past its best before date
26:     </outcome>
27:     <outcome msg="a1">
28:     went to a different store
29:     </outcome>
30:     <outcome msg="a2">
31:     homework finished early
32:     </outcome>
33: </note>
```

В строке 8 объявляется атрибут phone элемента message как атрибут типа NMTOKEN. Как видите, документ является правильным; однако, если значение атрибута phone в строке следующим образом: <message number="a1" from="Kathy Shepherd" phone="720 555 1212">, то документ уже не будет считаться правильным, поскольку объявление атрибута как атрибута типа NMTOKEN запрещает использование пробелов.

Использование в схемах DTD атрибутов типов с перечислением

Атрибуты типов с перечислением содержат список допустимых значений, из которых можно выбирать любое. Для того что обеспечить правильность документа, атрибут должен иметь значение, указанное в списке; в противном случае документ будет считаться неправильным. Значения из списка указываются с помощью знака "|", ко-

торый воспринимается обработчиком XML как логический оператор "или". В строке 8 листинга 4.22 показан атрибут `alert` вместе с тремя допустимыми значениями (`low`, `normal`, `urgent`) и заданным по умолчанию значением `normal`.

Листинг 4.22. Атрибут с перечислением `message21.xml`

```
1: <?xml version = "1.0"?>
2: <!DOCTYPEnote     [
3: <!ELEMENTnote          ( message+, outcome+ ) >
4: <!ELEMENTmessage  ( #PCDATA)>
5: <!ATTLISTmessage
6:                 number   ID           #REQUIRED
7:                 fromCDATA         #REQUIRED
8:                 alert    ( low | normal | urgent) "normal">
9: <!ELEMENToutcome  (#PCDATA)>
10: <!ATTLIST    outcome
11:                msg  IDREF        #IMPLIED>
12: <!ENTITY today    "073001">
13:                ]>
14: <note>
15:      <message number="a1" from="Kathy Shepherd" alert="low">
16:         &today; - Remember to buy milk on the way home from work
17:      </message>
18:      <message number="a2" from="Greg Shepherd" alert="urgent">
19:         &today; - I need some help with my homework
20:      </message>
21:      <message number="a3" from="Kristen Shepherd">
22:         &today; - Please play Scrabble with me tonight
23:      </message>
24:      <outcome msg="a1">
25:      milk was past its best before date
26:      </outcome>
27:      <outcome msg="a1">
28:      went to a different store
29:      </outcome>
30:      <outcome msg="a2">
31:      homework finished early
32:      </outcome>
33: </note>
```

В строках 15 и 18 объявлены значения `low`, `normal`, `urgent` для атрибута `alert`. Вы можете только представить, каким образом нужно спроектировать приложение для того, чтобы обращаться с элементами `message`, зависящими от значения атрибута `alert`. Элемент `message` начинается со строки 21 и обрабатывается приложением так, как будто значение его атрибута `alert` равно `normal`, что соответствует значению, заданному по умолчанию.

Строку 8 можно изменить таким образом, что из нее будет исключено значение атрибута `alert` по умолчанию, которое заменяется ключевым словом `#IMPLIED`.

```
8: alert( low | normal | urgent) #IMPLIED>
```

В этом случае атрибут `alert` оказывается необязательным, поэтому при его отсутствии значение по умолчанию не предполагается.

Индикаторы вхождения и объявления последовательностей в схемах DTD

Хотя я не упоминал их явным образом, вы уже познакомились с несколькими индикаторами вхождения, а также объявлениями последовательностей, которые являются частью схем DTD в примерах этого урока. Если вы вводили все эти приложения, вы уже должны были немного разобраться со взаимосвязями различных элементов схемы DTD. Например, вы уже знаете, к чему приводит использование таких операторов, как "+", "*","|" и "," в описаниях моделей содержания. В табл. 4.4 перечислены все специальные индикаторы DTD, используемые для обозначения последовательностей и вхождений.

Таблица 4.4. Индикаторы последовательностей и вхождений для схем DTD

Символ	*Пример*	*Интерпретация*
,	(a, b, c)	Этот оператор последовательности разделяет члены списка, когда необходимо последовательное использование членов списка (сначала `a`, затем `b`, а после этого — `c`)
|	(a|b|c)	Этот оператор выбора разделяет члены списка, когда необходимо использовать только один из членов списка (`a`, `b` или `c`)
	date	Отсутствие какого-либо символа указывает на необходимое вхождение одного и только одного элемента `date`
?	subject?	Этот символ указывает на необязательное вхождение (нуль или один раз) элемента `subject`
+	paragraph+	Этот символ указывает на обязательное и повторяющееся вхождение (один или несколько раз) элемента `paragraph`
*	brother*	Этот символ указывает на необязательное и повторяющееся вхождение (нуль или несколько раз) элемента `brother`

Индикаторы последовательностей иногда называют соединителями (connector), так как они служат для соединения или установления взаимосвязи между двумя или большим числом элементов. В листинге 4.23 приведен экземпляр XML вместе со схемой DTD, в которой используются все индикаторы последовательностей и вхождений.

Листинг 4.23. Объявление типа документа с использованием индикаторов последовательностей и включений `message22.xml`

```
1:  <?xml version="1.0"?>
2:  <!DOCTYPE note                [
3:  <!ELEMENT note                ( message, source?, outcome)+>
4:  <!ELEMENT message    (#PCDATA)>
5:  <!ELEMENT source        EMPTY>
6:  <!ELEMENT outcome    (action*)>
7:  <!ELEMENT action        (#PCDATA)>
8:  <!ATTLIST message
9:                              number    ID          #REQUIRED
10:                              from        CDATA   #REQUIRED>
11: <!ATTLIST    source
12:                            delivery  (phone | inperson | email) "email">
13:                                      ]>
14: <note>
```

```
15: <message number="call_01" from="Kathy Shepherd">
16:     arrived for lunch date
17: </message>
18: <source delivery="inperson"/>
19: <outcome>
20:     <action>went to restaurant</action>
21:     <action>drove in my car</action>
22: </outcome>
23: <message number="call_02" from="Kristen Shepherd">
24: </message>
25: <source delivery="phone"/>
26: <outcome>
27:     <action>returned call</action>
28: </outcome>
29: <message number="call_03" from="Kathy Shepherd">
30: </message>
31: <source />
32: <outcome>
33:     <action>reply to E-Mail</action>
34: </outcome>
35: <message number="call_04" from="Greg Shepherd">
36:     reminder to book fishing trip
37: </message>
38: <outcome>
39: </outcome>
40: </note>
```



В строке 3 определяется содержание элемента `note` для одного или нескольких наборов элементов `message`, `source` и `outcome`, строго в указанном порядке. Если элемент `source` есть, он может быть только один. Действительно, в каждом наборе может быть только по одному элементу `message`, `source` и `outcome`. В экземпляре может присутствовать только один набор, хотя за его пределами может существовать любое количество наборов.

В строке 4 указывается, что элемент `message` содержит только допустимые текстовые данные. Строка 16 показана как пример данных `#PCDATA`, содержащихся в подобном элементе `message`.

В строке 5 объявляется пустой элемент `source`. Из строк 18, 35 и 31 вы можете видеть, что элементы `source` предполагаются пустыми.

Из строки 6 вы можете видеть, что элемент `outcome` содержит любое количество элементов `action`, а может и не содержать их вовсе.

Элемент `action` содержит текстовые данные, что соответствует объявлению `#PCDATA` в строке 7.

В строке 8 начинается объявление атрибутов элемента `message`. Таких атрибута всего два — `number` и `from`. В строке 9 объявляется, что атрибут `number` необходим и является атрибутом типа `ID`; это означает, что каждый подобный атрибут уникален. В строке 9 объявляется, что атрибут `from` необходим и может содержать текстовые данные.

Объявление атрибута `delivery` элемента `source` начинается со строки 11. В строке 12 указано, что каждый из атрибутов может принимать одно из следующих значений: `phone` и `inperson`, или `email`. Если атрибут `delivery` в рамках элемента `source` не указан, приложение использует значение по умолчанию — `email`.

Строки с 14 по 40 содержат правильный экземпляр XML, который удовлетворяет всем ограничениям, налагаемым с помощью схемы DTD. Вы можете видеть, что элементы `message`, `source` и `outcome` повторяются документе неоднократно, за исклю-

чением того, что в последнем наборе элемент `outcome` отсутствует. Такое отсутствие допустимо, поскольку после имени элемента `source` в строке 3 указан знак `?`.

В строках 18 и 25 атрибуты `delivery` соответствующих элементов `source` объявляются равными `"inperson"` и `"phone"` соответственно. В строке 31 атрибут `delivery` элемента `source` отсутствует, поэтому приложение использует указанное значение по умолчанию — `email`.

Анализ правильных экземпляров XML

Начиная с урока 2, вы анализировали хорошо оформленные документы, просматривая их с помощью броузера Microsoft Internet Explorer версии 5.0 или более поздней. Это возможно благодаря наличию в Internet Explorer встроенного парсера XML. Скоро я предложу вам загрузить несколько созданных в настоящей главе документов в Internet Explorer для просмотра результатов. Начните с документа, сохраненного под именем `message19.xml` (листинг 4.20) и посмотрите, получите ли вы результат, приведенный раньше на рис. 4.3. Вы должны помнить, что этот пример иллюстрирует подстановку строк. Если вы не получили результат, приведенный на рис. 4.3, сравните содержимое документа с приведенным в книге листингом строка за строкой, чтобы найти ошибку и исправить ее. Получив необходимые результаты при просмотре файла `message19.xml`, поэкспериментируйте еще с несколькими созданными сегодня примерами.

В уроке дня 2-го вы также видели результаты анализа плохо оформленного документа с помощью Microsoft Internet Explorer 5.0. В частности броузер отобразил сообщение об ошибке, содержащее сведения о причине проблемы: несоответствие открывающего и закрывающего дескрипторов, пропущенная угловая скобка или неправильно оформленный элемент. Все созданные вами в настоящем уроке примеры, содержащие схемы DTD, хорошо оформлены и правильны. Таким образом, вам необходимо создать неправильно оформленный экземпляр XML для просмотра в окне броузера Internet Explorer сообщения об ошибке. Загрузите в редактор файл `message15.xml` и измените его таким образом, чтобы он совпадал с листингом 4.24. Если вы не сохранили файл `message15.xml`, просто скопируйте в текстовый редактор содержание листинга 4.24

Листинг 4.24. Неправильный документ XML `message23.xml`

```
1:  <?xml version = "1.0"?>
2:  <!DOCTYPE note    [
3:  <!ELEMENT note          ( message ) >
4:  <!ELEMENT message       ( #PCDATA)>
5:  <!ATTLIST message
6:            number   CDATA    #REQUIRED
7:            dateCDATA    #REQUIRED
8:            fromCDATA    #FIXED   "Kathy Shepherd">
9:                    ]>
10: <note>
11: <!-- объявление фиксированного значения
    атрибута в соответствии со схемой DTD -->
12:      <message number="10" date="073001" from="someone else">
13:         Remember to buy milk on the way home from work
14:      </message>
15: </note>
```

Изменения, которые вы внесли в файл `message15.xml`, предполагают изменение значения атрибута `from` элемента `message` в строке 12 таким образом, что оно уже не было фиксированным значением `"Kathy Shepherd"` (на что указывал тип атрибута `#FIXED`),

как до этого указывалось в схеме DTD. Измените значение атрибута `from` на `"someone else"` (кого-нибудь другого) или что-нибудь подобное. Это больше не имеет значения, поскольку в строке 8 значение атрибута `from` не объявляется как строго определенное.

Добавьте комментарий, показанный в строке 11, чтобы вы могли легко контролировать внесенные изменения при просмотре результатов в окне броузера Internet Explorer 5.0. Закончив, сохраните файл под именем `message23.xml` и проанализируйте его с помощью Internet Explorer 5.0. После загрузки файла `message23.xml` в Internet Explorer 5.0 вы должны увидеть результат, похожий на приведенный на рис. 4.4.

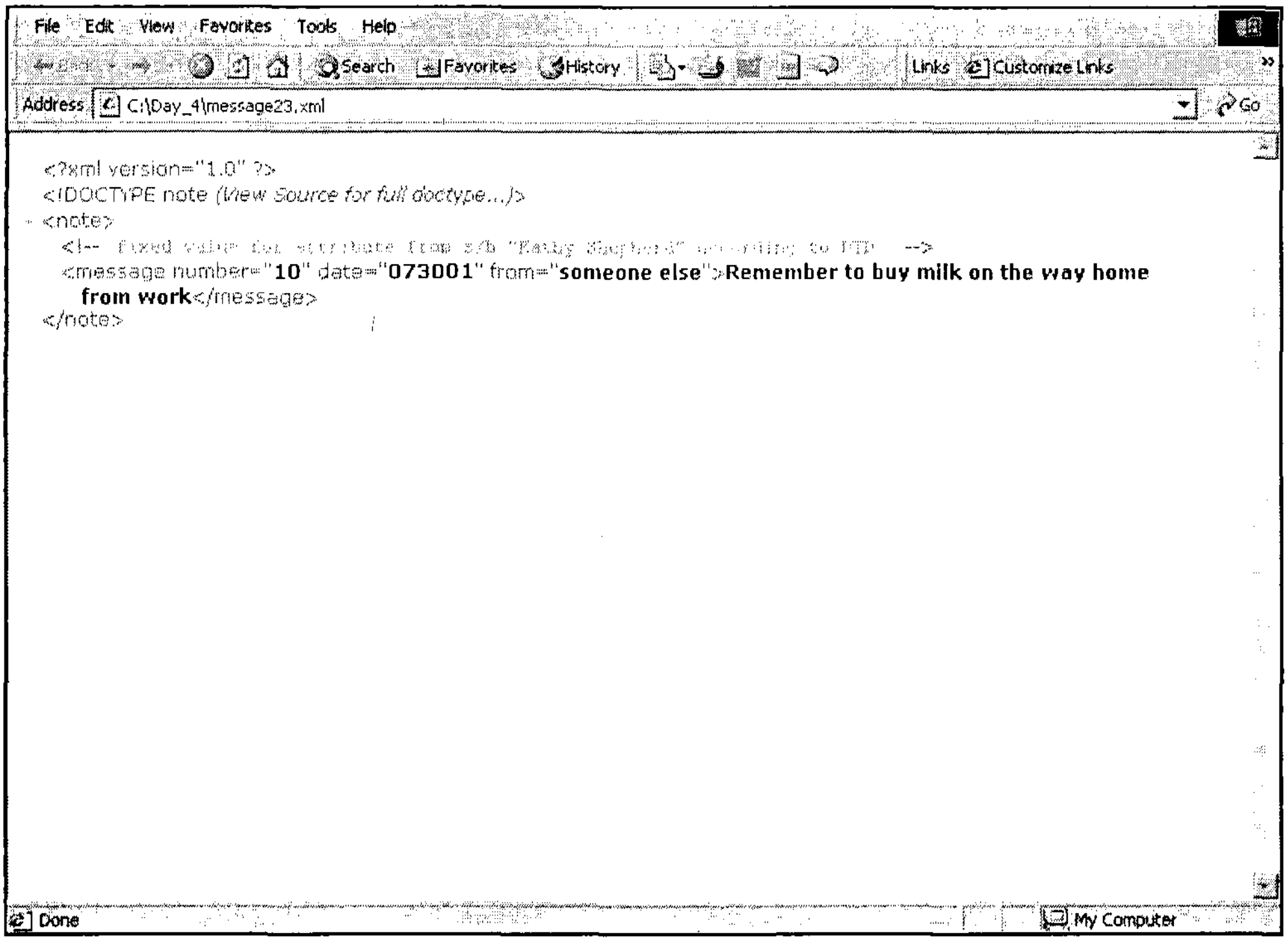


Рис. 4.4. Неправильный документ XML, отображенный в окне Internet Explorer 5.0

Вы знаете расположение ошибки в документе, посмотрите, этот ли результат вы ожидали? Вероятно, нет! Действительно, поскольку вам известно, что значение атрибута `from` элемента `message` в строке 12 неправильно, разве вы не ожидаете от Internet Explorer 5.0 сообщения об ошибке? Подобного сообщения вы не получаете. Действительно, глядя на полученный результат, можно подумать, что ваш документ был обработан совершенно нормально. На то, что с документом что-то не так, указывает только ваш комментарий.

Какие же выводы можно сделать о парсере, входящем в состав Internet Explorer 5.0?

Это упражнение подтвердило тот факт, что парсер из Internet Explorer 5.0 не позволяет проверять все существующие схемы. Internet Explorer не обеспечивает проверку документ XML с использованием схемы DTD; схема DTD просто игнорируется. Версия парсера MSXML, входящая в состав Internet Explorer 5.0, сообщает, что с документом все нормально, представляя его в виде дерева. В действительности ваш документ хорошо оформлен, но не является правильным согласно встроенной схеме DTD. Internet Explorer не удалось провести проверку документа XML с помощью схемы DTD; программа просто сообщила о том, что документ хорошо оформлен.

Таким образом, для проверки документов XML с использованием схемы DTD вам необходим соответствующий парсер. Подобный парсер поставляется в виде COM-объекта вместе с Internet Explorer, хотя и не используется броузером. Для получения доступа к этому парсеру вам следует создать объект парсера в броузере с помощью сценария (например, используя JavaScript или VBScript). Вы научитесь создавать сценарии для доступа к парсеру MSXML в уроке дня 12-го "Объектная модель документа XML — DOM". А пока вы можете загрузить сценарий JavaScript или VBScript, который выполнит все остальные действия за вас. Для того чтобы это сделать, направьте свой броузер к Web-узлу MSDN (Microsoft Developer Network) по адресу `http://msdn.microsoft.com/downloads/samples/internet/xml/xml_validator/` и выполните указания по загрузке и установке сценариев XML Validator. Загрузятся две версии сценария XML Validator, написанные на JavaScript и VBScript, любую из которых вы сможете запускать с помощью Internet Explorer 5.0. Если вы запустите один из этих сценариев (файл `validate_js.htm` или `validate_vbs.htm`), вы должны увидеть результат, приведенный на рис. 4.5.

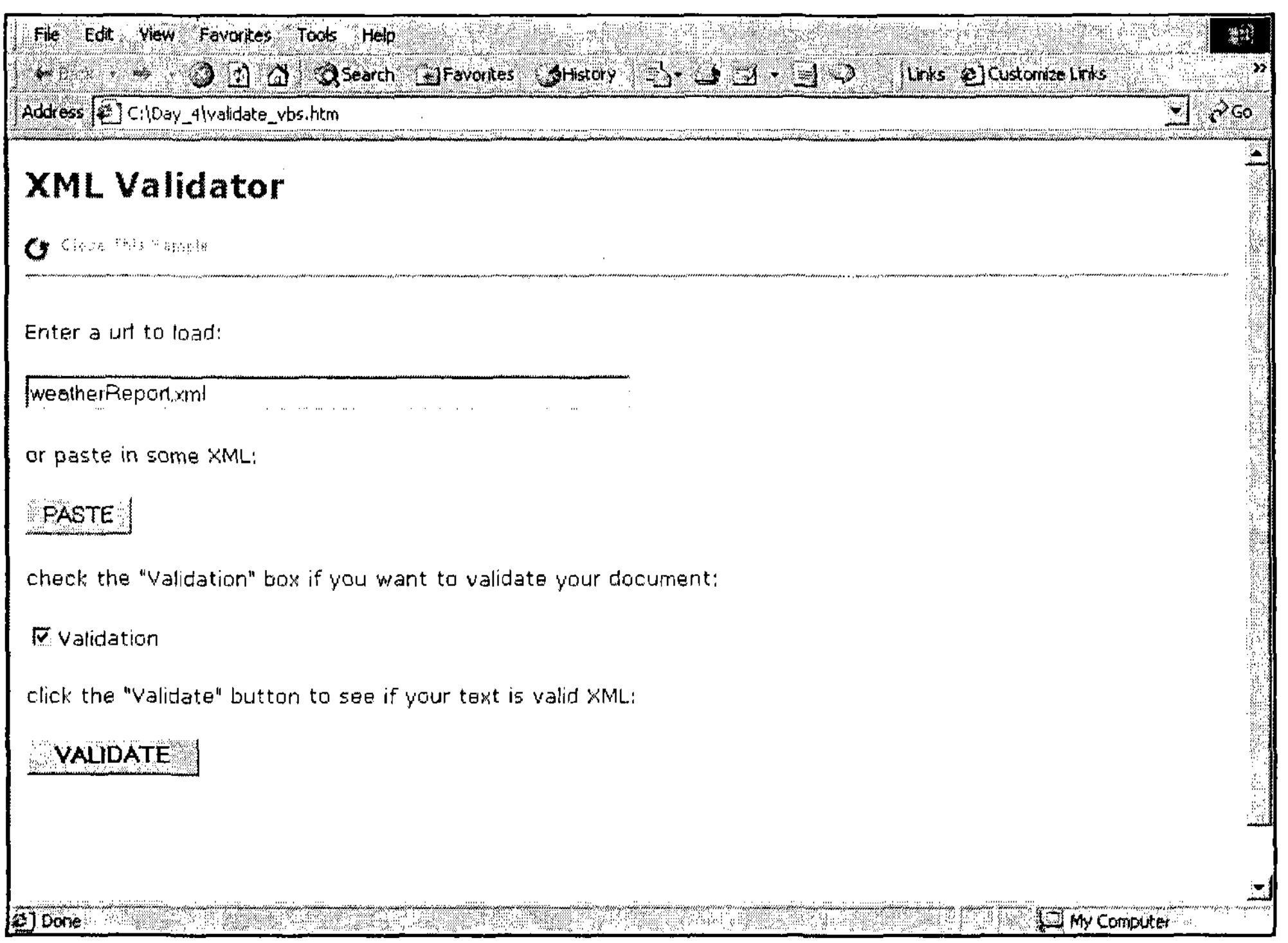


Рис. 4.5. Сценарий XML Validator, загруженный с Web-узла компании Microsoft и отображенный в окне Internet Explorer 5.0

Прежде чем пытаться проверить один из примеров, созданных при изучении материала настоящей главы, переместите файл сценария `validate_js.htm` или `validate_vbs.htm` в папку, с которой вы работаете. Это позволит вам просто указывать имена файлов, а не полный путь к ним.

Когда файл сценария и созданные вами файлы XML будут находиться в одной папке, обработайте некоторые из них, чтобы увидеть результаты. На рис. 4.6 приведен результат проверки файла `message19.xml` из листинга 4.20. Для того чтобы получить именно такой результат, как показано на рисунке, вам необходимо раскрыть каждый узел документа,

просто щелкая на нем. В результате будут отображены дочерние элементы и текстовые данные. Когда весь экземпляр раскроется, структура, представленная на экране, будет полностью соответствовать древовидной структуре, описываемой документом.

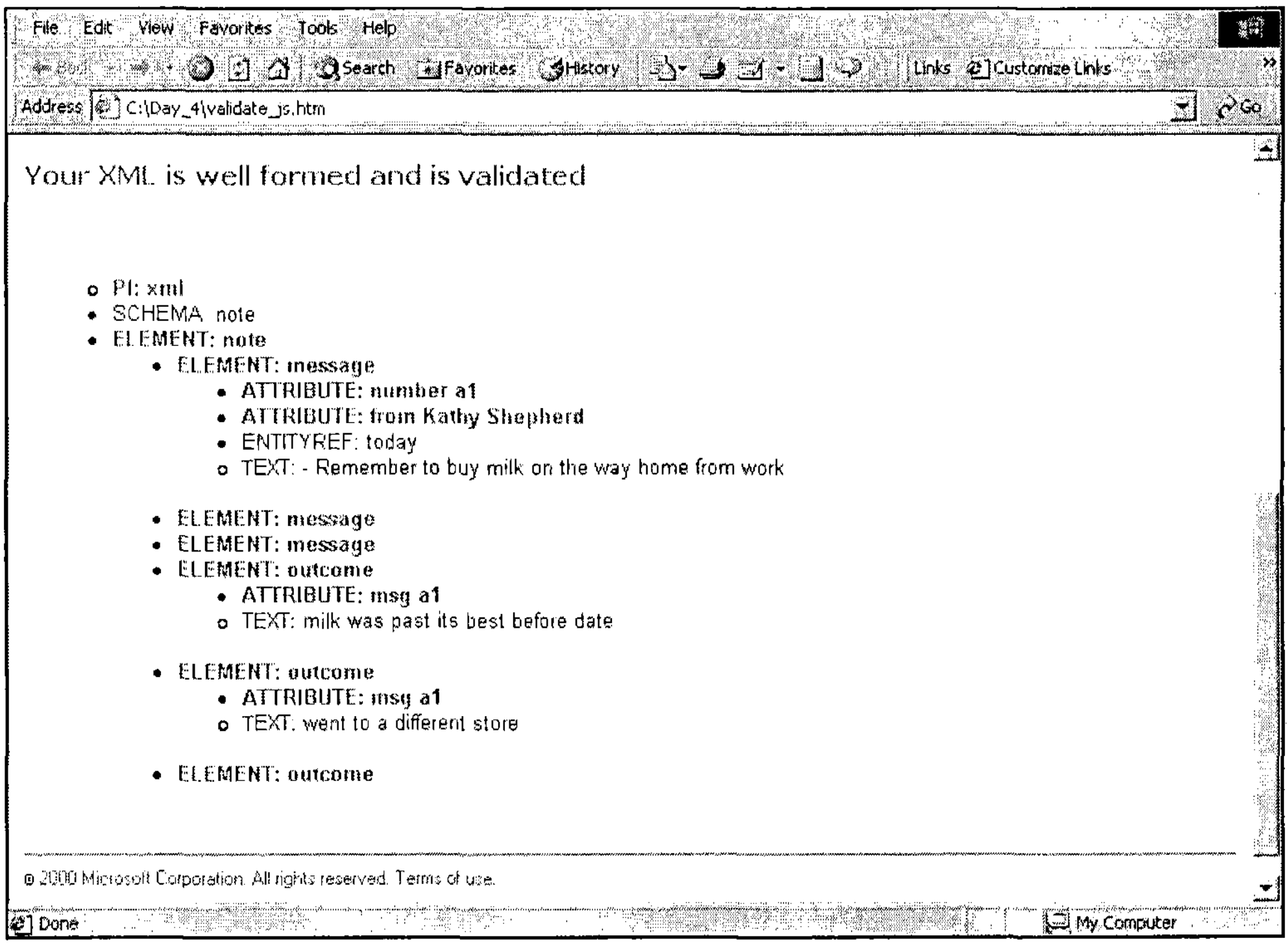


Рис. 4.6. Правильный документ XML (`message19.xml`) успешно прошел проверку с помощью сценария XML Validator

Теперь, когда вы увидели результаты анализа хорошо оформленных и правильных документов, попробуйте проанализировать документ, в который вы специально внесли ошибку, показанный в листинге 4.24 (и сохраненный под именем `message23.xml`). Результат выполнения этого упражнения приведен на рис. 4.7.

Сообщение об ошибке, сгенерированное парсером, гласит о том, что значение атрибута `from` не соответствует фиксированному значению `"Kathy Shepherd"`, указанному в схеме DTD. Вы уже знаете, что именно эту ошибку вы и "заложили" в документ. Документ хорошо оформлен, но больше не является правильным в соответствии с ограничениями, наложенными на него с помощью схемы DTD. Продолжайте экспериментировать с несколькими другими документами, созданными сегодня, внося в них ошибки и просматривая полученные результаты после обработки документов парсером.

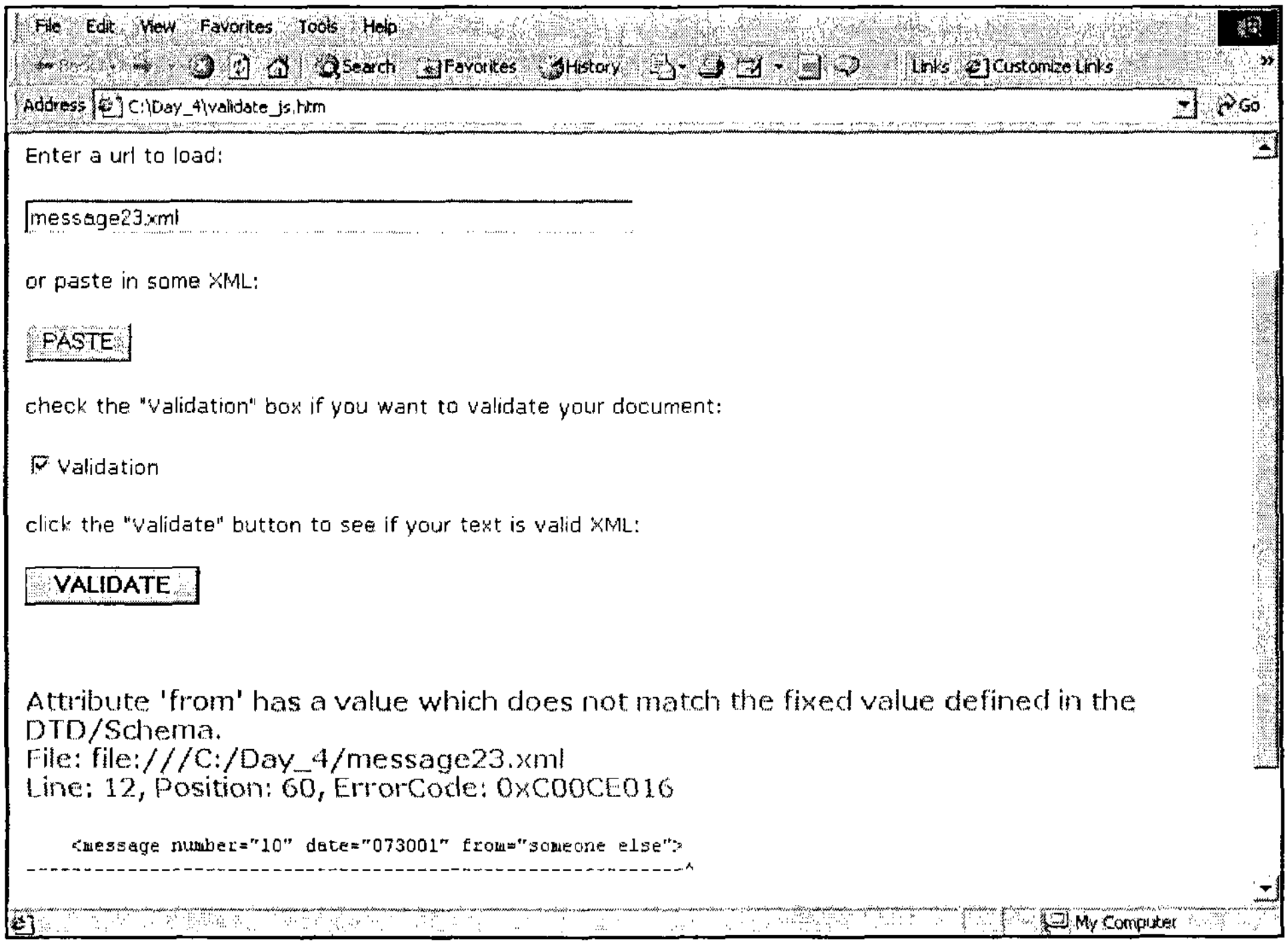


Рис.4.7. Неправильный документ XML (`message23.xml`) не прошел проверку с помощью сценария XML Validator

Объявление внешних схем DTD

Сегодня вы уже узнали, как создать ряд мощных ограничений, используя внутренние схемы DTD. Теперь вы изучите, какие именно изменения необходимо внести в объявление типа документа, чтобы связать документ XML со внешней схемой DTD. Внешняя схема DTD может практически полностью соответствует любой из созданных вами сегодня внутренних схем. Основное различие состоит в том, что схема DTD сохраняется в виде отдельного файла, а не как часть экземпляра XML. Парсер, который в состоянии определить связь между документом XML и схемой DTD, хранящимися в виде различных файлов, может проверить, выполняются ли все ограничения, налагаемые схемой DTD. Для упрощения подобного положения вы будете хранить документы XML, схемы DTD и сценарии парсера в одной папке на жестком диске вашего компьютера.

Объявление типа документа, используемое вами до настоящего времени, представляется в следующем виде:

```
<!DOCTYPE    корневой_элемент [
<!ELEMENT    корневой_элемент (модель содержания)>
                              ]
```

Все, что указано в квадратных скобках ([...]) образует внутреннею схему DTD. Внешние схемы DTD обычно содержатся в отдельных файлах с расширением `.DTD`. Полная схема DTD для экземпляра XML состоит из комбинации внутреннего и внешнего подмножества DTD, если оба они существуют. Другими словами, если с экземпляром DTD связана не только внутренняя, но и внешняя схема DTD, проверка экземпляра проводится с помощью обеих схем.

Для того чтобы указать внешнюю схему DTD, объявление типа документа нужно изменить таким образом, что он содержал ключевое слово SYSTEM или PUBLIC, после которого указывается адрес URL файла, содержащего схему DTD. Ключевое слово PUBLIC обычно используется в том случае, если схема DTD общедоступна и совместно используется многими пользователями. Подобные общедоступные схемы DTD обычно сохраняются в специальной библиотеке, или репозитарии. Если вы работаете над определенным проектом XML, вы можете обратиться к репозитарию, чтобы посмотреть, не содержится ли в нем подходящая вам схема DTD. Служебные схемы DTD обычно недоступны в подобных репозитариях. Создавая собственные внешние схемы DTD, вы будете ссылаться на них с помощью ключевого слова SYSTEM в объявлении типа документа, как показано ниже:

```
<!DOCTYPE    корневой_элемент SYSTEM "myrules.DTD">
```

Раньше в настоящей главе вы уже просматривали объявление о начале учебных курсов, создав вчера соответствующий документ. Если вы следовали всем инструкциям, вы сохранили документ файле courses.xml, как было показано в листинге 4.1. Теперь вам необходимо немного изменить экземпляр, включив в него соответствующее объявление типа документа. В листинге 4.25 приведен файл courses.xml, в строку 2 которого добавлено объявление DOCTYPE.

Листинг 4.25. Документ courses.xml с добавленным объявлением DOCTYPE

```
1: <?xml version="1.0"?>
2: <!DOCTYPE announcement SYSTEM "courses.dtd">
3: <announcement>
4:   <distribution>For Immediate Release</distribution>
5:   <to>All Potential Students</to>
6:   <from>Devan Shepherd</from>
7:   <subject>Public Course Offerings in August</subject>
8:
9:     <notice>ACME Training is pleased to announce the following
10: public courses, which are offered on a monthly basis.</notice>
11:
12:     <more_info>For more information, or to register for any of
        these courses,
13: please visit <website>http://ACME-
    Train.com/university</website></more_info>
14:
15:   <courses>
16:     <course id="XML111" instructor="Bob Gonzales">Introduction to
        XML</course>
17:     <course id="XML333" instructor="Devan Shepherd">Advanced
        XML</course>
18:     <course id="XMT222" instructor="Gene Yong">XMetal Core Con-
        figuration</course>
19:   </courses>
20: </announcement>
```

Как только вы внесли необходимые изменения, сохраните файл под именем courses2.xml.

В следующем упражнении вы создадите внешнюю схему DTD, сохранив ее в файле courses.DTD, как указано в строке 2, после чего проверите полученный результат. Вспомните рис. 4.2 , на котором было показано дерево DTD, описывающее правила для этого экземпляра. Используя текстовый редактор, создайте схему DTD, приведенную в листинге 4.26, после чего сохраните ее в виде файла courses.DTD.

Листинг 4.26. Схема DTD для проверки документа `courses2.xml`

```
1:  <!ELEMENT announcement ( distribution, to, from, subject,
                             notice, more_info, courses ) >
2:  <!ELEMENT distribution ( #PCDATA ) >
3:  <!ELEMENT to                    ( #PCDATA ) >
4:  <!ELEMENT from                  ( #PCDATA ) >
5:  <!ELEMENT subject             ( #PCDATA ) >
6:  <!ELEMENT notice              ( #PCDATA ) >
7:  <!ELEMENT more_info     ( #PCDATA | website )* >
8:  <!ELEMENT website             ( #PCDATA ) >
9:  <!ELEMENT courses             ( course+ ) >
10:  <!ELEMENT course                 ( #PCDATA ) >
11:  <!ATTLIST course
12:            id   CDATA  #REQUIRED
13:          instructor  CDATA #REQUIRED >
```



В строке 1 определяется элемент `announcement`, выступающий в роли контейнера для таких дочерних элементов, как `distribution`, `to`, `from`, `subject`, `notice`, `more_info`, и `courses`. Каждый из этих элементов, в свою очередь, объявляется как элемент, содержащий текстовые данные. В строках 12 и 13 объявляются атрибуты `id` и `instructor` элемента `course`.

Используя один из сценариев Microsoft XML Validator, проанализируйте файл `courses2.xml`, чтобы получить результат, приведенный на рис. 4.8. Все узлы должны быть раскрыты.

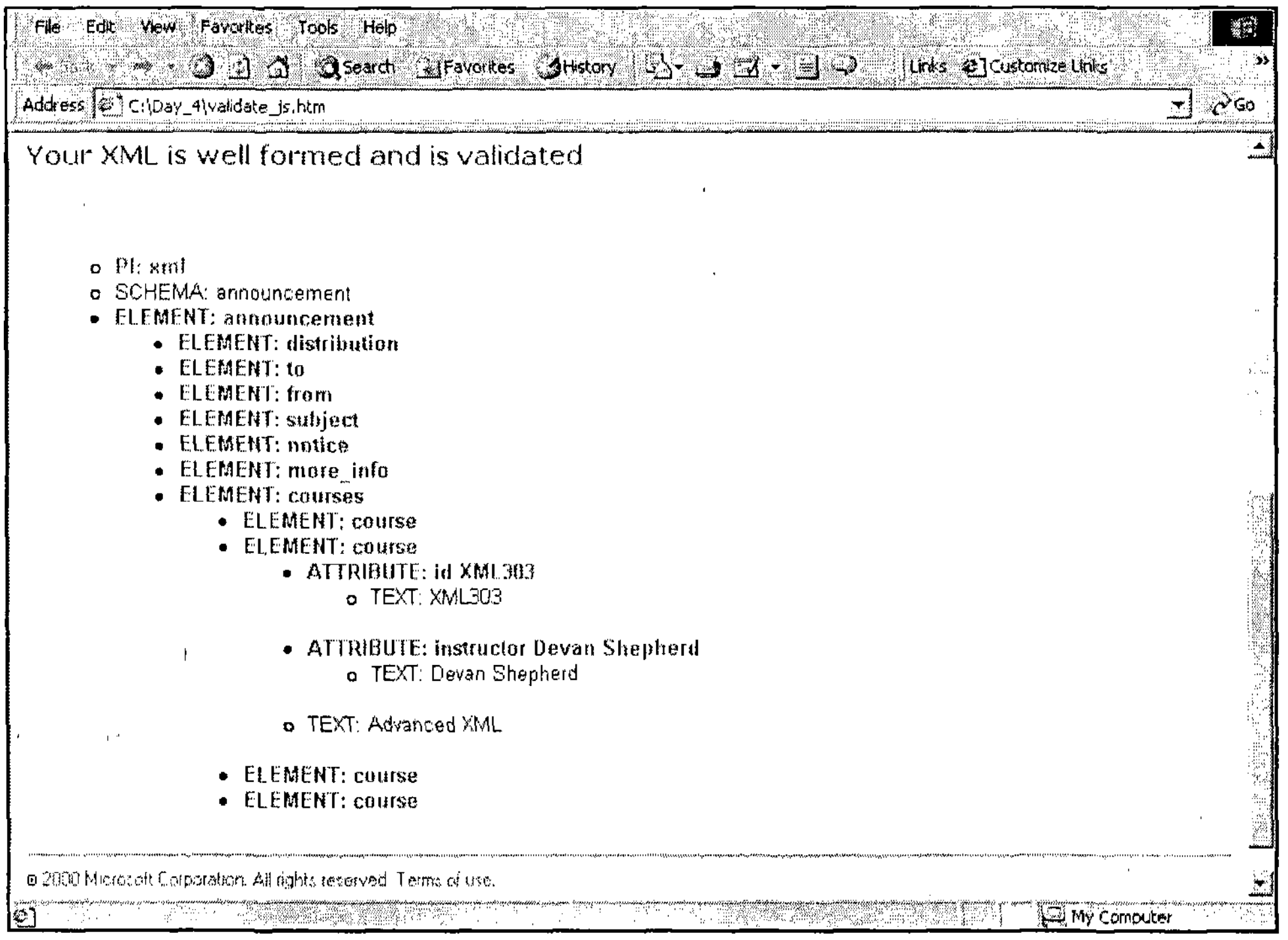


Рис. 4.8. Правильный документ XML (`courses2.xml`) вместе с внешней схемой DTD (`courses.DTD`) успешно прошел проверку с помощью сценария XML Validator

Резюме

Сегодня вы научились создавать внутренние и внешние схемы определения типов документов DTD. Кроме того, вы научились объявлять элементы и их атрибуты, а также увидели ряд примеров достаточно сложных применений ограничений DTD. Вы познакомились с парсерами, которые позволяют проверять документы на соответствие правилам, определенным в схемах DTD.

Для того чтобы завершить упражнения настоящего урока, вам необходимо изучить совершенно новый язык, который имеет некоторое сходство с XML. Он использует похожие конструкции и правила XML, но при этом его синтаксис кардинально отличается от синтаксиса XML.

В уроках 5 и 6 вы познакомитесь с двумя различными языками описания схем, которые позволяют проверять документы XML. Они обладают значительными преимуществами по отношению к DTD. Они используют тот же синтаксис, что и XML, и сами могут быть экземплярами XML. Они также предоставляют возможность использования более изысканных средств контроля, включая ограничения типов данных, а также другие средства, недоступные в DTD.

Вопросы и ответы

Откуда появился подход с использованием схем DTD (Document Type Definition — Определение типа документа)?

Этот подход унаследован от языка SGML, в котором он использовался на протяжении многих лет.

Для чего именно применяются схемы DTD в языке XML?

Схемы DTD (как и любые другие схемы XML) предоставляют шаблон разметки документа, в котором определяется наличие, порядок и расположение элементов и их атрибутов в документе XML.

В чем состоит различие между деревом схемы DTD и деревом документа?

Дерево DTD само по себе не содержит повторяющихся элементов или структур. Однако его структура обеспечивает повторение элементов в соответствующем правильном экземпляре XML.

Что означает объявление типа документа для экземпляра XML?

Объявление типа документа, которое располагается в прологе документа XML, связывает схему DTD с документом XML. Как только обработчик сталкивается с подобным объявлением, он загружает схему DTD и проверяет документ XML на соответствие всем ограничениям, изложенным в этой схеме.

Не путайте термины *объявление типа документа* и *определение типа документа* (DTD). Это совершенно разные вещи.

Какие ключевые слова DTD можно использовать для определения различных типов содержания элементов в XML?

Ключевое слово `ANY` допускает любое содержание элемента — или данные, или другие элементы. Смешанное содержание позволяет элементам содержать комбинацию текстовых данных и элементов. Ключевое слово `EMPTY` указывает на отсутствие в элементе какого-либо содержания. Содержание можно объявить для элементов-контейнеров прямо в документе XML.

Какие два типа парсеров используются в XML?

Все парсеры XML делятся на две группы — способные проверять документы на соответствие ограничениям, изложенным в схеме DTD или подобной схеме другого типа, и не обладающие подобными возможностями. Парсер, который не поддерживает проверку выполнения ограничений, может определить только то, что документ XML хорошо оформлен, т. е. соответствует основным синтаксическим правилам XML. Парсер, который поддерживает такую проверку, способен пойти дальше, не только определив то, что документ XML хорошо оформлен, но и то, что он соответствует всем ограничениям, заданным в связанной с ним схеме.

Упражнение

Проверить свои знания, полученные в результате изучения материала настоящего урока. Ответ вы найдете в Приложении А, Ответы на вопросы".

Вчера, в уроке 3, вы создали диаграмму в виде древовидной структуры и перечислили ряд правил для языка разметки MCML. Используя знания, полученные сегодня, создайте схему DTD для проверки файлов MCML.

День 5-й

Схема XML-Data Reduced

Вы уже знаете, как создавать схемы определений типов документов DTD для проверки выполнения ограничений в хорошо оформленных документах XML. В настоящем уроке вы познакомитесь с еще одним типом схем — XML-Data Reduced (XDR).

Сегодня рассматриваются следующие темы.

- Что такое схема XDR и каким образом она связана с документом XML.
- Различие между схемами DTD и другими схемами, такими как XDR.
- Написание и интерпретация схем XDR.
- Объявление элементов и атрибутов в схеме XDR.
- Использование типов данных в схемах XDR.
- Несколько примеров использования схем XDR и почему они важны для программистов на XML.

Зачем нужны другие схемы

В уроке дня 4-го "Определение типа документа" вы научились создавать схемы DTD для проверки ограничений в документах XML. На самом деле, вы научились создавать документы, в которых описываются структура, семантика и содержание ваших экземпляров XML для того, чтобы последние еще и соответствовали необходимым ограничениям. Добавление ограничивающих правил с помощью схемы DTD или схемы другого типа позволяет обеспечить не только хорошее оформление экземпляра XML, но и его соответствие ряду дополнительных правил.

Затем вы научились не только ограничивать содержание атрибутов, но и задавать их стандартные значения, используемые в том случае, если значения атрибутов не указаны явно в экземпляре XML. Вы научились создавать подстановочные элементы, на которые можно ссылаться в документах для включения определенного содержания. Вы также узнали о возможности использования обработчиков XML для соответствия документов определенным правилам.

Процесс проверки документов на соответствие ограничивающим правилам позволяет связать семантику документа с конкретной задачей, играющей немалую роль в обмене информацией в электронном бизнесе или других разновидностях электронных коммуникаций. В конце концов, если все необходимые правила и структуры данных используются в экземпляре XML должным образом, то различные обработчики смогут при предоставле-

нии еще и схемы DTD подтвердить, что проанализированный экземпляр XML является правильным; тогда исключаются недоразумения, например, связанные с несоответствием типов данных, при проведении транзакций. Предположим, есть два приложения, используемые деловыми партнерами для обмена данными электронной коммерции через Internet. Для того чтобы осуществляемые при этом транзакции считались надежными, правила бизнес-модели должны строго соблюдаться. Реализуя проверку данных на уровне документа благодаря использованию схемы, приложения, генерирующие и принимающие транзакции, можно оптимизировать для обеспечения максимального быстродействия; соответствие полей и правильность записей проверяются на уровне экземпляров XML.

Разработчики могут определить и согласовать структуру и словарь данных, обмен которыми осуществляется через Internet, и использовать определения типов документов и другие схемы для соблюдения ограничивающих правил. Точно также разработчики могут использовать схемы в качестве основы при необходимости преобразования данных между различными автоматизированными системами, обеспечивая надежные функции связывания при переходе от одной схемы словаря данных к другим. Таким образом, схемы образуют основу для крупномасштабных обменов данных. Приложения, которые принимают или генерируют данные обмена, должны создаваться, однако по крайней мере можно выбрать такую структуру данных, тогда становится возможной независимая разработка подобных приложений и вспомогательных средств. Это позволяет защитить капиталовложения в существующие модели обработки данных, а также сохранить хранилища накопленных данных, которые в противном случае претерпели бы некоторую переработку; вместо этого можно получать или генерировать данные в соответствии с заданными структурными ограничениями.

В уроке 4 вы познакомились с такой схемой, как DTD. Подходы, с которыми вы познакомитесь сегодня, также касаются использования схем. Каждый из подобных подходов обладает как преимуществами, так и недостатками. Схемы DTD неплохо подходят для тех ситуаций, для которых они разрабатывались, однако они обладают целым рядом недостатков по сравнению с другими схемами. Например, вчера, в уроке 4, вы узнали, как использовать схемы DTD для проверки структуры документа на соответствие стандартам. Однако при этом вам необходимо дополнительно изучать целый ряд языков программирования, которые используют синтаксис, отличный от XML. В этом состоит серьезный недостаток данного подхода с точки зрения новичков. Таким образом, одно из преимуществ схем, с которыми вы познакомитесь в настоящем и следующем уроках, как раз и состоит в том, что они базируются на известных синтаксических и структурных правилах, характерных для уже знакомого вам XML. Другими словами, эти схемы выражаются с помощью синтаксиса XML.

ПРИМЕЧАНИЕ

Для описания XML и DTD может использоваться грамматика EBNF (Extended Backus-Naur Form), однако в каждом случае используется свой "диалект". Таким образом, схемы DTD используют грамматику EBNF, а другие схемы — синтаксис XML (в том числе элементы и атрибуты, открывающие и закрывающие дескрипторы и т.д.), что значительно упрощает их использование.

Грамматика EBNF разработана в конце 1960-х годов для выражения синтаксически правил компьютерных языков и других точных языковых структур. Николас Вирт (Nicklaus Wirth), создатель языков Pascal и Modula 2, приложил немало усилий для популяризации грамматики EBNF. Приведем в качестве примера следующую инструкцию на EBNF, которая отражает синтаксис XML:

```
[1] document ::= prolog element Misc*
```

Эту простую инструкцию можно интерпретировать следующим образом: "Документ XML должен содержать пролог, за которым должен следовать элемент, за которым следует 0 или другие элементы".

Мы обсудили один из самых серьезных недостатков схем DTD, хотя существует и целый ряд других. В частности, к связанным со схемами DTD проблемам относятся следующие.

- Как вы уже видели, схемы DTD не являются экземплярами XML; работа с ними требует от разработчика изучения совершенно другого языка программирования, использующего достаточно сложный синтаксис.
- Кроме того, создавать, поддерживать и проверять на соответствие стандартам с помощью редакторов и утилит XML схемы DTD достаточно сложно; однако следует отметить, что для создания схем DTD существуют специальные инструменты.
- Более того, схемы DTD не предоставляют контроль за типами данных, за исключением самых простых текстовых данных и типов документов, что затрудняет проверку на соответствие стандартам в приложениях, использующих другие типы данных, такие как финансовые транзакции, транзакции данных, обмен научными данными и т.д.
- Схемы DTD не являются экземплярами XML, поэтому их нельзя легко расширить или преобразованы к другим языкам разметки —HTML или DHTML — как другие документы XML.
- Схемы DTD не обеспечивают поддержки пространств имен XML, которые позволяют вам смешивать в документе элементы разных структур документов. (Подробные сведения о пространствах имен вы найдете в уроке 8.)
- В связи с использованием автоматической обработки документов XML требуется более точное объявление ограничений, чем это возможно с использованием только схемы DTD.

По всем этим причинам просто нужна какая-нибудь альтернатива схеме DTD, которую можно описать с помощью синтаксиса XML. Стандартизация подобных решений — одна из задач рабочей группы W3C XML Schema Working Group. 2 мая 2001 года консорциум *W3C* сформулировал последний стандарт, известный как XSD (XML Schema Definition Language — язык определения схем XML), с которым вы подробно познакомитесь в уроке 6. В частности, эта рабочая группа должна была согласовать предложения нескольких групп и компаний — поставщиков коммерческих решений. Предмет нашего рассмотрения, схема XDR, базируется на одном из подобных предложений консорциума *W3C*: XML-Data. Стоит отметить, что XML-Data — это полное имя языка описания схем, предложенного компанией *Microsoft*, а XML-DataReduced (XDR) — это "часть" полной рекомендации. Несмотря на то, что консорциум *W3C* формализировал стандарт XSD, стандарт XDR все еще используется многими приложениями. Многие программные продукты, причем не только от компании *Microsoft*, предлагают поддержку стандарта XDR. Вам стоит познакомиться с синтаксисом DTD, XDR и XSD. Такие средства, как XML-Spy (`www.xmlspy.com`), позволяют преобразовать схему, написанную на одном из этих "диалектов", и преобразовать к другим, используя при этом транслитерацию. Подобное преобразование позволяет сэкономить массу времени; однако вам обязательно придется просмотреть результаты подобного преобразования, чтобы убедиться в том, что типы данных и другие нюансы преобразованы должным образом.

Рабочая группа W3C XML Schema Working Group рассмотрела следующие предложения.

- **Схемы DTD**. Эти схемы появились применительно к SGML, и только потом их перенесли в мир XML. Спецификация XML включает подробные сведения о схемах DTD.
- **XML-Data/XML-Data Reduced**. Компания *Microsoft* и ее партеры направили свое предложение консорциуму *W3C*, касающееся схемы XML-Data, в январе 1998 года, еще до того, как был утвержден стандарт XML 1.0. Одно из преимуществ этого подхода заключается в том, что он поддерживает большое количество различных типов данных. Полной поддержкой схем XML-Data могут похвастаться синтаксические анализаторы XML компании *XML*.

- **Schema for Object-Oriented XML (SOX)**. Схема SOX была разработана компанией *VEO Systems*, которую в дальнейшем поглотила компания *CommerceOne*. Схема SOX (отправленная в консорциум *W3C* в сентябре 1998 года) прошла через несколько версий, и на данный момент существует уже в версии 2.0. Одно из преимуществ SOX заключается в простоте использования при реализации структуры экземпляров XML.
- **DOD (Document Content Description — Описание содержания документа).** Компании *Microsoft* и *IBM* объединили свои силы для разработки стандарта DOD (отправленного консорциуму *W3C* в августе 1998 года), который объединяет концепции XML-Data и синтаксис, известный как RDF (Resource Description Framework — Структура описания ресурсов).
- **DDML (Document Description Markup Language — язык разметки описания документов).** Раньше известный как Xschema, этот подход разработан членами группы рассылки XML-dev, которые работали над обеспечении функциональных возможностей DTD с использованием синтаксиса XML. Работа над стандартом DDML была остановлена в связи с созданием рабочей группы W3C XML Schema Working Group. Несмотря ни на что, стандарт DDML был отправлен на рассмотрение консорциумом *W3C* в январе 1999 года.

Поскольку консорциум *W3C* создал нечто, что может считаться наилучшим решением, базирующимся на полученных предложениях, очень важно понимать, откуда появился стандарт XSD. В приведенном выше списке перечислены только основные подходы и игроки, которые объединили свои усилия для выработки наилучшего варианта схемы. В качестве такого решения можно рассматривать и XSD. Однако некоторые компании предлагают использовать другие языки описания схем. Например, язык описания схем XML-Data Reduced связан с BizTalk, ведушей технологией, используемой в электронной коммерции, электронном бизнесе, электронных покупках, управлении звеньями поставок и т.д.

У вас может возникнуть вопрос о целесообразности изучения нескольких языков описания схем, если вы можете выбрать и использовать только один из них. Если язык описания схем XSD считается стандартом электронной коммерции, за которым стоят годы разработки, почему не ограничиться изучением и использованием только его? В настоящее время положение несколько иное. Поддержка XSD достаточно ограничена. Следовательно, многие наследуемые системы создавались с использованием схем DTD и XDR или вообще других подходов. Компания *Microsoft* прилагает немало усилий для поддержки как ее собственного языка описания схем XDR, так и принятого в качестве стандарта языка XSD, но факт остается фактом: пионеры электронной коммерции, бизнес-пользователи, правительственные организации (здесь речь идет преимущественно о США. — *Прим. ред.*), а также поставщики готовых решений уже вложили миллиарды долларов в разработку решений, базирующихся на схемах DTD и XDR и других стандартах. Должно пройти определенное время, прежде чем мир сможет перейти к использованию одного стандартизированного подхода к формулированию ограничений для документов XML. По этой причине я предлагаю вам получить хотя бы общие сведения о нескольких, наиболее распространенных на сегодня схемах.

Рабочая группа W3C XML Schema Working Group разработала требования, к языку описания схем XSD в начале 1999 года, а спецификация XSD на данный момент относится к категории рекомендаций. Вы поймете различия между языками описания схем XSD и XDR после того, как изучите сегодняшний и завтрашний уроки, а также выполните соответствующие упражнения.



Если вы потратите много времени на изучение одного из доступных языков описания схем, вы сможете воспользоваться специальными инструментами, упрощающими преобразование полученных результатов к другим схемам. К подобным коммерческим программным продуктам относятся XML Authority и XML Spy. Кроме того, компании-разработчики также предлагают различные утилиты преобразования. Компания *Microsoft* предлагает бесплатный инструмент для преобразования схем XDR в схемы XSD. Однако имейте в виду, что преобразования схем с одного диалекта на другой часто приводят к возникновению ошибок. Как и в случае любого другого процесса автоматической генерации кода, вам понадобится потратить немало усилий на переработку полученных результатов преобразования, прежде чем будут получены необходимые результаты.

Подробно о XDR

Стандарт XML-Data Reduced, или просто XDR, — это только часть полной спецификации XML-Data, направленной компанией *Microsoft* в консорциум *W3C*. При изучении материала предыдущего урока вы уже загрузили и использовали специальный сценарий для проверки соответствия стандартам, который обращается к синтаксическому анализатору Microsoft XML Parser (MSXML). Реализация схемы в этом анализаторе базируется на спецификации языка XML-Data Reduced, хотя анализатор позволяет проверять схемы DTD и XSD. Несмотря на то, что анализатор MSXML достаточно полон, он обладает некоторыми ограничениями, такими как отсутствие поддержки наследования и других средств объектно-ориентированного подхода. Все эти нюансы подробно рассматриваются в следующем уроке. Более того, несмотря на поддержку широкого диапазона языков описания схем, реализованную в синтаксических анализаторах от таких компаний, как *Sun*, *Apache*, *IBM* и многих других, вам обязательно следует убедиться в отсутствии проблем, связанных с выбором определенной аппаратной платформы или операционной системы. К счастью, как же отмечалось раньше, существуют синтаксические анализаторы и другие средства проверки для практически любой аппаратной платформы или операционной системы.



Для использования XDR для проверки документа XML необходим соответствующий синтаксический анализатор. На данный момент самым удачным вариантом изо всех доступных считается синтаксический анализатор MSXML. На момент написания настоящей книги доступна последняя версия 4, которая обладает поддержкой XSD и XDR.

Как и схемы DTD, схемы XDR определяют элементы, представленные в экземпляре XML, а также любые атрибуты, которые связаны с указанными элементами. Для того чтобы провести проверку на соответствие стандартам, каждый элемент в экземпляре должен быть объявлен в соответствующей схеме. Отсутствие элемента в схеме определяет его отсутствие в правильном экземпляре XML.

Взаимосвязи элементов также определяются в схеме XDR; таким образом, все дочерние элементы, содержащиеся в определенных элементах экземпляра, объявляются и в дальнейшем определяются в соответствии с их собственным содержанием. Сюда относятся последовательность, в которой дочерние элементы приводятся в документе, а также количество дочерних элементов.

Схема определяет и то, пустой ли элемент или он содержит текстовые данные, другие элементы или определенную комбинацию последних. Схема также может определять стандартные значения атрибутов.

Объявления элементов и атрибутов XDR

Вы начнете изучение XDR с рассмотрения того, каким образом подобные схемы налагают ограничения на элементы и атрибуты в вашем документе XML. Все ограничения задаются с использованием элементом и атрибутов, принадлежащих диалекту XDR. Это может привести к недоразумениям. Элемент или атрибут в одном языке (XDR) используется для определения правил и добавления структуры к другому языку — вашему языку разметки.

Схемы XDR — это экземпляры XML, а значит, они состоят из элементов и атрибутов XML. Язык XDR содержит определенные элементы и атрибуты XML в соответствии со схемой, которая проверяет на соответствие стандартам все экземпляры схем XDR. Другими словами, так как схемы XDR соответствуют всем синтаксическим правилам и стандартам XML, они должны соответствовать набору правил, установленных для данного языка описания схем. Как бы там ни было, схемы XDR расширяемы, о чем вы узнаете дальше в настоящей главе.

Поскольку схемы XDR проверяются с помощью других схем, к их структурам применяются определенные правила и ограничения. Например, корневой элемент в схеме XDR всегда называется `Schema` и всегда содержит атрибут `xmlns`, определяющий пространство имен как для него, так и для содержащихся в нем дочерних элементов.



Вы уже неоднократно сталкивались с понятием пространства имен, поэтому интересуетесь, что же это такое. Подробные сведения о пространствах имен вы найдете в уроке дня 8-го, однако в настоящей книге они будут еще неоднократно упоминаться до этой главы. Если говорить кратко, пространство имен — это идентификация элемента или атрибута как принадлежащего к определенному заданному множеству. Элементы и атрибуты в определенном языке описания схем привязываются к набору, определенному именно для этого языка.

Это означает, что вы создаете документ XML, который соответствует схеме, которая, в свою очередь, должна соответствовать еще одной схеме. К счастью, вам придется иметь дело только с двумя из этих трех документов: вашим документом XML и схемой, используемой для его проверки. В настоящей главе вы познакомитесь с синтаксисом и использованием языка описания схем XML-Data Reduced в рамках знакомства с ним. Вы получите возможность создать несколько простых схем XDR и проверить их с помощью анализатора MSXML. Основная цель, которую вы преследуете при изучении материала настоящего урока, — возможность создания простых схем, понимание синтаксиса основных элементов и атрибутов, а также знание того, где можно найти дополнительные сведения, выходящие за рамки рассмотрения настоящей главы.

Объявления ElementType

Изучите простой документ XML, приведенный в листинге 5.1. Вспомните, что этот документ уже приводился в уроке 4, постепенно претерпевая изменения по мере изучения вами различных нюансов схем DTD. Сегодня вы проделаете подобный путь еще раз, используя различные схемы XDR для проверки документа по мере его развития. Создайте документ XML, приведенный в листинге 5.1, и сохраните его под именем `message01.xml`.

Листинг 5.1. Простой документ XML со ссылкой на схему XDR `message01.xml`

```
1: ?xml version="1.0"?>
2: <note xmlns = "x-schema:message01.xdr">Remember to buy milk on the
   way home from work</note>
```

ПРИМЕЧАНИЕ

Если вы создали документ XML, используя редактор, который позволяет проводить проверку, ваш документ не будет правильным, поскольку вы еще не создали файл со схемой XDR; вы сделаете это немного позже. Если вы используете простенький текстовый редактор, подобных проблем у вас не возникнет.

Корневой элемент в файле `message01.xml` содержит атрибут `xmlns` со значением `"x-schema:message01.xdr"`. Это указывает проверяющему обработчику XML на то, что схема, связанная со стандартным пространством имен для этого документа, содержится в файле `message01.xdr`. На момент написания настоящей книги единственным анализатором, поддерживающим работу со схемами XDR, был MSXML.

Вы создадите схему XDR для этого элемента, содержащую корневой элемент `Schema`, и элемент `ElementType`, определяющий элемент `note` в экземпляре XML.

Корневой элемент `Schema` в схеме XDR содержит элементы и атрибуты, соответствующие всем ограничивающим правилам для экземпляра документа XML. Все определения в схеме XML содержатся в рамках элемента `Schema` верхнего уровня. Определение элемента `Schema` должно базироваться на пространстве имен `xmlns = "urn:schemas-microsoft-com:xml-data"`. Объявление пространства имен с точки зрения XML является инструкцией обработки.

Элемент `Schema` в документе схемы XML также должен содержать объявления пространств имен для других схем, таких как пространство имен, определяющее встроенные типы данных для схемы XML. Типы данных используются для ограничения строковых данных таким образом, чтобы они принимали только такие значения, как даты, числовые значения, числа с плавающей запятой и т.д. Пространство имен для схемы XDR: `xmlns = "urn:schemas-microsoft-com:xml-data"`.

Таким образом, корневой элемент `Schema` схемы XML практически всегда выглядит следующим образом:

```
<Schema
    name="имя_схема  xmlns="urn:schemas-microsoft-com:xml-data"
    xmlns:dt="urn:schemas-microsoft-com:datatypes">
<--Объявления других элементов-->
</Schema>
```

При объявлении основных компонентов структуры документов XML используются такие типы элементов, как `ElementType`, `AttributeType`, `Entity` и `Notation`. Другими словами, например, элемент `ElementType` — это элемент XDR, используемый для определения ограничений, которые накладываются на элементы в соответствующем экземпляре XML.

Элемент `ElementType` в схемах XDR имеет следующий синтаксис:

```
<ElementType
    content="{empty|textOnly|eltOnly|mixed}"
    dt:type"datatype"
    model="{open|closed}"
    name="idref"
    order="{one|seq|many}"
```

Каждый из этих атрибутов имеет значение, зависящее от описываемого элемента. Например, атрибут `content` свидетельствует о том, что содержание объявляемого элемента должно быть пустым, быть только текстовыми данными, другими элементами или соответствовать модели смешанного содержания. Атрибут `dt:type` объявляет тип данных для элемента. Допустимые типы данных — `date`, `number`, `time` и т.д. — будут рассматриваться подробно дальше в настоящей главе. Атрибут `model` указывает на то, может ли содержание быть только определяемого моделью типа, или оно может выходить за ее рамки. Если модель содержания определена как открытая, элемент

может включать в себя дополнительные элементы и атрибуты, явно не указанные в этой модели. Если модель содержания определена как закрытая, элемент не может включать в себя дополнительные элементы и атрибуты, а также смешанное содержание, явно не указанные в его модели содержания. Например, в схемах DTD используется закрытая модель содержания. Атрибут `name` используется для идентификации самого определяемого элемента. Обратите внимание: если тип элемента объявлен как дочерний, его имя указано в элементе `element`. Скоро вы увидите несколько примеров. Атрибут `order` определяет порядок следования дочерних элементов. Значение `one` допускает включение только одного элемента из набора, значение `seq` требует указания элементов в строго определенном порядке, а значение `many` допускает указание элементов в любом порядке, вплоть до отсутствия некоторых из них. Сведения об этих атрибутах подытожены в табл. 5.1.



В схеме XDR, если модель содержания объявлена открытой, определяемый элемент может содержать дополнительные элементы или атрибуты, которые не были явно определены в модели. Если модель была объявлена как закрытая, элемент не может содержать элементы или смешанные данные, которые не были явно указаны в модели содержания.

В листинге 5.2 приведена схема `message01.xdr`. Используя свой текстовый редактор, введите этот документ и сохраните его под именем `message01.xdr`. После этого вы сможете изучить пошаговый анализ схемы.

Листинг 5.2. Схема XDR для проверки документа `message01.xml` (сохранена в файле `message01.xdr`)

```
1: <?xml version="1.0"?>
2: <Schema
3:  name="message01"
4:  xmlns="urn:schemas-microsoft-com:xml-data"
5:  xmlns:dt="urn:schemas-microsoft-com:datatypes">
6:  <ElementType name="note" model="closed" content="textOnly"
    dt:type="string"/>
7: </Schema>
```



Давайте подробно изучим документ из листинга 5.2. Поскольку схема XDR является экземпляром XML, в строке 1 содержится соответствующее определение. Открывающий дескриптор корневого элемента Schema начинается в строке 2. В данном случае открывающий дескриптор содержит три атрибута. Атрибут `name` в строке 3 содержит имя документа, которое вы присвоили этой схеме. Это необязательный элемент; однако его включение считается хорошим тоном в программировании.

В строке 4 определяется стандартное пространство имен для схемы XDR (`xmlns="urn:schemas-microsoft-com:xml-data"`), это указывает синтаксическому анализатору MSXML, что используется язык описания схем XML-Data Reduced. Это стандартное пространство имен URI, используемое в схемах XDR. Стандартное пространство имен назначается для всех элементов, содержащихся в корневом элементе `Schema` до тех пор, пока они не будут объявлены в префиксах для других пространств имен. Пространство имен XML под названием URI (Unified Resource Identifier — Унифицированный идентификатор ресурсов) используется в качестве уникальной метки идентифицируемого пространства имен. Пространство имен URI — это не что иное, как последовательности символов, используемые для различения имен. Как вы уже знаете, URL — это аббревиатура от Uniform Resource Locator (унифицированный локатор информационного ресурса). Адрес URL — это глобальный адрес удаленного

ресурса или документа. Объявления пространств имен могут содержать URI, напоминающие адреса URL из-за схожести с адресами Web-страниц; однако это означает только уникальность пространства имен. Более подробные сведения об этом вы найдете в уроке 8. URI могут также содержать и универсальные имена ресурсов URN (Universal Resource Name). Имена URN также должны быть уникальными; однако они необязательно должны совпадать с адресами URL.



URI, используемый в пространстве имен XML, — это не что иное, как метка, или строка символов, которая уникальным образом идентифицирует пространство имен. При этом не играет совершенно никакой роли, на что именно указывает URI.

В строке 5, последней строке открывающего дескриптора корневого элемента Schema, объявляется пространство имен для типов данных, объявленных в схеме. Это указывает анализатору на необходимость подтверждения любого элемента в оставшейся части схемы XDR, который содержит атрибут `dt:type`, соответствующий правилам типов данных, определенных компанией *Microsoft*. Подробные сведения о типах данных вы найдете дальше в настоящей главе.

В строке 6 содержится объявление единственного элемента в документе `message01.xml` — элемента `note`:

```
<ElementType name="note" model="closed" content="textOnly"
dt:type="string"/>
```

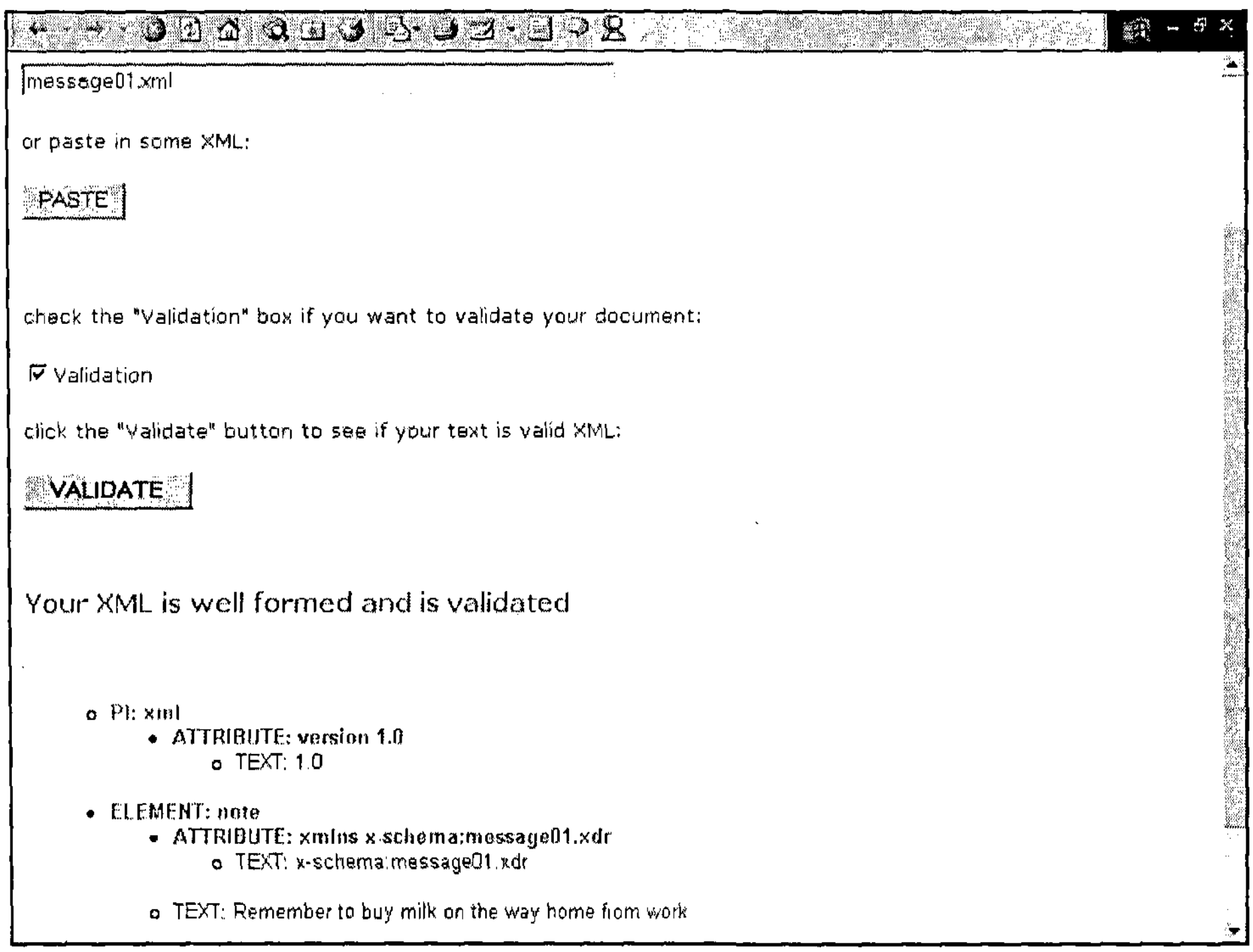


Рис. 5.1. Результаты проверки файла `message01.xml` с помощью анализатора MSXML

В строке 6 также указаны атрибуты, которые более тонко описывают элемент `note`. Атрибут `name` определяет имя элемента в экземпляре XML. В данном случае атрибут `name` определяет имя элемента `note`. Значение атрибута `model` равно `closed`, что ука-

зывает на то, что документ XML может содержать элементы, которые объявляются в данной схеме. Другими словами, если документ XML будет содержать элементы, не указанные в схеме, он считается неправильным. Атрибут `content` указывает на то, что элемент может содержать только текстовые данные, поскольку значение этого атрибута равно `textOnly`. С другими значениями этого атрибута вы познакомитесь дальше в настоящей главе. Атрибут `type` относится к пространству имен типов данных, на что указывает префикс `dt:`. В данном случае допустимым типом данных для элемента `note` является `string`. Это соответствует типу данных `CDATA` в схеме DTD.

Для проверки экземпляра XML, сохраненного в виде файла `message01.xml`, на соответствие синтаксическим правилам и стандартам используйте сценарий, который вы загрузили во время изучения материала предыдущей главы. Результаты проверки приведены на рис. 5.1.

Элемент `ElementType` допускает объявление нескольких предопределенных атрибутов. Сведения об этих атрибутах подытожены в табл. 5.1.

Таблица 5.1. Стандартные атрибуты элемента `ElementType`

Имя атрибута	Описание
`name`	Это обязательный атрибут, определяющий имя элемента
`content`	Этот атрибут описывает содержание элемента. Допустимыми значениями этого атрибута являются `empty`, соответствующее пустому элементу; `eltOnly`, означающее, что элемент может быть только контейнером для других элементов; `textOnly`, означающее, что элемент может содержать только текстовые данные; `mixed`, которое позволяет элементу содержать смешанные данные. Стандартным значением атрибута `content` является `mixed`
`dt:type`	Этот атрибут определяет тип данных элемента. Префикс `dt:` позволяет указать пространство имен для URI типа данных
`model`	Этот атрибут может принимать такие значения, как `open` или `closed`, разрешая или запрещая использовать элементы, не определенные в схеме XDR соответственно
`order`	Этот атрибут объявляет порядок следования дочерних элементов экземпляре XML. К допустимым значениям атрибута `order` относятся `one`, предполагающее наличие всего одного элемента; `many`, предполагающее наличие любого количества элементов, следующих в любом порядке; а также `seq`, предполагающее указание элементов в строго определенном порядке.

Вложенные объявления `ElementType`

Как и в случае схем DTD, вложенные элементы, указанные в схеме XDR, должны быть объявлены и определены. В листинге 5.3 документ `message01.xml` был изменен таким образом, что в нем содержатся один дочерний элемент, `message`, и один корневой элемент `note`. Используя текстовый редактор, создайте документ (или измените уже существующий), соответствующий листингу 5.3.

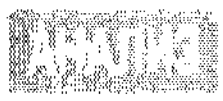
Листинг 5.3. Добавление вложенного элемента `message02.xml`

```
1: <?xml version="1.0"?>
2: <note xmlns="x-schema:message02.xdr">
3:  <message>Remember to buy milk on the way home from work</message>
4: </note>
```

В листинге 5.4 приведена схема XDR для проверки документа — `message02.xml`. Создайте эту схему и сохраните как файл — `message02.xdr`.

Листинг 5.4. Схема XDR для документа `message02.xml` (сохранена в файле `message02.xdr`)

```
1:  <?xml version="1.0"?>
2:  <Schema
3:   name="Untitled-schema"
4:   xmlns="urn:schemas-microsoft-com:xml-data"
5:   xmlns:dt="urn:schemas-microsoft-com:datatypes">
6:   <ElementType name="note" model="closed" content="eltOnly">
7:       <element type="message"/>
8:   </ElementType>
9:   <ElementType name="message" model="closed" content="textOnly"
     dt:type="string"/>
10:  </Schema>
```



Строки 1—5 практически совпадают с аналогичными строками из схемы `message01.xdr`. Однако в строке 3 указано имя схемы `"Untitled-schema"`. Указывать это имя необязательно.

Строки с 6 по 8:

```
<ElementType name="note" model="closed" content="eltOnly">
        <element type="message"/>
    </ElementType>
```

указывают на то, что элемент `note` содержит только другие элементы (`content="eltOnly"`). Элемент `message` объявляется как дочерний по отношению к элементу `note`.

В строке 9 элемент `message` определяется как содержащий только текстовые данные. Значение `textOnly` указывает на содержимое элемента, а не его тип данных. Это приводит к некоторым недоразумениям, поскольку термин *textOnly* может выступать в качестве определения типа данных, который позволяет содержать только текст. На самом деле он означает, что элемент содержит данные (любого типа), а не другие элементы.

Обратите внимание на то, что элемент `message` указан в схеме дважды, один раз в строке 7, где он объявляется как дочерний по отношению к элементу `note`, а второй раз — в строке 9, где он определяется полностью. Когда вы работали со схемами DTD в предыдущем уроке, вы поступали подобным образом: сначала объявляли дочерние элементы, а затем давали их полное определение.

Элемент `element` — это дочерний элемент по отношению к элементу `ElementType` в схеме XDR. В элемент `ElementType` могут быть включены дочерние элементы, перечисленные в табл. 5.2.

Таблица 5.2. Дочерние элементы для элемента `ElementType`

Имя элемента	*Описание*
element	Объявляет дочерний элемент
description	Обеспечивает описание элемента `ElementType`
datatype	Определяет тип данных элемента `ElementType`
group	Определяет порядок следования элементов
AttributeType	Определяет атрибут
attribute	Определяет сведения о дочернем элементе `AttributeType`

Вложенные элементы

Вы обязательно должны помнить, что пустые элементы в экземпляре XML — это такие элементы, которые не содержат данные или другие элементы. Пустые элементы все равно могут содержать атрибуты, но между их открывающим и закрывающим дескрипторами не содержится никаких данных. Это не означает, что пустые элементы совершенно бесполезны. Атрибуты пустых элементов могут содержать достаточно много полезных сведений; другие пустые элементы могут выступать в роли подстановочных. В любом случае, они имеют информационное значение даже без содержания. Из-за того, что между открывающим и закрывающим дескрипторами не содержится никаких данных, XML допускает сокращенную форму представления пустого элемента, приведенную ниже:

```
<имя_элемента/>
```

Закрывающий дескриптор отсутствует, а завершающий символ в виде косой приведен прямо в открывающем дескрипторе.

Используя в качестве основы файл `message02.xml`, вы добавите в него пустой элемент `number`. После этого вы увидите, каким образом можно задать ограничение для этого элемента в схеме XDR.

В листинге 5.5 приведен экземпляр XML, в строке 3 которого содержится пустой элемент `number`.

Листинг 5.5. Документ XML с пустым элементом `message03.xml`

```
1: <?xml version = "1.0"?>
2: <note xmlns="x-schema:message03.xdr">
3:     <number />
4:     <message>Remember to buy milk on the way home from work</message>
5: </note>
```

Вы можете видеть, что этот экземпляр достаточно прост. Хотя может показаться, что добавление пустого элемента не имеет никакого значения, вы сможете изменить экземпляр в дальнейшем таким образом, что элемент `number` превратится в атрибут элемента `message`. Когда вы создаете документы и схемы XML, очень часто оказывается сложно принять решение о том, должен ли определенный объект данных представлять собой элемент или атрибут. Вы уже рассматривали этот вопрос в уроке 2, в которой создавали синтаксически правильные экземпляры. Никакие правила не требуют от вас, чтобы определенный объект данных мог быть только элементом или только атрибутом. Решение зависит только от автора документа. При решении этого вопроса вам следует учесть несколько моментов. Например, элементы обычно содержат информацию, ценную как данные, в то время как атрибуты содержат дополнительную информацию — метаданные, которая используется для модификации элементов точно так же, как прилагательные используются для дополнения существительных. Элементы могут содержать дочерние элементы, а атрибуты — нет.

Теперь вы определите пустой элемент `number`, чтобы показать соответствующий синтаксис в схеме XDR. Затем вы добавите необходимые атрибуты.

Для того чтобы указать, что в схеме XDR элемент соответствует модели содержания `empty`, вам необходимо объявить его, используя атрибут `content="empty"` элемента `ElementType`. Вы помните, что `empty` — это одно из допустимых значений атрибута `content`. Синтаксис нового объявления и определения включает элемент `element`, который должен быть добавлен к элементу `note` для указания того, что элемент `number` содержится в корневом элементе. Синтаксис выглядит следующим образом:

```
<element type="number"/>
```

После этого новый элемент ElementType должен быть добавлен для объявления нового пустого элемента:

```
<ElementType name="number" model="closed" content="empty">
```

Полная схема XDR, приведенная в листинге 5.6, теперь подтверждает правильность файла message03.xml. Создайте эту схему и сохраните ее под именем message03.xdr в той же папке, в которой содержится и документ message03.xml. После этого вы сможете проверить документ XML на соответствие стандартам точно так же, как и все остальные примеры, которые вы создадите при изучении настоящей главы.

Листинг 5.6. Схема XDR для документа XML с пустым элементом message03.xml (сохранена в файле message03.xdr)

```
1:  <?xml version="1.0"?>
2:  <Schema
3:   name="Untitled-schema"
4:   xmlns="urn:schemas-microsoft-com:xml-data"
5:   xmlns:dt="urn:schemas-microsoft-com:datatypes">
6:   <ElementType name="note" model="closed" content="eltOnly" order="seq">
7:       <element type="number"/>
8:       <element type="message"/>
9:   </ElementType>
10:  <ElementType name="number" model="closed" content="empty"/>
11:  <ElementType name="message" model="closed" content="textOnly"
     dt:type="string"/>
12:  </Schema>
```

АНАЛИЗ

В строках 6–9 объявляется элемент note, который содержит только элементы, в частности элемент number, который следует после элемента message. Значение атрибута order элемента note равно seq, что указывает на четко фиксированный порядок следования дочерних элементов. Как уже отмечалось раньше, атрибут order может иметь только следующие значения: one, seq или many.

Проверка атрибутов в экземпляре XML с помощью схемы XDR

В предыдущем примере вы объявили дочерний элемент number элемента message как пустой. Это предоставило вам восхитительную возможность поэкспериментировать с синтаксисом представления пустого элемента в схеме XDR, однако в контексте работы с сообщениями целесообразность подобных действий оказывается под вопросом. В следующем примере вы удалите пустой элемент number, заменив его атрибутом number элемента message. В этом случае вы используете хорошие приемы программирования, превращая элемент, который действительно не содержит никаких данных, в атрибут, который можно использовать при определении элемента. Это неплохой пример аналога атрибута с прилагательным, которое дополняет существительное. При создании экземпляров XML вы часто будете использовать этот метод проб и ошибок даже в том случае, если хорошо спланировали свои действия перед созданием кода. Никогда не поздно передумать и объявить как элемент тот объект, который только что вы планировали объявить как атрибут.

В листинге 5.7 приведен экземпляр XML, в котором к элементу `message` добавлен новый атрибут. Внесите необходимые изменения в созданный вами документ и сохраните его под именем `message04.xml`.

Листинг 5.7. Добавление атрибута к документу `message04.xml`

```
1: <?xml version="1.0"?>
2: <note xmlns="x-schema:message04.xdr">
3:  <message number="10">
4:        Remember to buy milk on the way home from work
5:    </message>
6: </note>
```

Поскольку вы добавили атрибут в экземпляр XML, вам необходимо отразить это дополнение и в схеме XDR. Атрибуты объявляются в схемах XDR с помощью элементов `AttributeType`, синтаксис которых приведен ниже:

```
<AttributeType
    default="default-value"
    dt:type="primitive-type"
    dt:values="enumerated-values"
    name="idref"
    required="{yes|no}">
```

Сведения об атрибутах элемента `AttributeType` подытожены в табл. 5.3.

Таблица 5.3. Атрибуты элемента `AttributeType`

Атрибут	Описание
`default`	Значение атрибута по умолчанию. Это значение допустимо для данного экземпляра атрибута. Например, если атрибут относится к типу с перечислением, значение по умолчанию должно указываться в списке
`dt:type`	Указывает тип данных для атрибута определенного типа. Атрибут может относиться к одному из следующих типов: `entity`, `entities`, `enumeration`, `id`, `idref`, `idrefs`, `nmtoken`, `nmtokens`, `notation` или `string`. При выбранном типе `enumeration` также нужно указывать и атрибут `dt:type`, содержащий все допустимые значения. Подробнее с типами данных, используемыми в схемах XDR, вы познакомитесь дальше в настоящей главе
`dt:values`	Если атрибут `dt:type` задан равным `enumeration`, то атрибут `dt:type` содержит все допустимые значения
`name`	Имя типа атрибута. Этот атрибут обязателен
`required`	Указывает на обязательное наличие атрибута в описании элемента

В нашем примере атрибут относится к элементу `message`. Таким образом, элемент `ElementType` для элемента `note` будет содержать элемент `AttributeType` и элемент `attribute` для объявления нового атрибута. Синтаксис использования элемента `AttributeType` вы уже видели. Теперь рассмотрим элемент `attribute`.

Элемент `attribute` относится к объявленному типу атрибута, который представлен в области элемента `ElementType`. Синтаксис элемента `attribute` следующий:

```
<attribute
    default="default-value"
    type="attribute-type"
    [required="{yes|no}"]>
```



СОВЕТ

Допустимые атрибуты элемента `attribute` очень похожи на атрибуты элемента `AttributeType`, в чем вы можете убедиться, сравнив табл. 5.3 и 5.4.

В табл. 5.4 перечислены допустимые атрибуты элемента `attribute` в схемах XDR.

Таблица 5.4. Атрибуты элемента attribute в схемах XDR

Атрибут	*Описание*
`default`	Значение атрибута по умолчанию. Имеет преимущество перед любым значением по умолчанию, указанным в элементе `AttributeType`
`type`	Имя элемента `AttributeType`, определенного в данной схеме (или другой схеме, указанной с помощью соответствующего пространства имен). Указанное значение должно соответствовать атрибуту `name` элемента `AttributeType`
`required`	Указывает на обязательное наличие атрибута в описании элемента. Необязателен, если необходимый атрибут присутствует в соответствующем элементе `AttributeType`

Затем измените схему `message03.xdr` с тем, чтобы учесть удаление пустого элемента и добавление атрибута `number` к элементу `message`. Начните с удаления строк 7 и 10 (объявления и определения пустого элемента). Эти строки выглядят следующим образом:

```
7:   <element type="number"/>
...
10: <ElementType name="number" model="closed" content="empty"/>
```

Теперь вам необходимо изменить элемент `ElementType` для элемента `message`. Раньше этот элемент был пустым, однако теперь он должен содержать элементы `AttributeType` и `attribute` для объявления и определения нового атрибута `number`. Теперь он должен выглядеть следующим образом:

```
<ElementType name="message" model="closed" content="textOnly"
dt:type="string">
        <AttributeType name="number" dt:type="number" required="yes"/>
        <attribute type="number"/>
    </ElementType>
```

Полная схема XDR для проверки документа `message04.xml` приведена в листинге 5.8.

Листинг 5.8. Объявление и определение атрибутов в схеме XDR `message04.xdr`

```
1: <?xml version="1.0" encoding="UTF-8"?>
2: <Schema
3:  name="Untitled-schema"
4:  xmlns="urn:schemas-microsoft-com:xml-data"
5:  xmlns:dt="urn:schemas-microsoft-com:datatypes">
6:  <ElementType name="note" model="closed" content="eltOnly">
7:      <element type="message"/>
8:  </ElementType>
9:  <ElementType name="message" model="closed" content="textOnly"
    dt:type="string">
10:     <AttributeType name="number" dt:type="number" required="yes"/>
11:     <attribute type="number"/>
12: </ElementType>
13: </Schema>
```

АНАЛИЗ В строках 10 и 11 объявляется и определяется атрибут `number` элемента `message`. Элемент `AttributeType` в строке 10 подтверждает, что значение атрибута number относится к типу данных `"number"`, определяющему число, которое может состоять из любого числа цифр. Как вы узнаете из настоящей главы, существует достаточно много различных "числовых" типов данных. Атрибут `number` также необходим, на что указывает значение `"yes"` атрибута `required` в строке 10.

Типы данных в схемах XDR

В схемах XDR вы можете указывать типы данных для элементов и атрибутов. Различных типов данных существует достаточно много, что идеально подходит для построения тех или иных приложений, работающих с данными. Типы данных определяют их форматирование и разрешают синтаксическим анализаторам XML и приложениям проверять типы данных во время проверки документов на соответствие стандартам. Для того чтобы в схеме XDR могли быть определены типы данных, в объявлении корневого элемента Schema должно быть указано пространство имен типов данных *Microsoft*:

```
Xmlns:dt="urn=schemas-microsoft-com:datatypes">
```

По соглашению префикс `dt:` назначается этому пространству имен и используется в схеме XDR для идентификации типов данных. Типы данных, используемые в схемах XDR, перечислены в табл. 5.5. Список типов данных, поддерживаемых в схемах XDR, а также дополнительные сведения о проверке типов данных, вы найдете по следующему адресу.

```
http://msdn.microsoft.com/library/psdk/xmlsdk/xmlp1103.htm
```

Таблица 5.5. Типы данных для схем XDR

Тип данных	*Описание*
bin.base64	Определяет двоичные данные с использованием шифрования MIME Base64
bin.hex	Определяет двоичные данные в шестнадцатеричном формате
boolean	0 или 1, где 0 == "ложь", а 1 == "истина"
char	Строка, состоящая из одного символа
date	Указывает дату (без времени) в подмножестве формата ISO 8601. Например, 2002-11-05. Правильность указания даты не проверяется (например, дата 2-31-99 успешно пройдет проверку).
dateTime	Указывает дату (указание времени не обязательно) в подмножестве формата ISO 8601. Доли секунд можно указаны вплоть до наносекунд, например 1988-04-07T18:39:09
dateTime.tz	Указывает дату (указание времени и часового пояса не обязательно) в подмножестве формата ISO 8601. Доли секунд можно указать вплоть до наносекунд, например 1988-04-07T18:39:09-08:00
fixed.14.4	Определяет число, содержащее не больше 14 цифр слева и не больше 4 цифр справа от десятичной точки. Например 12.0044
float	Определяет действительное число (без ограничений на количество цифр); может содержать знак, дроби, а также показатель степени. Значения изменяются от 1.7976931348623157E+308 до 2.2250738585072014E-308

Тип данных	Описание
int	Целое число с необязательным знаком
number	Определяет число (без ограничений на количество цифр); может содержать знак, дроби, а также показатель степени. Значения изменяются от 1.797693134862315 7E+308 до 2.2250738585072014E-308
time	Указывает время без даты и часового пояса в подмножестве формата ISO 8601. Например 08:15:27
time.tz	Указывает время и часовой пояс (необязательно) без даты в подмножестве формата ISO 8601. Например 08:15:27–05:00
i1	Определяет целое число, представленное один байтом (знак необязателен; дробные части и знаки возведения в степень не допускаются). Например 1, 127, –128
i2	Определяет целое число, представленное одним словом (знак необязателен; дробные части и знаки возведения в степень не допускаются). Например, 1, 703, –32768
i4	Определяет целое число, представленное четырьмя байтами (знак необязателен; дробные части и знаки возведения в степень не допускаются). Например 1, 703, –32768, 148343, –1000000000
i8	Определяет целое число, представленное восьмью байтами (знак необязателен; дробные части и знаки возведения в степень не допускаются). Диапазон значений — от –92233720368547758 08 до 9223372036854775807
r4	Действительное число с точностью в семь цифр. Может содержать знак, дробную часть, а также необязательный знак степени. Диапазон значений от 1.17549435E-38F до 3.40282347E+38F
r8	Действительное число с точностью 15 цифр. Может содержать знак, дробную часть, а также необязательный знак степени. Диапазон значений от 2.2250738585072014E-308 до 1.797693134862315 7E+308
ui1	Целое число без знака, дробной части или степени. Например 1, 255
ui2	Состоящее из двух байтов целое число без знака, дробной части или степени. Например 1, 255, 65535
ui4	Состоящее из четырех байтов целое число без знака, дробной части или степени. Например 1, 703, 3000000000
ui8	Состоящее из восьми байтов целое число без знака, дробной части или степени. Диапазон значений от 0 до 18446744073709551615
uri	Унифицированный идентификатор ресурса URI (Uniform Resource Identifier), например urn:schemas-microsoft-com:Office9
uuid	Шестнадцатеричное число, разбитое на октеты, содержит необязательные переносы, которые игнорируются, например 333C7BC4-460F-11D0-BC04-0080C7055A83

Примеры типов данных

В экземпляре `message` вы добавили к элементу `message` атрибут `number`. Вы указали этот атрибут с помощью команды `dt:type="number"`, что означает, что данному ограничению соответствует любое числовое значение. Предположим, что вы хотите

убедиться в том, что указанное значение числа является целым. Для этого вам необходимо соответствующим образом изменить атрибут типа данных. В соответствии с данными из табл. 5.5 , новый атрибут должен выглядеть следующим образом:

```
dt:type="int"
```

В следующем примере вы внесете эти изменения, а также некоторые другие, чтобы поэкспериментировать с проверкой типов данных в схемах XDR. Для того чтобы удостовериться в том, что ваш программный код пройдет проверку точно так же, как и этот пример, измените значения атрибутов в файле `message04.xml` именно так, как показано дальше.

Начните с изменения документа `message03.xml`, добавив в открывающий дескриптор элемента `message` атрибуты `date` и `from`. Дата должна быть представлена в формате CCYY-MM-DD, где CC — столетие, YY — год, MM — месяц, а DD — число. Значение атрибута `from` должно быть строковым. Поэтому, внесенные вами изменения могут иметь, например, следующий вид:

```
<message number="10" date="2001-07-29" from="Kathy Shepherd">
```

Вы можете выбрать различные числа, даты и текстовые строки, однако они должны соответствовать всем ограничениям на типы данных, чтобы ваш документ смог пройти проверку на соответствие стандартам. После внесения в документ всех необходимых изменений сохраните его под именем `message05xml`, после чего внесите необходимые изменения в схему XDR. Полный документ XML со всеми внесенными изменениями приведен в листинге 5.9.

Листинг 5.9. Добавление новых атрибутов к документу `message05.xml`

```
1:  <?xml version="1.0"?>
2:  <note xmlns="x-schema:message05.xdr">
3:   <message number="10" date="2001-07-29" from="Kathy Shepherd">
4:          Remember to buy milk on the way home from work
5:      </message>
6:  </note>
```

Схема XDR должна проверить, что значение атрибута `number` является *целым числом*, значение атрибута `date` является *датой*, а значение атрибута `from` — *строкой*. Не забудьте также объявить тип содержания для элемента `message`, который включает в себя только текстовые данные типа `string`. Предположим, что каждый из этих атрибутов обязателен. Теперь вы располагаете всеми сведениями, необходимыми для указания объявлений и определений новых атрибутов, а также проверки того, что они соответствуют допустимым типам данных. Объявления и определения атрибутов должны выглядеть следующим образом:

```
10:      <AttributeType name="number" dt:type="int" required="yes"/>
11:      <AttributeType name="date" dt:type="date" required="yes"/>
12:      <AttributeType name="from" dt:type="string" required="yes"/>
13:      <attribute type="number"/>
14:      <attribute type="date"/>
15:      <attribute type="from"/>
```



В строках 13 и 14 указывается, что эти атрибуты относятся к элементу, объявляемому с помощью объекта `ElementType`. В строках 10—12 указаны типы данных. Так как все атрибуты необходимы, если только один из них отсутствует, документ XML считается неправильным.

Полный вариант схемы XDR, используемой для проверки типов данных, приведен в листинге 5.10.

Листинг 5.10. Схема XDR для проверки указанных типов данных `message05.xdr`

```
1:  <?xml version="1.0"?>
2:  <Schema
3:   name="message"
4:   xmlns="urn:schemas-microsoft-com:xml-data"
5:   xmlns:dt="urn:schemas-microsoft-com:datatypes">
6:   <ElementType name="note" model="closed" content="eltOnly">
7:       <element type="message"/>
8:   </ElementType>
9:   <ElementType name="message" model="closed" content="textOnly"
     dt:type="string">
10:      <AttributeType name="number" dt:type="int" required="yes"/>
11:      <AttributeType name="date" dt:type="date" required="yes"/>
12:      <AttributeType name="from" dt:type="string" required="yes"/>
13:      <attribute type="number"/>
14:      <attribute type="date"/>
15:      <attribute type="from"/>
16: </ElementType>
17: </Schema>
```

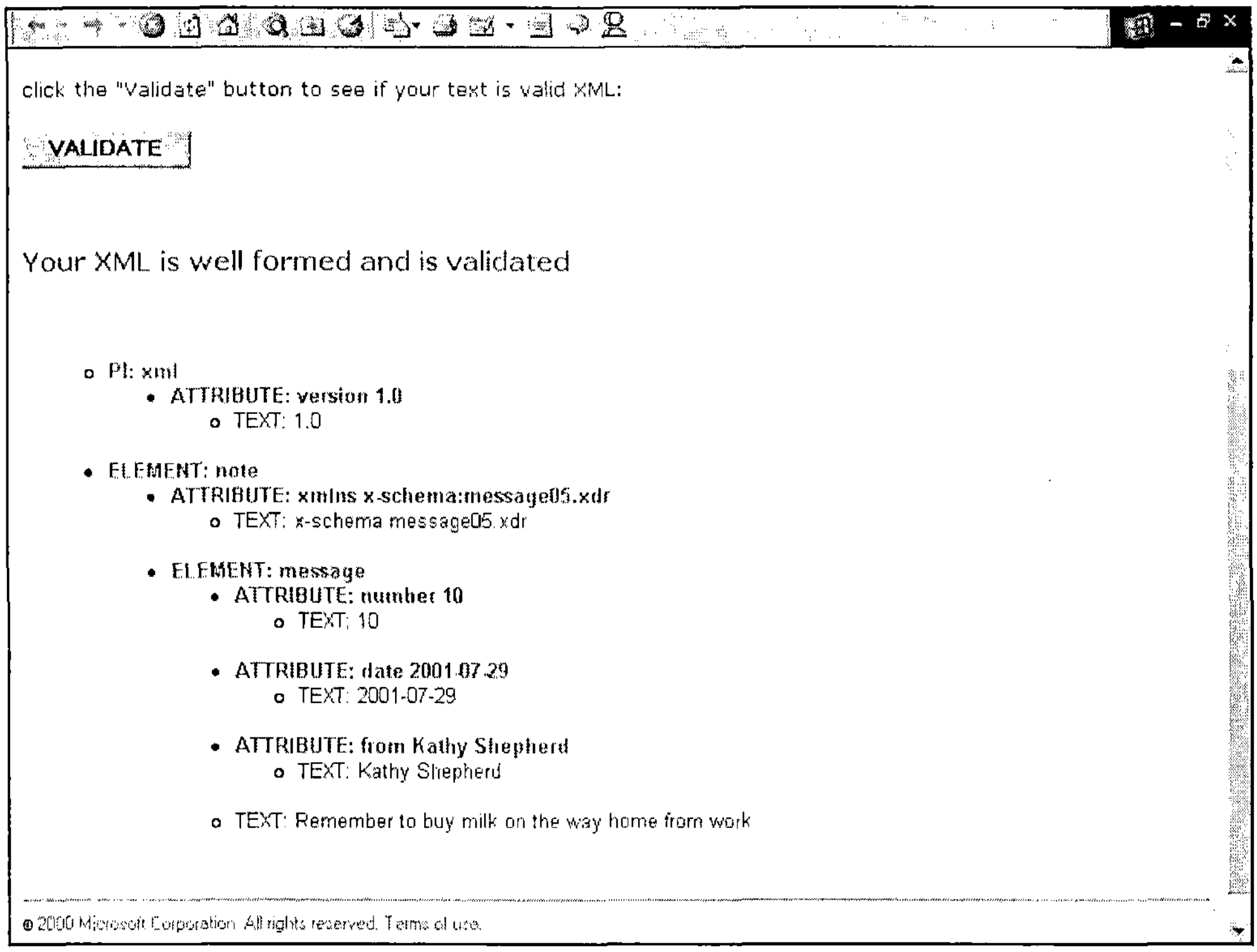


Рис. 5.2. Синтаксически правильный и соответствующий стандартам документ XML, содержащий только разрешенные типы данных

В строке 9 содержится объявление элемента `message` как содержащего только текстовые данные типа `string`. Обратите внимание на то, что типы данных `int`, `date` и `string` соответствуют атрибутам `number`, `date` и

from в строках 10 -12. В приведенном листинге эти атрибуты также указаны как обязательные. Проверьте документ message05.xml с помощью сценария, загруженного вами при изучении материала урока 4. Результаты проверки приведены на рис. 5.2.

Предположим, что вы изменили документ message05.xml таким образом, чтобы он выглядел так, как показано в листинге 5.11.

Листинг 5.11. Ошибка в типе данных message06.xml

```
1: <?xml version="1.0"?>
2: <note xmlns="x-schema:message05.xdr">
3:  <message number="ten" date="2001-07-29" from="Kathy Shepherd">
4:        Remember to buy milk on the way home from work
5:    </message>
6: </note>
```

В строке 3 значение атрибута number указано в строковом формате. Если вы "прогоните" этот документ через сценарий Microsoft XML Validator, вы увидите сообщение об ошибке, подобное тому, что показано на рис. 5.3, так как тип данных integer не допускает использования букв. Синтаксический анализатор MSXML сообщит о том, что он не может обработать знание "ten" (см. строку 3) как значение типа int.

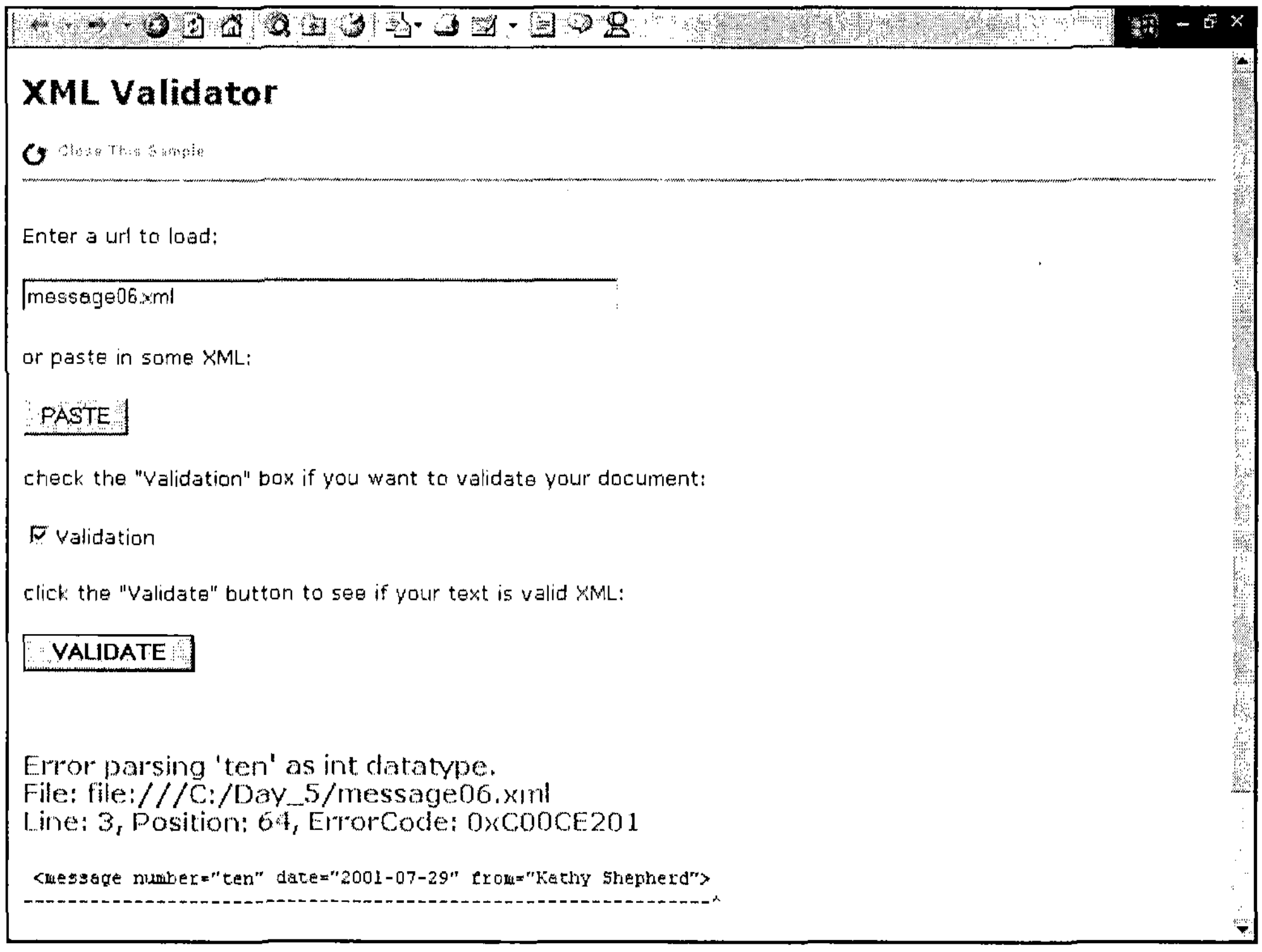


Рис. 5.3. Синтаксический анализатор MSXML сообщил об ошибке, указывающей на то, что экземпляр XML не удовлетворяет всем ограничениям в соответствующей схеме XDR

Индикаторы вхождения в схемах XDR

В схемах XDR необходимо указывать наличие нескольких дочерних элементов. Укажите, сколько раз дочерний элемент может встречаться в родительском элементе, используя атрибуты `minOccur` и `maxOccur` элемента `element`. Полный синтаксис элемента `element` приведен ниже:

```
<element
    type="element-type"
    [minOccur="{0|1}"]
    [maxOccur="{1|*}"]
```

Вы уже сталкивались с элементами `element`, которые содержали обязательный атрибут `type`. Этот атрибут связывает объявление с элементом в экземпляре XML.

Атрибут `minOccur` может принимать значения только `"0"` или `"1"`. Строка `minOccur="1"` указывает на то, что дочерний элемент должен содержаться в родительском как минимум один раз. Строка `minOccur="0"` указывает на то, что дочерний элемент необязателен.

Атрибут `maxOccur` может принимать значения только `"1"` или `"*"`. Строка `maxOccur="1"` указывает на то, что дочерний элемент должен содержаться в родительском максимум один раз. Строка `maxOccur="*"` указывает на то, что дочерний элемент может встречаться в родительском элементе неограниченное число раз.

Атрибуты `minOccur` и `maxOccur` по умолчанию равны `"1"`. Если для какого-то элемента значения этих атрибутов не заданы явным образом, используются значения по умолчанию, а значит, этот элемент должен встречаться в родительском элементе только один раз.

В следующем упражнении добавьте элементы `message` в свой документ XML, чтобы проверить все подобные ограничения с помощью схемы XDR. В листинге 5.12 вы видите, что элемент `note` теперь содержит несколько элементов `message`. Внесите необходимые изменения в уже существующий документ и сохраните его под именем `message07.xml`. Ваши элементы могут содержать элементы `message` с разными значениями, а также с различными значениями атрибутов, но постарайтесь не использовать в документе XML неправильные типы данных.

Листинг 5.12. Экземпляр XML с несколькими дочерними элементами
`message07.xml`

```
1: <?xml version="1.0"?>
2: <note xmlns="x-schema:message07.xdr">
3:  <message number="10" date="2001-07-29" from="Kathy Shepherd">
4:        Remember to buy milk on the way home from work
5:     </message>
6:  <message number="12" date="2001-07-30" from="Greg Shepherd">
7:        I need some help with my homework
8:     </message>
9:  <message number="14" date="2001-07-31" from="Kristen Shepherd">
10:        Please play Scrabble with me tonight
11:    </message>
12: </note>
```

Теперь, когда у вас есть документ XML, содержащий несколько элементов `message`, вы можете изменить схему XDR таким образом, чтобы в ней, например, указывалось наличие как минимум одного элемента `message`. Например, предположим, что этот документ является частью системы, ведущей учет сообщений, полученных на протяжении дня. Он может сохранять несколько типов данных, но для обеспечения целостности данных необ-

ходима как минимум одна запись. Схема, которую вы создадите, сможет реализовать эти ограничения благодаря использованию атрибутов `minOccur` и `maxOccur`.

Атрибуты `minOccur` и `maxOccur` относятся к дочернему элементу `element` элемента `ElementType`, относящегося к документу XML, для которого вы и создаете ограничения. Это покажется несколько запутанным, однако все это означает, что элемент `note` содержит элемент `message`, на вхождение которого накладываются определенные ограничения, в частности как минимум один элемент `message` должен существовать, однако при выполнении этого условия может существовать любое количество этих элементов. Таким образом, новое ограничение должно иметь следующий вид:

```
<element type="message" minOccurs="1" maxOccurs="*"/>
```

Добавьте эту строку в документ `message06.xdr` и сохраните его под именем `message07.xdr`. Проверьте полученный документ, загрузив файл `message07.xml` в сценарий XML Validator. Полная схема XDR, используемая для проверки документа `message07.xml`, приведена в листинге 5.13.

Листинг 5.13. Схема XDR с атрибутами `minOccur` и `maxOccur` — `message07.xdr`

```
1:  <?xml version="1.0" encoding="UTF-8"?>
2:  <Schema
3:   name="message"
4:   xmlns="urn:schemas-microsoft-com:xml-data"
5:   xmlns:dt="urn:schemas-microsoft-com:datatypes">
6:   <ElementType name="note" model="closed" content="eltOnly" or
     der="seq">
7:        <element type="message" minOccurs="1" maxOccurs="*"/>
8:   </ElementType>
9:   <ElementType name="message" model="closed" content="textOnly"
     dt:type="string">
10:       <AttributeType name="number" dt:type="int" required="yes"/>
11:       <AttributeType name="date" dt:type="date" required="yes"/>
12:       <AttributeType name="from" dt:type="string" required="yes"/>
13:       <attribute type="number"/>
14:       <attribute type="date"/>
15:       <attribute type="from"/>
16: </ElementType>
17: </Schema>
```

АНАЛИЗ

Строка 7 `<element type="message" minOccurs="1" maxOccurs="*"/>` указывает на то, что элемент `note` содержит как минимум один (`minOccurs="1"`) и может содержать любое количество (`maxOccurs="*"`) дочерних элементов `message`.

СОВЕТ

> Хотя это и удобно для некоторых приложений, присвоение атрибутам `minOccur` и `maxOccur` любых значений, отличных от указанных в схеме, на данный момент невозможно. Таким образом, нельзя определить необходимый диапазон вхождений дочерних элементов, используя атрибуты `minOccur` и `maxOccur`.

Дополнительные ограничения типов данных в схемах XDR

Если вы решили использовать тип данных `number` или `string` (или `bin.hex` или `bin.base64`), вы можете задать минимальную и максимальную длину этих атрибутов. В пространстве имен типов данных *Microsoft* существуют специальные атрибуты

minLength и maxLength, которые используются для проверки связанного документа XML. Эти атрибуты указываются в рамках элемента AttributeType; при этом используется следующий синтаксис:

```
<AttributeType name="idref"
               dt:type="(одно из следующих значений:
string|number|bin.hex|bin.base64)"
               required="(yes|no)"
               dt:minLength="положительное целое число"
               dt:maxLength="положительное целое число"/>
```

Предположим, вам необходимо ограничить размер значения атрибута from, чтобы согласовать его с параметрами поля базы данных в приложении, которое будет использовать сведения, предоставляемые вашей системой управления сообщений на основе XML. Возможно, ваша программа требует, чтобы поле from содержало не меньше 8 и не больше 15 символов. Для реализации этого ограничивающего правила вы можете использовать атрибуты minOccur и maxOccur.

В качестве примера в листинге 5.14 приведены дополнительные ограничения, налагаемые на значение атрибута from, которое должно содержать не менее 8 и не более 15 символов. Внесите эти изменения в схему XDR и сохраните ее под именем message08.xdr. Измените экземпляр XML message07.xml для связи с новым документом XDR, изменив объявление xmlns в элементе note.

Листинг 5.14. Ограничения длины, примененные к атрибуту from — message08.xdr

```
1:  <?xml version="1.0" encoding="UTF-8"?>
2:  <Schema
3:   name="message"
4:   xmlns="urn:schemas-microsoft-com:xml-data"
5:   xmlns:dt="urn:schemas-microsoft-com:datatypes">
6:   <ElementType name="note" model="closed" content="eltOnly" order="seq">
7:       <element type="message" minOccurs="1" maxOccurs="*"/>
8:   </ElementType>
9:   <ElementType name="message" model="closed" content="textOnly"
     dt:type="string">
10:      <AttributeType name="number" dt:type="int" required="yes"/>
11:      <AttributeType name="date" dt:type="date" required="yes"/>
12:      <AttributeType name="from" dt:type="string" required="yes"
13:                                      dt:minLength="8"
14:                                      dt:maxLength="15"/>
15:      <attribute type="number"/>
16:      <attribute type="date"/>
17:      <attribute type="from"/>
18:  </ElementType>
19:  </Schema>
```

АНАЛИЗ В строках 12—15 определяются ограничения, указанные в атрибуте from. В частности, в строке 13 объявляется, что значение атрибута from должно содержать минимум 8 символов. В строке 14 объявляется, что значение атрибута from должно содержать максимум 15 символов.

После того как вы внесете необходимые изменения в свои документы, загрузите их в сценарий XML Validator для проверки на наличие ошибок. Полученный результат должен напоминать то, что показано на рис. 5.4.

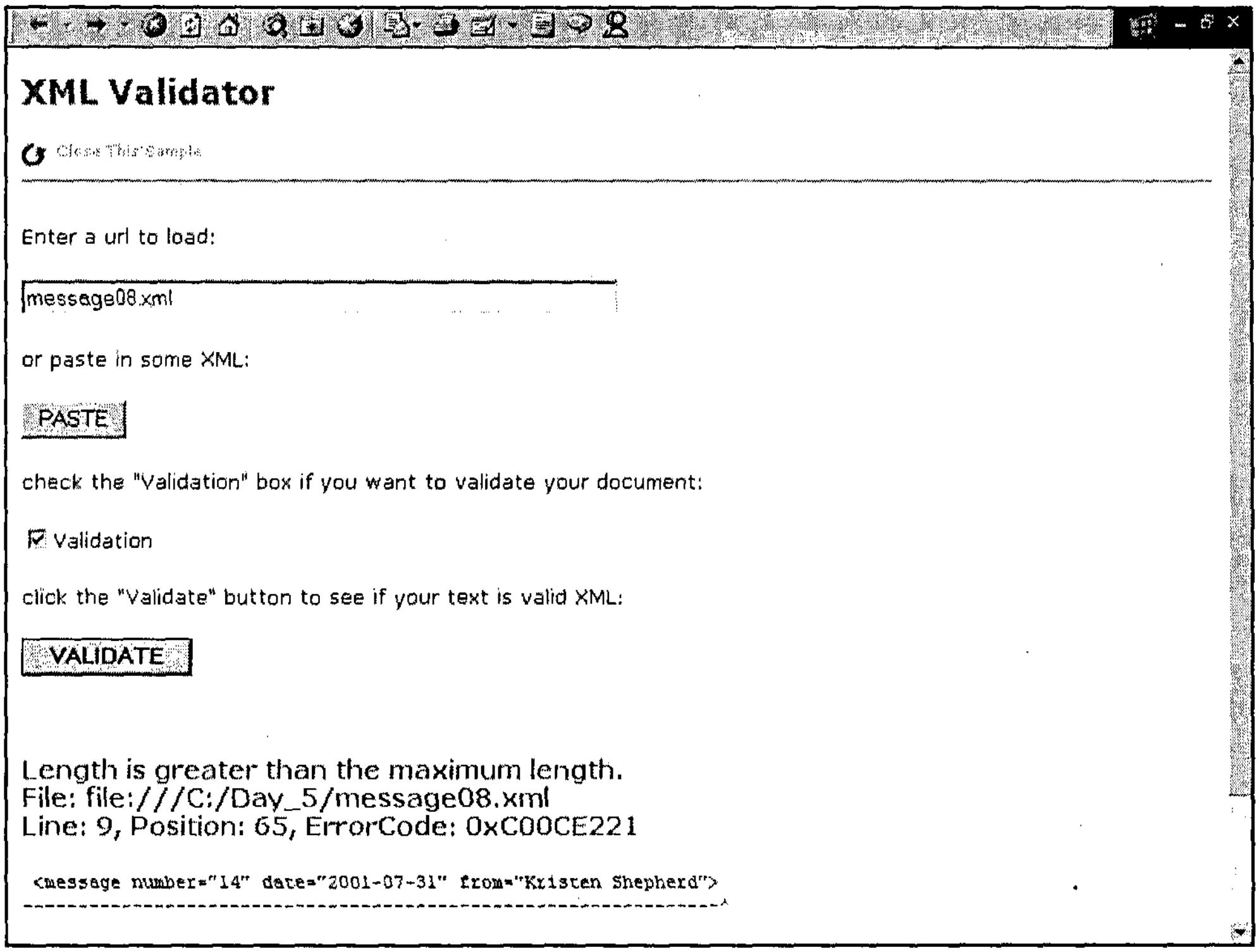


Рис. 5.4. Синтаксический анализатор MSXML сообщил об ошибке, т.е. значение атрибута выходит за рамки, указанные в соответствующей схеме XDR

Определение групп содержания

Предположим, вам необходимо проверить целый ряд необязательных элементов в экземпляре XML, используя схему XDR. Например, в вашем приложении управления сообщениями вам надо реализовать выбор между двумя вариантами, такими как `complete` и `incomplete`. Например, если элемент `message` содержит элемент `complete`, это означает, что соответствующее сообщение принято приложением полностью. Возможно, какие-то неполадки не позволяют получить сообщение целиком. В этом случае элемент `message` будет содержать элемент `incomplete`, а не элемент `complete`.

Такой подход реализован во многих приложениях электронной коммерции. Вместо того чтобы отклонить целую транзакцию, часто удобнее принять как можно больше данных и уведомить о том, что они неполные, после чего предпринять какие-то операции обработке данных. (Если вас интересуют подобные варианты обработки данных, много полезных сведений по этому вопросу, вы найдете на Web-узле `http://www.xmledi-group.org/xmledigroup/guide.htm`.)

В качестве примера предположим, что вам необходимо получать только сведения о том, что сообщение было получено полностью. Кроме того, вам необходимо, чтобы приложение проверяло экземпляр XML таким образом, чтобы в нем требовалось наличие того или иного элемента. Измените документ XML должным образом. После каждого элемента `message` добавьте новый элемент `receipt`. В рамках каждого элемента `receipt` укажите пустой дочерний элемент `complete` или `incomplete`, после

чего сохраните полученный результат под именем `message09.xml`. То, какие сообщения вы определите как полностью полученные, а какие — нет, не играет никакой роли, но убедитесь в том, что в каждом элементе `receipt` содержится только один пустой дочерний элемент `complete` или `incomplete`. Один из возможных вариантов документа XML приведен в листинге 5.15.

Листинг 5.15. Правильный экземпляр с группированием элементов `message09.xml`

```
1:  <?xml version="1.0"?>
2:  <note xmlns="x-schema:message09.xdr">
3:   <message number="10" date="2001-07-29" from="Kathy Shepherd">
4:         Remember to buy milk on the way home from work
5:   </message>
6:
7:   <receipt>
8:       <complete/>
9:   </receipt>
10:
11:      <message number="12" date="2001-07-30" from="Greg Shepherd">
12:           I need some help with my homework
13:  .   </message>
14:
15:      <receipt>
16:          <complete/>
17:      </receipt>
18:
19:
20:      <message number="14" date="2001-07-31" from="Kristen
        Shepherd">
21:          Please play Scrabble with me tonight
22:      </message>
23:
24:      <receipt>
25:          <complete/>
26:      </receipt>
27: </note>
```



В этом примере в экземпляр XML внесено совсем немного изменений. В строках 7, 15 и 24 после элементов `message` добавлены элементы `receipt`. В строках 8 и 16 добавлены новые элементы `complete`. Элемент `incomplete` добавлен в последнем элементе `receipt`, в строке 25.

Для того чтобы проверить этот документ, создайте группу содержания в схеме XDR, включающей оба элемента — `complete` и `incomplete`. Вы также создадите ограничение, которое будет требовать наличие одного из этих элементов в документе XML. Подобная группа содержания указывается с помощью элемента `group` схемы XDR. Элемент `group` позволяет вам задавать ограничения для группы дочерних элементов в рамках родительского элемента. Элемент `group` содержится в элементе `ElementType`, а для его указания используется следующий синтаксис:

```
<group order="(one|seq|many)" minOccur="(0|1)" maxOccur="(1|*)">
    <element type="ElementType"/>
    <element type="ElementType"/>
    <element type="ElementType"/>
    <element type="ElementType"/>
    <element type="ElementType"/>
</group>
```

Элемент group допускает включение атрибутов order, minOccur и maxOccur. Он может содержать любое количество дочерних элементов element. В качестве примера используем в элементе group атрибут order="one", чтобы соответствующий экземпляр XML содержал только один дочерний элемент element. Элементы group содержат дочерние элементы таким образом, что все последние относятся к семейству. Ваш элемент group будет содержать два дочерних элемента, которые выглядят следующим образом:

```
<element type"comlete"/>
<element type"incomlete"/>
```

Поскольку эти элементы element объявляют новые типы элементов в вашем экземпляре XML, вам необходимо указать в схеме XDR элемент ElementType для каждого из них. В частности, вам нужно добавить следующие элементы ElementType:

```
<ElementType name="complete" model="closed" content="empty"/>
<ElementType name="incomplete" model="closed" content="empty"/>
```

В дополнению к такому группированию элементов вам необходимо учесть новый элемент receipt, который содержит элемент complete или incomplete.

Вам необходимо внести несколько изменений в схему XDR для проверки полученного документа. Например, вам необходимо предпринять следующие действия.

- Изменить корневой элемент, включив в него новый элемент-контейнер receipt, а также определить атрибут order равным "many" (<ElementType name="note" model="closed" content="eltOnly" order="many"/>).
- Добавить элемент receipt в содержание корневого элемента (<element type="receipt"/>).
- Добавить элемент ElementType для элемента receipt, содержащий новый элемент-контейнер group с элементами complete и incomplete.

Новый элемент ElementType для элемента receipt выглядит следующим образом:

```
<ElementType name="note" model="closed" content="eltOnly"
order="many"/>
    <group order="one">
        <element type"comlete"/>
        <element type"incomlete"/>
    </group>
</ElementType>
```

Элемент ElementType интерпретируется таким образом, что элемент receipt содержит только элементы, определенные в текущей схеме (model="closed"). Элемент group разрешает выбрать один из своих дочерних элементов в качестве дочернего элемента по отношению к элементу receipt. Этими вариантами выбора являются элементы complete и incomplete.

Законченный вариант схемы XDR приведен в листинге 5.16.

Листинг 5.16. Элемент group с ограничением order="one" — message09.xdr

```
1:  <?xml version="1.0" encoding="UTF-8"?>
2:  <Schema
3:   name="message"
4:   xmlns="urn:schemas-microsoft-com:xml-data"
5:   xmlns:dt="urn:schemas-microsoft-com:datatypes">
6:   <ElementType name="note" model="closed" content="eltOnly"
7:                 order="many">
8:       <element type="message" minOccurs="1" maxOccurs="*"/>
9:       <element type="receipt"/>
10:  </ElementType>
```

```
11: <ElementType name="complete" model="closed" content="empty" />
12: <ElementType name="incomplete" model="closed" content="textOnly"
13:           dt:type="string"/>
14:      <ElementType name="message" model="closed" content="mixed">
15:      <AttributeType name="number" dt:type="int" required="yes"/>
16:      <AttributeType name="date" dt:type="date" required="yes"/>
17:      <AttributeType name="from" dt:type="string" required="yes"/>
18:      <attribute type="number"/>
19:      <attribute type="date"/>
20:      <attribute type="from"/>
21: </ElementType>
22: <ElementType name="receipt" model="closed" content="eltOnly">
23:         <group order="one">
24:          <element type="complete" />
25:          <element type="incomplete" />
26:                 </group>
27:          </ElementType>
28: </Schema>
```



В строке 9 объявляется, что новый элемент `receipt` содержится в корневом элементе `note`.

Строки 11 и 12 определяют новые пустые элементы `complete` и `incomplete`:

```
11: <ElementType name="complete" model="closed" content="empty" />
12: <ElementType name="incomplete" model="closed" content="textOnly"
```

В строке 22 начинается определение элемента `receipt`, а в строках 23–26 объявляется элемент `group` с атрибутом `order="one"`. Это указывает на то, что в документе XML должен содержаться только один элемент `complete` или `incomplete` в рамках элемента `message`, чтобы этот документ считался правильным. В строке 27 содержится закрывающий дескриптор элемента `ElementType` для элемента `receipt`.

Дополнительные преимущества схем XDR

Сегодня вы изучили синтаксис языка описания схем XDR. Вы убедились в том, что схемы XDR конструируются с помощью словаря, использующего синтаксис XML. Другими словами, язык описания схем использует те же способы описания элементов атрибутов, которые используются в самих экземплярах XML. Этот словарь содержится или в пространстве имен XML-Data Reduced (`urn:schemas-microsoft-com:xml-data`) или пространстве имен типов данных *Microsoft* (`urn:schemas-microsoft-com:datatypes`). Вам, как разработчику, предлагается целый ряд преимуществ. Поскольку вы уже знакомы с синтаксическими правилами XML, сосредоточьтесь на семантике словаря XML-Data Reduced, а не семантике собственно разметки. При создании схем DTD вам приходилось изучать и то, и иное. Например, в схемах XDR не используются расширенные наборы символов, обладающие уникальным значением. Схемы XDR можно "проверить" с помощью синтаксического анализатора, что контролирует в них выполнение синтаксических правил, найти в схемах пропуски завершающих символов, угловых скобок или неправильно оформленные дескрипторы. Все это позволяет вам сэкономить массу времени.

Также достойны упоминания и некоторые другие характеристики XML. В настоящем разделе я укажу только на наиболее значимые из них.

Расширяемость в схемах XDR

Схемы XML-Data Reduced расширяемы, поскольку базируются на моделях открытого содержания, в которых используется синтаксис XML, расширяемый по умолчанию. Это дает значительные преимущества по сравнению со схемами DTD, с которыми вы познакомились при изучении урока 4. Авторы документов расширяют схемы XDR, а также могут добавлять пользовательские элементы и атрибуты в экземпляры схем XDR. Если вам необходимо расширить схему XDR, вы должны включить пользовательское объявление типа данных, чтобы отличать пользовательские элементы и атрибуты от стандартных элементов и атрибутов XDR. Как уже отмечалось раньше, стандартный словарь схем XDR определяется пространствами имен данных и типов данных. При расширении языка вы расширяете пространство имен таким образом, чтобы оно содержало ваши элементы и атрибуты.

Схемы XDR обычно расширяются только в тех ситуациях, когда стандартный словарь XDR не обеспечивает простого способа ограничений, необходимых для решения определенной задачи. Например, если рассмотреть пример системы управления сообщениями, с которой вы познакомились в настоящей и предыдущей уроках, можно предположить, что система за данный промежуток времени в состоянии принять и обработать ограниченное количество сообщений. Вам понадобится добавить дополнительное ограничение для объявления элемента `limit`. Вы можете объявить элемент `limit` и назначить ему тип данных `int`, а также отнести его к пространству имен `mystuff`. Один из возможных вариантов заключается в использовании объявления `ElementType`, как показано ниже:

```
<ElementType name="limit" xmlns:mystuff="urn:mystuff-
extensions:limits">
    <datatype dt:type="int">
    <mystuff:max>100</mystuff:max>
</ElementType>
```

В этом фрагменте кода показан новый элемент `mystuff:max`, принадлежащий к пространству имен `urn:mystuff-extensions:limits`. Несмотря на то, что в ходе проверки подтверждается только то, что значение элемента `limit` является целым, ваше приложение может использовать сведения, предоставляемые пользовательскими элементами из вашего собственного пространства имен, для выполнения дополнительных проверок. Подробные сведения о создании собственных пространств имен вы найдете в уроке 8.



Если вы расширили схему XDR, синтаксический анализатор MSXML не подтвердит соответствие вашего документа всем синтаксическим правилам и стандартам, выполнив эти действия для основных типов данных.

Преобразование схем XDR

Используя преобразования XSLT, вы сможете программным способом преобразовать экземпляр XML в другой экземпляр XML, документ HTML или любую другой форму разметки. С преобразованиями XSLT вы познакомитесь в уроке дня 16-го.

Поскольку схемы XDR создаются с использованием синтаксических правил XML, их также можно преобразовать к другим диалектам развертки. Одна из причин сделать это состоит в преобразовании экземпляра XDR в экземпляр XSD (XML Schema Definition). Компания *Microsoft* предлагает соответствующую программу, полезную для тех, кто уже изучил XDR, но хочет изучить еще и XSD. Как уже отмечалось раньше, существуют и коммерческие программы подобного рода, например XML-Spy (`http://www.xmlspy.com`). При использо-

вании схем DTD подобные преобразования невозможны. Для преобразования схем DTD на XDR, XSD или другие языки описания схем потребуется использование приложений, выходящих за рамки технологий XML.

Резюме

Сегодня вы познакомились с языком описания схем XDR (XML-Data Reduced). Вы убедились в значительных отличиях между схемами XDR и DTD. В частности, схемы XDR представляют собой документы, созданные с использованием синтаксиса XML, что позволяет применять к ним различные средства проверки, а также ограничения типов данных. Схемы XDR представляют собой полноценные экземпляры XML, поэтому их можно преобразовать к другим диалектам языков разметки точно так же, как и обычные документы XML.

Вопросы и ответы

Предназначены ли схемы XDR и другие базирующиеся на технологиях XML схемы для замены схем XML?

Схемы, написанные с использованием грамматики XML, обладают значительными преимуществами перед схемами DTD. В то же время, схемы DTD достаточно широко распространены, а многие сложные системы полагаются на стабильность проверенных временем схем DTD.

Какие специфические функции выполняют схемы XDR?

Схема определяет структуру документа XML, налагая на него определенные ограничения. При этом схема обеспечивает наличие необходимых элементов и атрибутов, а также проверяет отношения между ними. Схемы XDR обеспечивают достаточно сложные способы проверки типов данных.

Какова роль синтаксического анализатора MSXML в схемах XDR?

Разработанный компанией *Microsoft* синтаксический анализатор MSXML входит в состав поставки Web-броузера Internet Explorer. Используя это средство, вы сможете проанализировать документ XML как со связанной с ним схемой, так и без нее.

Что такое *идентификатор URI*?

Идентификатор URI — это всего лишь метка, которая позволяет отличить одно пространство имен от другого. Подробные сведения об объявлении и синтаксисе пространств имен вы найдете в уроке 8.

Упражнение

Проверьте свои знания, полученные в результате изучения материала настоящей главы. Ответ вы найдете в Приложении А “Ответы на вопросы”.

В уроке дня 4-го вы создали схему DTD для языка разметки MCML. Создайте схему XDR для проверки файлов MCML.

День 6-й

Язык определения схем XML

В двух предыдущих уроках вы создавали схемы DTD и XDR для проверки экземпляров документов XML. Вы создавали ограничения, определяющие структуру документов и их соответствие определенным правилам. Вы исследовали различия между схемами DTD и схемами XDR, использующими синтаксис XML. Сегодня рассмотрим следующие темы.

- Основные черты языка определения схем XML (XSD).
- Создание схем XSD для определения ограничений для элементов и атрибутов.
- Выбор схемы для решения определенной задачи.

Схема, предложенная консорциумом W3C

Вчера, в уроке 5, вы познакомились с рядом преимуществ схем XDR перед схемами DTD. Вы также узнали о существовании других языков описания схем. Подход с использованием схем XDR очень важен, поскольку широко используется в различных решениях для электронной коммерции, например, BizTalk. Однако язык описания схем XDR требует использования синтаксического анализатора MSXML, что ограничивает реализацию решений для других платформ, а также интеграции с определенными наследуемыми моделями данных.

Конечно же, схемы DTD используются очень широко. Проблемы с использованием этих схем начинаются уже с того, что их нельзя выразить с помощью синтаксиса XML, что требует от разработчиков изучения нового языка программирования, использования специальных инструментов и т.д. Кроме того, схемы DTD не поддерживают проверку типов данных.

Консорциум *W3C* выработал рекомендацию языка определения схем XML (XSD), объединив наиболее популярные языки описания схем в один стандарт. Основная цель, которая при этом преследовалась, — получение стандарта, который можно широко реализовать и при этом он *платформно-независимый*. Время покажет, сбудутся ли эти ожидания.

Теперь, когда консорциум *W3C* опубликовал полные спецификации языков описания схем, он надеется на широкую поддержку языка XSD со стороны новых продуктов, а также тех продуктов, в которых в настоящее время для проверки документов используются другие схемы. Например, компания *Microsoft* пообещала реализовать в своем броузере поддержку XSD, а не только своего "фирменного" MSXML. На момент написания книги текущие версии синтаксического анализатора MSXML (4.0) и Microsoft XML 4.0 Parser SDK обеспечивают поддержку XSD. Компании *IBM*, *Sun* и *Apache*, относящиеся к основным игрокам на этом рынке, также уже выпустили син-

таксические анализаторы с поддержкой XSD. Скоро должны появиться решения и от других компаний. Конечно же, схемы DTD и XDR будут широко поддерживаться до тех пор, пока разработка всех необходимых средств для использования схем XSD не завершится и они не получат широкой поддержки.

Язык XML Schema Definition Language, который также называют XML Schema Language, во многом похож на язык XDR, с которым вы познакомились раньше. Схемы XSD способны решать следующие задачи.

- Перечисление элементов в документе XML и проверка наличия в документе только объявленных элементов.
- Объявление и определение атрибутов, модифицирующих элементы документа.
- Определение родительско-дочерних отношений между элементами.
- Определение состояний и моделей содержания для элементов и атрибутов.
- Задание типов данных.
- Установка значений по умолчанию.
- Возможность расширения.
- Поддержка использования пространств имен.

Статус языка XSD

Сегодня мы рассмотрим основные аспекты языка XSD. Основная цель, которую я преследую в настоящей главе, — снабдить вас всей необходимой информацией для сравнения XSD с другими подходами. Многие аналитики предсказывают быстрый рост популярности XSD в результате того, что он уже получил статус стандарта консорциума *W3C*. Превращение определенной технологии в стандарт *W3C* можно разделить на несколько этапов. С подобными этапами становления XSD как стандарта вы познакомитесь в следующем разделе.



Очень вероятно, что изложенные в настоящей главе сведения претерпят изменения после выхода книги. Последнее обновление спецификации XSD вы найдете по адресу `http://www.w3.org/XML/Schema`.

Рекомендация консорциума W3C

Несколько технологий, представленных в настоящей книге, по крайней мере на момент написания книги, еще не получили статуса официального стандарта. 2 мая 2001 года. XSD получил статус рекомендации консорциума *W3C*. Как известно, консорциум *W3C* (`http://www.w3.org`) — это организация, отвечающая за создание, публикацию и поддержку стандартов, касающихся различных Web-технологий. Стандартизация консорциумом *W3C* требует выполнения целого ряда шагов.

Консорциум *W3C* под понятием "рекомендация" (recommendation) понимает "стандарт" (standard). Получение определенной технологией такого статуса означает прохождение нескольких этапов. Приведенные ниже разделы взяты и официального материала, доступного на Web-узле консорциума *W3C* по адресу `http://www.w3.org/Consortium/Process-20010208/tr`.

- Working Draft (Рабочий черновик)
- Last Call Working Draft (Рабочий черновик: последний вариант)
- Candidate Recommendation (Кандидат в рекомендации)
- Proposal Recommendation (Предложенная рекомендация)
- W3C Recommendation (Рекомендация W3C)

Основы XSD

Как вы узнали из уроков 4 и 5, схемы, независимо от используемых для их создания словарей и синтаксических правил, предназначены для задания определенных ограничений на документы XML, а значит, для обеспечения соответствия последних определенным правилам. Языки описания схем, которые базируются на синтаксисе XML, обладают определенными преимуществами перед DTD, поскольку допускают расширение используемых дескрипторов разметки, а также проверку документов и схем с помощью стандартного синтаксического анализатора XML.

В схемах XSD дескрипторы, используемые в документах XML, разделяются на две категории — сложные типы и простые типы. Элементы сложных типов могут содержать другие элементы, а также обладают определенными атрибутами; элементы простых типов такими возможностями не обладают. Пример элементов простого типа приведен ниже:

```
<message>Remember to buy milk on the way home from work</message>
<delivery>email</delivery>
```

Приведенный ниже фрагмент содержит элементы, которые могут считаться сложными, поскольку содержат другие элементы или атрибуты:

```
<message number="10" date="2001-07-29" from="Kathy Shepherd">
         Remember to buy milk on the way home from work
         <receipt>
             <complete/>
         </receipt>
</message>
```

Элемент `message` сложного типа, он содержит атрибуты `number`, `date` и `from`. Он содержит элементы и данные, а значит, у него смешанное содержание. Элемент `receipt`, дочерний по отношению к элементу `message`, содержит пустой дочерний элемент `complete`.

Подробно с использованием элементов сложного и простого типов вы познакомитесь дальше в настоящем уроке.



В уроках 4 и 5 вы узнали, что правила схем, применяемые к элементам и атрибутам, состоят из двух различных шагов. Это справедливо и по отношению и к XSD. Эти два шага — определение и объявление типов — или типов элементов, или типов атрибутов.

Пространства имен XSD

Подробно с концепцией пространств имен вы познакомитесь в уроке дня 8-го. однако сейчас вам, как и в случае со схемами XDR, необходимо указать стандартное пространство имен для всех экземпляров XSD. Это пространство имен идентифицирует набор элементов и атрибутов, которые образуют словарь XSD.

Корневым элементов в схеме XML является элемент `Schema`, который содержит все остальные элементы в документе схемы. В рамках корневого элемента схемы XSD вы создаете пространства имен, используя атрибут `xmlns`. Этот префикс, который добавляется к именам типов элементов в экземпляре XSD. Текущее определение схемы XML вы сможете найти по адресу `http://www.w3c.org/1999/XMLSchema`. Например, приведенный ниже дескриптор следует соглашению об использовании префикса `"xsd"` для связи элементов с именованной коллекцией:

```
<xsd:schema xmlns:xsd="http://www.w3c.org/1999/XMLSchema"
```



Префикс `xsd` (аббревиатура от XML Schema Definition) используется в документе XSD для пространства имен XML Schema.

Экземпляр документа XML, который проверяется с помощью схемы, также должен содержать объявление пространства имен. Пространство имен всегда указывается в корневом элементе экземпляра документа XML с помощью атрибута `xmlns`:

```
xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance"
```

Это пространство имен содержит элементы и атрибуты XML Schema, которые можно включать экземпляр документа XML. По общему соглашению префикс `xsi` используется для этого пространства имен и добавляется в начале имен всех элементов и атрибутов, принадлежащих пространству имен, отделяясь от них двоеточием.

Для связи экземпляра документа XML со схемой чаще всего используются два атрибута — `xsi:schemaLocation` и `xsi:noNamespaceSchemaLocation`. Эти атрибуты позволяют вам связать документ со стандартом XML Schema консорциума *W3C*. Такое связывание не будет строго обязательным, можно быть использовать и другие зависящие от конкретных приложений механизмы (такие как параметры командной строки), однако оно позволят поддерживающим стандарт XML Schema консорциума *W3C* инструментам, таким как синтаксические анализаторы, быстрее находить схему.

Если документ XSD связывается без пространства имен — например, полностью определенный идентификатор URI или локальный файл, — используется атрибут `xsi:noNamespaceSchemaLocation`:

```
xsi:noNamespaceSchemaLocation="имя_файла.xsd"
```

С другой стороны, пространство имен может быть объявлено вместе с именем файла, тогда идентификатор URI для пространства имен идентификатор URI для схемы разделяются пробелами, образуя значение одного атрибута, как показано ниже:

```
xsi:schemaLocation="http://example.org/ns/books/ имя_файла.xsd"
```

Это тяжело увидеть в предыдущем примере, но пробелом разделены такие части значения, как пространство имен (`http://example.org/ns/books/`) и имя документа схемы (`имя_файла.xsd`).

Простые типы элементов

В языке XSD, в отличие от тех двух, с которыми вы познакомились раньше, существует концепция *именованных типов*. Например, когда вы создаете определение, вы можете присвоить этому определению *имя*, чтобы повторно использовать его в экземпляре XSD. Вспомните пример фрагмента XML, содержащего два простых элемента:

```
message>Remember to buy milk on the way home from work</message>
<delivery>email</delivery>
```

Вы можете создать определение элемента простого типа `simpleType` и назвать его, например, `msg`. В результате вы получите *именованное ограничение*. После этого вы сможете применять это ограничение и к другим элементам в схеме. Например, элемент `simpleType` может быть связан с элементами `message` и `delivery` для объявления содержания этих элементов как строковых данных. Например, это может выглядеть следующим образом:

```
<xsd:simpleType name="msg" base="xsd:string"/>
<xsd:element name="message" type="msg"/>
<xsd:element name="delivery" type="msg"/>
```

Прежде всего, вы должны обратить внимание на то, что все элементы XSD начинаются с префикса `xsd:`, который указывается для пространства имен XSD, объявленного в корневом элементе экземпляра схемы. В строке 1 элемент простого типа определяется элементом `xsd:simpleType` с именем `msg`. Атрибут `base` этого элемента ограничивает содержание только данными типа `string`. В строках 2 и 3 объявляются элементы `message` и `delivery` соответственно.



Когда вы создаете *именованное ограничение*, вы определяете имя — оно не содержится в вашем документе XML. Имя представляет собой ссылку на отдельное ограничение XSD.

На практике многие авторы документов предпочитают полностью определять все простые типы, добавляя необходимые атрибуты в элемент `xsd:element`, а не в отдельный элемент `xsd:simpleElement`. Это особенно полезно в том случае, если для полного определения ограничений, накладываемых на элементы, необходимо всего один или два атрибута. Вы можете или использовать элемент `xsd:simpleElement`, или исключить его и добавить необходимые атрибуты к элементу `xsd:element`; допустим любой из этих подходов. Если вы предпочитаете последний вариант, предыдущий пример будет выглядеть следующим образом:

```
<xsd:element name="message" type="xsd:string"/>
<xsd:element name="delivery" type="xsd:string"/>
```

В этом примере атрибуты `type` больше не содержат имя, связанное с элементом `xsd:simpleElement`; вместо этого они содержат полное определение типа данных. Обратите внимание на то, что значение атрибута содержит префикс `xsd:`, который указывает на то, что определение строки относится к пространству имен `xsd`. Таким образом, объявление типа данных string не конфликтует с ссылкой на элемент `string` типа `simpleElement`. Пространства имен тем и хороши, что предотвращают возникновение подобных противоречий, называемых *коллизией имен*.

В листинге 6.1 показан вариант созданного вами раньше документа, в который добавлены объявления, необходимые для использования схем XSD.

Листинг 6.1. Экземпляр XML, в котором показано пространство имен XSD, `message01.xml`

```
1: <?xml version="1.0"?>
2: <note
3:  xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance"
4:  xsi:noNamespaceSchemaLocation="message01.xsd">
5:  Remember to buy milk on the way home from work
6: </note>
```



Открывающий дескриптор элемента `note` содержится в строках 2–4. В строке 3 указано пространство имен для ссылки на экземпляр XML Schema. В строке 4 содержится ссылка на файл, используемые при проверке документа.

Как уже отмечалось раньше, все элементы в схеме XSD начинаются с префикса `xsd:`. Этот префикс указывается для пространства имен XSD посредством объявления `xmlns:xsd="http://www.w3.org/2000/10/XMLSchema"`. Это определение должно содержаться в открывающем дескрипторе корневого элемента (`xsd:schema`) документа схемы XSD. Как и в случае префиксов всех других пространств имен, назначение префикса — идентификация документов, которые принадлежат словарю языка XDR, а не словарю, созданному автором схемы. Подробно о пространствах имен мы поговорим в уроке 8. Не-

смотря на то, что пространство имен содержит фрагменты, напоминающие Web-адреса, обработчики XML не воспринимают их подобным образом.

Создав файл `message 01.xml` с помощью текстового редактора, создайте файл схемы XDR, сохранив его под именем `message 01.xsd`. Пример схемы XSD приведен в листинге 6.2.

Листинг 6.2. Схема XSD для проверки документа `message01.xsd`

```
1: <?xml version="1.0"?>
2: <xsd:schema
3:  xmlns:xsd="http://www.w3.org/2000/10/XMLSchema"
4:  elementFormDefault="qualified">
5:  <xsd:element name="note" type="xsd:string"/>
6: </xsd:schema>
```



В строке 3 показано объявление пространства имен XSD в качестве атрибута `xmlns` корневого элемента `xsd:schema`. В строке 4 объявляется и определяется элемент `note` с типом данных `string` с пространства имен XSD.

Типы данных XSD

Язык XSD содержит большое количество встроенных простых типов данных точно так же, как и язык XDR. В дополнение к подобным встроенным типам данных, таким как `string` или `decimal`, XSD предоставляет возможность создания новых типов данных, основанных на встроенных. Автор схемы может создать подобные производные типы данных и использовать их в документе XSD. Полный список простых типов данных XSD вы можете найти по адресу `http://www.w3.org/TR/xmlschema-0/`. Часто используемые простые типы данных перечислены в табл. 6.1.

Таблица 6.1. Простые типы данных, используемые в схемах XSD

Простой тип	*Описание*
string	Буквенно-цифровая строка
normalizedString	Строка без пробелов
byte	–1126
unsignedByte	0,126
base64Binary	GpM7
hexBinary	OFB7
integer	–126789, –1, 0, 1, 126789
positiveInteger	1, 126789
negativeInteger	–1267879, –1
nonNegativeInteger	0, 1, 126789
nonPositiveInteger	–1267879, –1, 0
int	–1, 126789675
unsignedInt	0, 1267896754
long	–1, 12678967543233
unsignedLong	0, 12678967543233

Простой тип	*Описание*
decimal	–1.23, 0, 123.4, 1000.0
float	–INF, –1E4, –0, 0, 12.78E–2, 12, INF, Nan
double	–INF, –1E4, –0, 0, 12.78E–2, 12, INF, Nan
boolean	True, false, 1, 0
time	13:20:00.000, 13:20:00.000–5:00
dateTime	1999–05–31T1313:20:00.000–5:00
duration	P1Y2M3DT10H30M12.S3
date	1999–05–31
gMonth	--05--
gYear	1999
gYearMonth	1999-02
gDay	---31
gMonthDay	--05-31
Name	shipTo
Qname	po:USAdress
NCName	USAdress
anyURI	http://www.example.com http://www.example.com/doc.html#ID5

Определения элементов сложных типов

Простые или сложные типы элементов — это уникальные характеристики языка XSD. Как уже отмечалось, сложные типы элементов, используемые в схемах XSD, позволяют элементам содержать другие элементы и атрибуты.

В следующем примере вы увидите, как используются элементы сложного типа в документе, которым вы манипулируете на протяжении последних нескольких дней. В листинге 6.3 приведен измененный экземпляр XML, в котором элемент `note` содержит элемент `message`. Так как один элемент содержит другой элемент, в схеме XSD необходимо использовать элемент сложного типа. Вы должны помнить, что элементы `simpleType` соответствуют элементам, содержащим значения, но не другие элементы и атрибуты.

Листинг 6.3. Документ XML, в котором содержится элемент с вложенным элементом, `message02.xml`

```
1: <?xml version="1.0"?>
2: <note
3:  xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance"
4:  xsi:noNamespaceSchemaLocation="message02.xsd">
5:  <message>Remember to buy milk on the way home from work</message>
6: </note>
```

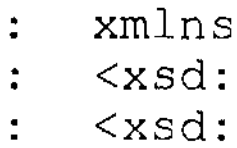


В этом экземпляре XML корневой элемент `note` (строки 2—6) считается сложным, поскольку он содержит дочерний элемент `message` в строке 5. Атрибут `xsi:noNamespaceSchemaLocation` равен `"message02.xsd"`, связывая файл XSD с экземпляром.

Базируясь на концепции сложного элемента, созданная вами схема XSD для этого документа должна устанавливать отношения между элементами `note` и `message`. В частности элемент `message` является дочерним по отношению к элементу note. Фрагмент схемы XSD приведен ниже.

Листинг 6.4. Фрагмент схемы XSD, содержащей элемент сложного типа

```
1:  <xsd:element name="имя_родительского_элемента">
2:   <xsd:complexType>
3:        <xsd: элемент name="имя_дочернего_элемента"
          type="simple_type_ref"/>
4:   </xsd:complexType>
5:  </xsd:element>
```



В строке 1 начинается объявление родительского элемента сложного типа. Элемент `complexType` занимает строки 2—4 и содержит дочерние элементы. В приведенном фрагменте схемы XSD показан только один дочерний элемент, в строке 3. Имя дочернего элемента в экземпляре документа XML определяется значением атрибута `name` элемента `xsd:element`. В этом случае атрибут `type` ссылается на объявление дочернего элемента простого типа. Если дочерний элемент сложный, атрибут `type` будет ссылаться на элемент `complexType` в схеме XSD.

В листинге 6.5 приведена схема XSD для экземпляра документа XML. Однако в этом случае было добавлено несколько новых элементов по сравнению с краткой формой схемы, приведено раньше. Например, добавлен элемент `xsd:sequence`, он позволяет указать последовательность дочерних элементов. Приведенный пример содержит только один дочерний элемент `xsd:element`, объявляющий элемент `message` в документе XML. Однако в тех ситуациях, когда элемент `note` содержит несколько дочерних элементов, заключение этих элементов в элемент `xsd:sequence` позволяет вам контролировать порядок дочерних элементов документе XML.

Листинг 6.5. Схема XSD, содержащая элемент xsd:ComplexType, `message02.xsd`

```
1:  <?xml version="1.0"?>
2:  <xsd:schema
3:   xmlns:xsd="http://www.w3.org/2000/10/XMLSchema">
4:   <xsd:element name="message" type="xsd:string"/>
5:   <xsd:element name="note">
6:        <xsd:complexType>
7:             <xsd:sequence>
8:                  <xsd:element ref="message"/>
9:             </xsd:sequence>
10:       </xsd:complexType>
11:  </xsd:element>
12:  </xsd:schema>
```



Сложный тип определяется в строках 7—10. В строке 7 объявляется последовательность, хотя на самом деле в схеме содержится всего один дочерний элемент, `message`. Если в родительском элементе содержится несколько дочерних, элемент `xsd:sequence` можно использовать для определения порядка следования дочерних элементов. Другими словами, элемент `xsd:sequence` только указывает на то, что дочерние элементы должны быть представлены в документе XML именно в том порядке, в котором они указаны в схеме XSD. В строке 8 атрибут `ref` используется

для ссылки на определение простого типа для элемента `message` из строки 4. Атрибут `ref` играет ту же роль, что и атрибут `type` в предыдущих примерах, позволяя автору схемы определять элементы всего один раз, даже если они неоднократно используются в объявлениях.

Ограничения вхождений в схемах XSD

В отличие от других языков описания схем, XSD позволяет вам определить количество вхождений элемента с определенной точностью. Вы можете задать минимальное и максимальное количество вхождений элемента с помощью атрибутов `minOccur` и `maxOccur` элемента `xsd:element` соответственно. Использование этих атрибутов может показаться похожим на их использование в предыдущем уроке, но на этом все сходство заканчивается. В случае языка XSD на возможные значения этих атрибутов налагаются определенные ограничения. Это связано с тем, что консорциум *W3C* определил желательный диапазон вхождений. Это одна из наиболее отличительных характеристик XSD по отношению к XDR, DTD и другим языкам описания схем.

Если ничего другого не указано, то значения по умолчанию атрибутов `minOccur` и `maxOccur` равны `"1"`. Атрибут `maxOccur` также может принимать значение `"unbounded"`, что аналогично значению "*" в языках DTD и XDR.

Давайте рассмотрим документ XML, представленный в листинге 6.6. Этот документ содержит несколько вложенных дочерних элементов. Корневой элемент `note` содержит два элемента `notes`, каждый из которых содержит по пустому элементу `number`, после которого следуют элементы `message`.

Листинг 6.6. Экземпляр XML со сложным вложением `message04.xml`

```
1:  <?xml version="1.0"?>
2:  <note
3:   xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance"
4:   xsi:noNamespaceSchemaLocation="message04.xsd">
5:   <notes>
6:       <number/>
7:       <message>Remember to buy milk on the way home from
        work</message>
8:       <message>Skim Milk is preferred</message>
9:   </notes>
10:  <notes>
11:      <number/>
12:      <message>Pick up shirts from the cleaners</message>
13:      <message>Got to the bank</message>
14:      <message>Cut the lawn</message>
15:  </notes>
16:  </note>
```

Предположим, вам необходимо наложить на этот документ ряд ограничений, к которым относятся следующие.

- В документе допускается не больше двух элементов `notes`.
- Вхождение элементов `notes` необязательно.
- Элемент `number` должен предшествовать элементу `message`.
- Должен существовать как минимум один элемент `message`.

Для создания схемы XSD, отражающей все эти ограничения, вам необходимо указать количество элементов, порядок их следования, а также обязательные и необязательные

элементы. Для того чтобы задать ограничение на количество элементов, используйте атрибуты `minOccur` и `maxOccur` соответствующих элементов `xsd:element`. В XSD вы можете добавить ограничения, касающиеся обязательного наличия элементов, используя те же самые атрибуты `minOccur` и `maxOccur`. Например, если вы зададите атрибут `minOccur` равным 0 для определенного элемента или вообще исключите этот атрибут, чтобы использовать его значение по умолчанию, этот элемент будет считаться обязательным. (Обязательным является как минимум одно вхождение элемента.) Вы уже знаете, что значение больше 1 для атрибута `maxOccur` обозначается как `"unbound"`.

В листинге 6.7 показан возможный вариант схемы XSD, отражающей перечисленные раньше ограничения.

Листинг 6.7. Атрибуты `minOccur` и `maxOccur` в схеме XSD — `message04.xsd`

```
1:  <?xml version="1.0"?>
2:  <xsd:schema xmlns:xsd="http://www.w3.org/2000/10/XMLSchema">
3:   <xsd:element name="message" type="xsd:string"/>
4:   <xsd:element name="number"/>
5:   <xsd:element name="note">
6:       <xsd:complexType>
7:           <xsd:sequence>
8:               <xsd:element name="notes" minOccurs="0"
                 maxOccurs="2"/>
9:           </xsd:sequence>
10:      </xsd:complexType>
11: </xsd:element>
12: <xsd:complexType name="notesType">
13:      <xsd:sequence>
14:          <xsd:element ref="number"/>
15:          <xsd:element ref="message" maxOccurs="unbounded"/>
16:      </xsd:sequence>
17: </xsd:complexType>
18: </xsd:schema>
```

АНАЛИЗ В строке 8 определяются ограничения вхождений (`minOccurs="0" maxOccurs="2"`) для дочернего элемента `notes` корневого элемента `note`. В частности, проверяемый экземпляр должен содержать от нуля (`minOccurs="0"`) до двух (`maxOccurs="2"`) дочерних элементов `notes`. Так как в качестве минимального значения указан 0, дочерний элемент `notes` необязателен.

В строке 15 указано, что как минимум один элемент `message` должен следовать после элемента `number` в последовательности, содержащейся в каждом родительском элементе `notes`. Это задается с помощью элемента-контейнера `xsd:sequence`. Однако значение `"unbound"` указывает на то, что количество элементов `message`, по большому счету, неограниченно.

В строке 12 объявляется производный сложный тип `notesType`. Этот новый сложный тип является ненастоящим типом элемента, а связью между именем и ограничениями, определяющими "поведение" именованного типа в экземпляре документа. Другими словами, тип `notesType` в документе XML не существует; вместо этого элементы `notes` должны удовлетворять ограничениям, которые налагаются с помощью сложного типа `notesType`. Таким образом, элементы `notes` должны содержать только один пустой элемент `number`, после которого может следовать любое количество элементов `message`. Сложная последовательность называется `notesType` только для удобства; вы можете называть ее, как только захотите.

Атрибуты в схемах XSD

Объявление атрибутов в схемах XSD очень похоже на объявление элементов, за исключением того, что атрибуты объявляются с помощью объявлений `attributes`, а не `element`. Как уже отмечалось раньше, объявления атрибутов содержатся в определениях сложных типов в схемах. Вы уже видели на примерах объявлений элементов, что объявления являются не *типами*, а связями между именем и накладываемыми на него ограничениями в документе XML. То же самое справедливо и по отношению к объявлениям атрибутов.

В качестве примера листинг 6.8 приведен документ XML, который содержит как элементы, так и атрибуты. В частности, элемент `message` (строки 5–7) содержит текстовые данные и атрибут `number`.

Листинг 6.8. Элементы и атрибуты `message05.xml`

```
1: <?xml version="1.0"?>
2: <note
3:  xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance"
4:  xsi:noNamespaceSchemaLocation="message05.xsd">
5:  <message number="10">
6:          Remember to buy milk on the way home from work
7:       </message>
8: </note>
```

Этому документу в схеме XSD должен соответствовать сложный тип. Схема XSD для проверки документа приведена в листинге 6.9. В этом листинге проиллюстрировано несколько дополнительных средств, которые мы еще не рассматривали.

Указание типов данных в схемах XSD состоит из нескольких этапов. Вы начинаете ограничивать типы элементов одной достаточно общей категорией, после чего сужаете эту категорию до определенных типов. Такой подход может показаться избыточным, однако он полезен при работе с большими схемами, содержащими большое количество типов данных. Первый, наиболее общий тип, указывается с помощью элемента `xsd:restriction` с атрибутом `base`, объявляющим широкую категорию. Обычно атрибут `base` задается равным `"xsd:string"`. Дочерние элементы элементов `xsd:restriction` содержат подробные определения типов данных. Все это показано в листинге 6.9.

Листинг 6.9. Схема XSD с проверкой атрибутов `message05.xsd`

```
1: <?xml version="1.0"?>
2: <xsd:schema xmlns:xsd="http://www.w3.org/2000/10/XMLSchema">
3:  <xsd:complexType name="messageType">
4:      <xsd:simpleContent>
5:          <xsd:restriction base="xsd:string">
6:              <xsd:attribute name="number" type="xsd:integer"
                use="required"/>
7:          </xsd:restriction>
8:      </xsd:simpleContent>
9:  </xsd:complexType>
10: <xsd:element name="note">
11:     <xsd:complexType>
12:         <xsd:sequence>
13:             <xsd:element name="message" type="messageType"/>
14:         </xsd:sequence>
15:     </xsd:complexType>
16: </xsd:element>
17: </xsd:schema>
```

АНАЛИЗ В строках 3–9 определяется сложный тип `messageType`, который гарантирует, что элемент `message` содержит обязательный атрибут `number`, который, в свою очередь, содержит данные типа `integer`.

Обратите внимание: элемент `simpleContent` (строка 4) и элемент `complexType` (строка 11) не содержат атрибут `name`. Вы можете указать эти атрибуты, если планируете повторно использовать определения содержания в других местах схемы. Например, элемент `complexType` в строке 3 называется `messageType`. Объявление элемента `message` (строка 13) содержит ссылку (`type="messageType"`) на объявление сложного типа (строки 3–9) с объявлением атрибута (строка 6).

Элемент `sequence` (строки 12–14) включен только для того, чтобы показать вам типичные расположение и синтаксис, используемые в экземплярах, содержащих несколько дочерних элементов, для чего необходимы ограничения последовательностей.

В строках 5–7 определяется база для типа данных. Элемент `restriction` используется для указания существующего типа (с помощью атрибута `base`) и диапазона значений.

Типы данных, представленные в табл. 6.1 , относятся к элементам и к атрибутам. По сравнению с языком XDR язык XSD поддерживает намного больше типов данных. Это связано с тем, что при разработке и утверждении стандарта консорциум *W3C* уделил этому вопросу немало внимания.

Поддержка типов данных языками XSD и XDR значительно отличает их от DTD, который такой поддержкой не обладает вовсе. Консорциум *W3C* приложил все возможные усилия, чтобы реализовать в XSD поддержку наиболее широкого набора типов данных. XSD разделяет все типы данных на две группы — базовые (primitive) и производные (derived). *Базовые* типы данных — это такие типы данных, которые не выражаются с помощью других типов данных. *Производные* типы данных — это такие типы данных, которые выражаются с помощью других базовых типов данных. Рассмотрим, например, такой основной тип данных, как `float`, а также производный тип данных `integer`. Тип данных `float` — это четко определенная математическая концепция, которую нельзя выразить в терминах других типов данных, в то время как `integer` — это разновидность более общего типа данных `decimal`. Таким образом, тип данных `integer` произведен от типа данных `decimal`

В листинге 6.10 приведена более сложная коллекция атрибутов и различных типов данных.

Листинг 6.10. Использование нескольких атрибутов `message06.xml`

```
1: <?xml version="1.0"?>
2: <note
3:  xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance"
4:  xsi:noNamespaceSchemaLocation="message06.xsd">
5:  <message number="10" date="2001-07-29" from="Kathy Shepherd">
6:        Remember to buy milk on the way home from work
7:     </message>
8: </note>
```

АНАЛИЗ В строке 5 показаны такие атрибуты, как `xsd:integer` (числовой), `xsd:date` (дата) и `xsd:string` (строка), экземпляра документа XML. Объявление этих атрибутов требует явного определения типов данных для ограничения их значений. Элементы `xsd:attribute` выглядят следующим образом:

```
<xsd:attribute name="number" type="xsd:integer" use="required"/>
<xsd:attribute name="date" type="xsd:date" use="required"/>
<xsd:attribute name="from" type="xsd:string" use="required"/>
```

Каждый элемент attribute в этом пример объявляется с помощью атрибутов name (соответствует имени элемента в документе XML), type (объявление типа данных в соответствии с описанными раньше ограничениями) и use (определение того, обязательный или нет атрибут). Если вы объявляете атрибуты как обязательные, вам необходимо указать атрибут use="required" для каждого элемента xsd:attribute.

Полный вариант схемы XSD для проверки документа приведен в листинге 6.11.

Листинг 6.11. Схема XSD для проверки экземпляра XML с несколькими атрибутами

message06.xsd

```
1: <?xml version="1.0"?>
2: <xsd:schema xmlns:xsd="http://www.w3.org/2000/10/XMLSchema">
3:  <xsd:complexType name="messageType">
4:      <xsd:simpleContent>
5:          <xsd:restriction base="xsd:string">
6:              <xsd:attribute name="number" type="xsd:integer"
                use="required"/>
7:              <xsd:attribute name="date" type="xsd:date"
                use="required"/>
8:              <xsd:attribute name="from" type="xsd:string"
                use="required"/>
9:          </xsd:restriction>
10:     </xsd:simpleContent>
11: </xsd:complexType>
12: <xsd:element name="note">
13:     <xsd:complexType>
14:         <xsd:sequence>
15:             <xsd:element name="message" type="messageType"/>
16:         </xsd:sequence>
17:     </xsd:complexType>
18: </xsd:element>
19: </xsd:schema>
```



В строках 5–9 объявляется ограничение типа данных для атрибута с помощью дескриптора <xsd:restriction base="xsd:string">. Каждый атрибут относится к типу string (строка 5), поэтому в дальнейшем он ограничивается объявлениями типов данных в строках 6–8 (integer, date, string), а также атрибутом use. (Каждый атрибут обязателен в проверяемом документе XML.)

Три подхода к проверке документов: DTD, XDR и XSD

Три последних дня вы изучали связанные, но несколько отличающиеся подходы, обеспечивающие проверку документов на соответствие синтаксическим правилами и стандартам, определяемым специальными ограничениями. Схемы DTD используются дольше всего и поддерживаются большим количеством различных приложений. Если вам необходимо строгое указания ограничений, то DTD, наименее гибкий из трех рассмотренных языков описания схем, — именно то, что вам необходимо. Однако в схемах DTD не используется синтаксис документов XML, что является важным преимуществом других языков. Хотя использование синтаксиса XML в схеме необязательно приводит к улучшению процесса проверки, оно приводит к дополнительной гибкости, а также позволяет преобразования к другим языкам разметки, чем не может похвастаться DTD. Схемы, базирующиеся на словарях XML, могут содержать более сложную и выразительную разметку, а также сложные ограничения содержания.

Возможно, наиболее важным улучшением, связанным со схемами на базе XML, является возможность проверки типов данных. Как вы узнали из настоящего, а также предыдущего уроков, языки XSD и XDR поддерживают достаточно широкий диапазон типов данных. XSD поддерживает даже больше тппов данных, чем XDR.

Схемы на основе XML обеспечивают лучшую поддержку пространств имен. Подробно с пространствами имен вы познакомитесь в уроке 8, однако на данный момент самым важным является то, что концепция пространства имен относится к ключевым положениям, поддерживаемым консорциумом *W3C*.

На XSD, последнюю рекомендацию консорциума W3C, возлагаются очень большие надежды как на универсальный плаформно-независимый язык описания схем, поддерживающий огромное количество типов данных. Подробные сведения о существующих разработках в данном направлении вы найдете по адресу `http://www.w3.org`.

Резюме

Сегодня вы познакомились с ключевыми конструкциями языка XSD, поддерживающего большое количество типов данных. Теперь в вашем арсенале XML-инструментов целых три самых популярных подхода к проверке экземпляров документов XML. Последние сведения о текущих разработках вы найдете на Web-узле `http://www.w3.org`.

Вопросы и ответы

Какой из трех представленных в настоящей книге языков описания схем (DTD, XDR и XSD) используется шире других?

На данный момент существует очень много наследуемых решений, поддерживающих язык DTD; однако очень быстро распространение получает и язык XDR, преимущественно в сфере электронной коммерции, электронного бизнеса и т.д. Компания *Microsoft* разработала целый ряд решений на базе ключевых XML-технологий с использованием языка XDR для проверки документов. Несмотря на это, индустрия ожидает поддержки одного общепринятого стандарта. Хотя должно пройти достаточно много времени, прежде чем наследуемые системы обзаведутся поддержкой XSD, многие компании, обладая собственными разработками в данном направлении, уже высказали намерение реализовать в своих продуктах полную поддержку языка XSD.

В чем разница между схемами DTD и схемами на основе XML по отношению к типам данных?

Схемы DTD не обеспечивают проверку типов данных, за исключением текстовых. Схемы на базе XML поддерживают большое количество типов данных в дополнение к стандартным типам данных. Проверка типов данных делает эти языки наиболее подходящими для использования в приложениях, интенсивно работающих с данными.

Упражнение

Проверьте свои знания, полученные в результате изучения материала настоящего урока. Ответ вы найдете в Приложении А, "Ответы на вопросы".

Вчера вы создавали схему XDR для языка разметки MCML. Используя примеры, приведенные в настоящем уроке, создайте схему XSD для проверки файлов MCML.

День 7-й

Ссылки XML

Теперь, когда вы познакомились с документами XML, соответствующими основным синтаксическим правилам и заданными с помощью схем ограничений, пришло время познакомиться со специальным классом объектов XML — ссылками XML. На самом деле, схемы, которые вы создавали при изучении предыдущих уроков, являются специальными внешними ссылками, к которым обращается ваш документ XML.

Сегодня рассмотрим следующие темы.

- Внутренние и внешние общие ссылки в XML.
- Синтаксис ссылок и их использование.
- Параметрические ссылки и их использование для обеспечения модульности схем DTD.

Два вида ссылок

В уроке дня 4-го вы уже сталкивались с несколькими примерами ссылок в схемах DTD, но при этом не рассматривались различные типы ссылок, поддерживаемые в XML. В настоящем уроке вы познакомитесь с различными типами ссылок XML и научитесь использовать их для решения определенных проблем при создании документов, а также для обеспечения модульности в процессе программирования.

В XML существует два вида ссылок для инкапсуляции данных. Заранее определенные общие ссылки уже знакомы тем, кто создает HTML-страницы и сталкивался с отображением символов, которые совпадают с зарезервированными знаками разметки. В документе HTML, если нужно отобразить такие знаки, как <, > или &, вы должны определить их как ссылку.

Подобные ссылки в HTML представляются следующим образом:

```
&ИМЯ_ССЫЛКИ;
```

Когда броузер сталкивается с подобной ссылкой при интерпретации страницы, говоря о подстановке. Ссылка заменяется тем символом, который она представляет. Это же справедливо и по отношению к XML. Подстановка выполняется при каждой обработке экземпляра XML. Однако в данном случае ссылку заменяет нужным значением не броузер, а синтаксический анализатор.

Параметрические ссылки определяются и используются только во внешних схемах, таких как DTD, однако и другие языки описания схем предоставляют возможности для объявления ссылок. Параметрические ссылки позволяют сократить количество введен-

ных разработчиком данных. Кроме того, они позволяют повторно использовать внешние схемы DTD, обеспечивая "модульный" стиль работы. В настоящем уроке вы познакомитесь с двумя видами ссылок и научитесь использовать их в экземплярах XML.

Общие ссылки

Общие ссылки в HTML позволяют указывать символы, которые выходят за рамки стандартного набора символов ASCII или могут конфликтовать со знаками разметки. XML позволяет свободнее использовать нестандартные символы, например, выходящие за рамки стандартного набора символов ASCII. Это связано с тем, что XML проектировался таким образом, чтобы использовать преимущества набора символов Unicode, а не ASCII. 16-разрядный набор символов Unicode содержит 65536 символов, в то время как 7-разрядный ASCII — всего 128, а 8-разрядный — 256. При использовании набора символов ASCII, как нетрудно догадаться, количества символов явно недостаточно. До появления набора символов Unicode некоторые языки в HTML-документах представлялись в графическом виде, поскольку набор символов ASCII не содержал всех необходимых символов. Несколько европейских языков очень быстро заняли все 256 мест, доступных в ASCII. Азиатские языки содержат много символов и наборов знаков, образующих просто огромное количество идеограмм. Для решения подобных проблем и был разработан 16-разрядный набор символов Unicode. Однако, если и возможностей этого набора символов окажется недостаточно, в XML реализована поддержка и 32-разрядногой набора символов Unicode.

XML содержит пять встроенных ссылок, позволяющих разработчику избежать проблем с разметкой документа. Все они перечислены в табл. 7.1.

Таблица 7.1. Пять стандартных ссылок XML

Ссылка	Описание
`&`	Знак амперсанда (&)
`'`	Апостроф (`)
`>`	Больше (>)
`<`	Меньше (<)
`&quote;`	Кавычка (")

Помимо использования стандартного набора ссылок, поддерживаемых XML, вы можете создавать и свои собственные точно так же, как вы объявляете элементы. Ссылки позволяют сэкономить время, поскольку повторяющиеся задачи легко упростить благодаря определению ссылок для подстановки данных во время интерпретации документа обработчиком XML. Синтаксис объявления общей ссылки в схеме DTD приведен ниже:

```
<!ENTITY имя_ссылки данные_для_подстановки>
```

Синтаксис использования ссылки в экземпляре XML приведен ниже.

```
&ИМЯ_ССЫЛКИ;
```

Ссылки нельзя так просто объявить при использовании других языков описания схем. В настоящем уроке при объявлении ссылок вы будете использовать DTD. Обратите внимание на то, что все ссылки содержат знак амперсанда, после которого указано имя ссылки, а затем следует точка с запятой. В момент обработки синтаксическим анализатором XML ссылка заменяется соответствующими данными.

Использование этого механизма для создания ссылок очень полезно в случае экземпляров XML, содержащих повторяющиеся данные, изменяющиеся во "время жизни" документа. Например, официальный контракт можно разметить с использованием ссылок для описания различных частей соглашения. Подстановка реальных данных будет осуществляться во время обработки данных. В листинге 7.1 приведен пример соглашения, представляющего собой правильно оформленный документ XML, в котором используются ссылки для частей соглашения. Используя текстовый редактор, создайте этот документ и сохраните его под именем `contract01.xml`.

Листинг 7.1. Документ XML, содержащий ссылки, `contract01.xml`

```
1: <?xml version="1.0"?>
2: <!-- листинг 7.1 - contract01.xml -->
3:
4: <!DOCTYPE contract SYSTEM "contract01.dtd">
5: <contract>
6:  <para1>&buyer; agrees to buy the property defined below
7: on the terms and conditions set forth in this contract.</para1>
8:  <para2 section="1">&buyer; will take title of the real property
9: described below, for which consideration &buyer; agrees
10: to pay... </para2>
11: </contract>
```



Корневой элемент `contract` (строки 2–11) содержит элементы `para1` (строки 6 и 7) и `para2` (строки 8–10). Элемент `para2` содержит атрибут `section` со значением, равным 1. В строке 4 содержится объявление `DOCTYPE`, которое идентифицирует элемент `contract` как корневой, а также объявляет внешнюю схему DTD `contract01.dtd`. В строках 6, 8 и 9 содержится ссылка `&buyer;`.

Схема DTD для этого документа необходима не только для того, чтобы объявить структуру документа и отношения межу атрибутами, но и для обеспечения элементов, которые будут использоваться для подстановки вместо ссылок в документе XML после обработки последнего с помощью синтаксического анализатора. Предположим, покупатель указан как `Devan Shephard`. В схеме DTD соответствующая ссылка будет иметь следующий вид:

```
<!ENTITY buyer "Devan Shephard">
```

Этот фрагмент схемы DTD указывает обработчику XML подставить строку `Devan Shephard` вместо каждой ссылки `buyer` в экземпляре XML. Обработчик сможет найти ссылку по соответствующему синтаксису: `&buyer;`. В листинге 7.2 приведена полная схема для экземпляра документа `contract01.xml`. Создайте соответствующий документ и сохраните его в файле `contract01.dtd` в той же папке, что и файл `contract01.xml`.

Листинг 7.2. Схема DTD, содержащая объявления ссылок, `contract01.dtd`

```
1: <?xml version="1.0" encoding="UTF-8"?>
2: <!-- Листинг 7.2 - contract01.dtd -->
3:
4: <!ELEMENT contract (para1, para2)>
5: <!ELEMENT para1 (#PCDATA)>
6: <!ELEMENT para2 (#PCDATA)>
7: <!ATTLIST para2
8:  section CDATA #REQUIRED
9: >
10: <!ENTITY buyer "Devan Shepherd">
```

АНАЛИЗ В строках 4–9 объявляется структура элементов и атрибутов экземпляра документа XML. Это означает, что элемент `contract` содержит другие элементы, в частности дочерние элементы `para1` и `para2`. Элемент `para1` содержит только текст (`#PCDATA`) точно так же, как и элемент `para2`. Элемент `para2` также содержит обязательный атрибут `section`, который содержит только символьные данные (`CDATA`). В строке 10 определяется ссылка `buyer`, равная `"Devan Shepherd"`.

Для того чтобы увидеть результаты подстановки, загрузите документ `contract01.xml` в броузер Microsoft Internet Explorer. Синтаксический анализатор MSXML обработает ссылку и заменит ее данными, полученными из схемы DTD. Результат показан на рис. 7.1. Обратите внимание на то, что строка `"Devan Shepherd"` указана на месте ссылки `buyer`.

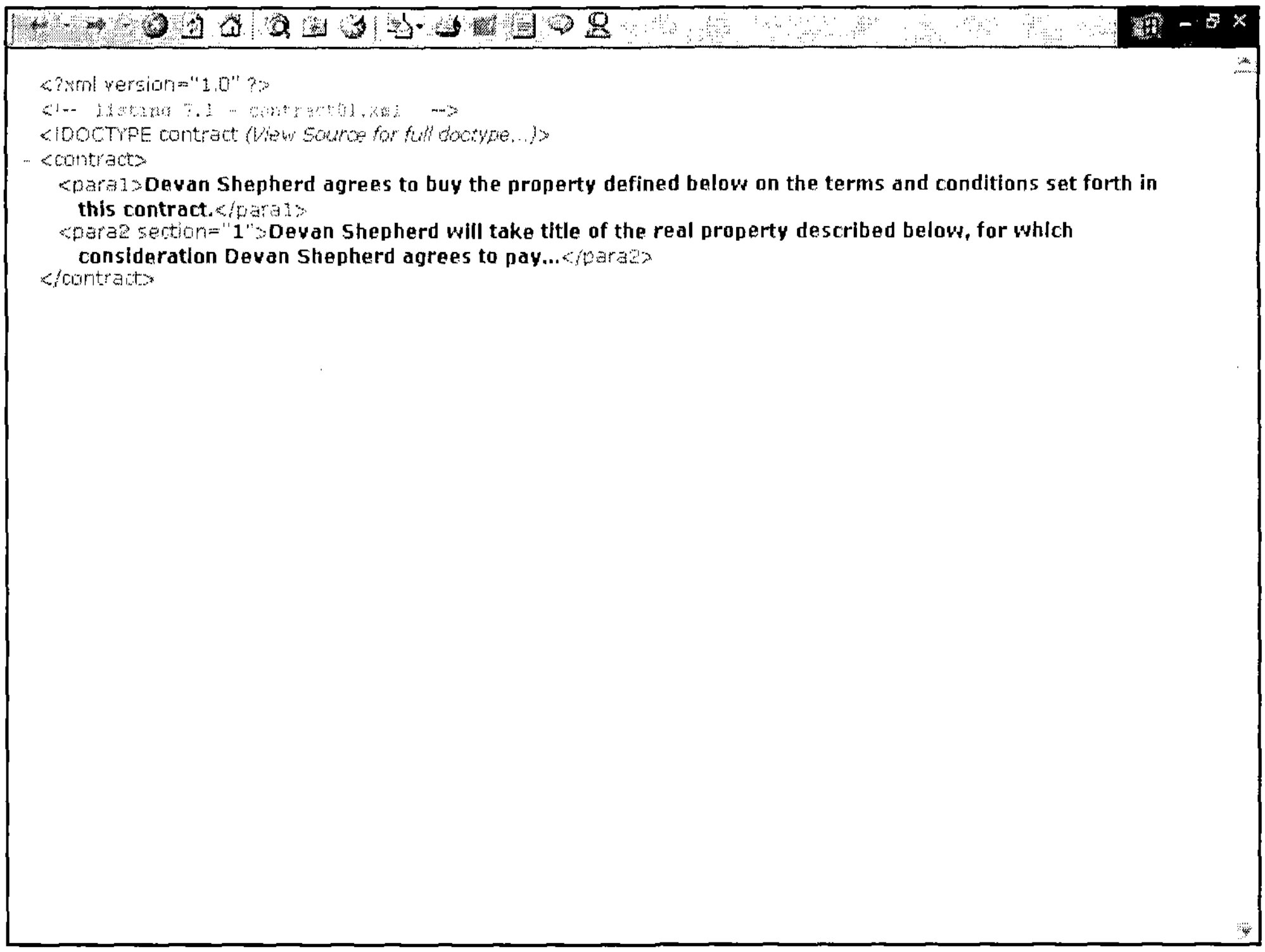


Рис. 7.1. Документ `contract01.xml` с обработанными ссылками

Представьте себе, что у вас есть большой многостраничный документ, размеченный подобным образом. Например, в случае контракта о продажах, ваш документ содержит несколько десятков страниц, на каждой из которых представлено по несколько ссылок `buyer` и т. д. Подстановка ссылок позволяет вам сэкономить массу времени и гарантирует, что в каждом случае имя покупателя будет указано без ошибок. И что еше важнее, когда вам следующий раз потребуется договор о продаже, вам необходимо всего лишь один раз изменить ссылку в схеме DTD, чтобы указать имя нового покупателя в соответствующих местах документа.

Такой подход позволяет вам очень легко повторно использовать документы, а также обеспечивает модульное представление данных. Даже на таком простом примере вы увидели, каким образом целые классы элементов представляются в документах с помощью ссылок.

Размер ссылок не ограничен, таким образом их можно использовать и при представлении резко повторяющихся, но длинных текстовых строк.

Параметрические ссылки

Параметрические ссылки используются в схемах DTD для предоставления информации, используемой в объявлениях разметки. Как правило, параметрические ссылки используются для определения набора общих атрибутов для ряда элементов, а также для предоставления связи с внешней схемой DTD. Целевые ссылки в схемах DTD называются внутренними параметрическими ссылками, а ссылки, указывающие на данные и объекты за пределами текущей схемы DTD, называются внешними параметрическими ссылками.

Параметрические ссылки полезны во многих ситуациях, но в них достаточно сложно разобраться. Параметрические ссылки в схемах DTD указываются с помощью знака процента (%), что позволяет отличить параметрическую ссылку от обычной. Синтаксис параметрической ссылки в схеме DTD следующий:

```
<!ENTITY %имя_ссылки определение_ссылки>
```

Знак процента должен отделяться от имени ссылки пробелом. Определением ссылки может быть любая допустимая в DTD строка. Строка должна быть указана там, где ссылка используется в схеме DTD. Другими словами, внутренние параметрические ссылки функционируют точно так же, как и обычные, за исключением того, что подстановка осуществляется в самой схеме DTD, а не в документе XML.

Предположим, вам необходимо создать схему DTD для счета-фактуры, который должен содержать такие сведения, как адрес платежа, адрес доставки, адрес продавца, а также адрес хранилища. Если все адреса имеют общую структуру, вы можете использовать параметрическую ссылку для создания структуры адреса, после чего использовать ее несколько раз в схеме DTD. Например, структура адреса в схеме DTD может выглядеть следующим образом.

```
<!ENTITY        % address
                "street         CDATA           #REQUIRED
                street2         CDATA           #IMPLIED
                city            CDATA           #REQUIRED
                State           CDATA           #REQUIRED
                zip             CDATA           #REQUIRED">
```

В данном случае ссылка address указывает на набор объявлений атрибутов. Знак процента указывает на то, что после ключевого слово ENTITY следует параметрическая ссылка. Параметрическая ссылка называется address и ссылается на целый ряд атрибутов, указанных в двойных кавычках (").

После этого вы можете использовать ссылку address для подстановки подобного набора атрибутов для элементов, ограничения для которых записаны в схеме DTD. Например, для проверки атрибутов адреса платежа в элементе bill вам необходимо указать следующее:

```
<!ELEMENT bill (#PCDATA)>
<!ATTLIST bill %address;>
```

Элемент bill содержит текстовые данные (#PCDATA) и несколько атрибутов. Атрибуты указаны с помощью ссылки %address, объявленной раньше в схеме DTD. Знак процента используется для указания на начало имени ссылки, однако между этим знаком и именем ссылки нет пробела, что указывает синтаксическому анализатору на то, что тот имеет дело с ссылкой. Пробел между знаком процента и именем ссылки указывается только, если необходимо отделить ее от других ссылок.

Когда синтаксический анализатор XML сталкивается с ссылкой в схеме DTD, результат будет таким же, как в случае схемы DTD.

```
<!ELEMENT   bill                        (#PCDATA)>
<!ATTLIST   bill
            street      CDATA       #REQUIRED
            street2     CDATA       #IMPLIED
            city        CDATA       #REQUIRED
            State       CDATA       #REQUIRED
            zip         CDATA       #REQUIRED
```

Если мы вернемся к нашему примеру счета-фактуры, более полная схема DTD будет выглядеть следующим образом:

```
<!ELEMENT   invoice                     (bill, ship, vendor, warehouse)>
<!ELEMENT   bill                        (#PCDATA)>
<!ATTLIST   bill                        %address;>
<!ELEMENT   ship                        (#PCDATA)>
<!ATTLIST   ship                        %address;>
<!ELEMENT   vendor                      (#PCDATA)>
<!ATTLIST   vendor                      %address;>
<!ELEMENT   warehouse                   (#PCDATA)>
<!ATTLIST   warehouse                   %address;>
<!ENTITY    % address
            "street     CDATA       #REQUIRED
            street2     CDATA       #IMPLIED
            city        CDATA       #REQUIRED
            State       CDATA       #REQUIRED
            zip         CDATA       #REQUIRED">
```

Вы сможете сэкономить массу времени на указании параметров каждого из элементов `bill`, `ship`, `vendor`, `warehouse`.

Внешние параметрические ссылки функционируют точно так же, за исключением того, что ссылки находятся в отдельном документе. Это означает, что в принципе схему DTD можно считать специальной формой внешней параметрической ссылки.

Ссылки в других языках описания схем

В настоящем уроке вы уже познакомились с большим количеством примеров ссылок, но все они касались объявлений в схемах DTD. Объявления ссылок в других языках разметок осуществляются далеко не так легко. Например, для объявления ссылки в схеме XSD, простейший способ состоит в создании специального элемента, после чего необходимо зафиксировать его содержание. После этого вы сможете использовать элемент в качестве ссылки. Кроме того, вам необходимо объявить отдельное пространство имен, а также убедиться в том, что созданный вами элемент связан именно с этим пространством имен.

Например, представьте себе, что вам необходимо создать эквивалент символьной ссылки, используя описанный подход, для представления строки подстановки. Возможно, вам необходима короткая форма представления настоящей книги, *Teach Yourself XML in 21 Days*. В схеме XSD вы можете создать элемент и указать его фиксированное значение. Соответствующее объявление XSD будет выглядеть следующим образом:

```
<xsd:element name="tyx21"> fixed="Teach Yourself XML in 21 Days, Second Edition">
```

Экземпляр документа XML, в котором используется этот элемент, может содержать его как относящийся к указанному пространству имен, после чего, уже после обработки с помощью синтаксического анализатора, можно указать соответствующее значение элемента. Ниже приведен фрагмент документа XML, в котором представлен элемент с фиксированным содержанием:

```
<?xml version="1.0"?>
<message xmlns="mystuff/message"
         xmlns:ent="mystuff/message/entities">

         ...другая разметка

        <Introduction>Welcome to <ent:tyx21>. This book is about...
        </Inrroduction>

         ...другая разметка

</message>
```

После обработки синтаксическим анализатором элемент `<ent:tyx21>` будет заменен строкой *Teach Yourself XML in 21 Days.*

Резюме

Сегодня вы познакомились с использованием в документе XML ссылок в качестве объектов, позволяющих реализовать подстановку данных при обработке документов с помощью синтаксического анализатора XML. В некоторых случаях подобные подстановки позволяют избежать различных проблем с разметкой. Например, с помощью ссылок можно включать в документы такие знаки, как < или &, и при этом не будет возникать никаких проблем при обработке этих документов синтаксическим анализатором XML. XML позволяет вам создавать собственные ссылки для сокращения времени при выполнении повторяющихся задач, обеспечивая подстановку регулярно изменяющихся данных, а также позволяя повторно использовать определенные модули программного кода.

Вопросы и ответы

Какие два вида ссылок XML существует и чем они отличаются между собой?

XML поддерживает стандартные и параметрические ссылки. Стандартные ссылки могут быть внутренними или внешними и позволяют реализовать подстановку данных. Параметрические ссылки существуют в схемах DTD, а не в самих экземплярах документов. Параметрические ссылки обеспечивают возможность подстановки фрагментов кода.

Каким образом синтаксический анализатор различает стандартную и параметрическую ссылку в схеме DTD?

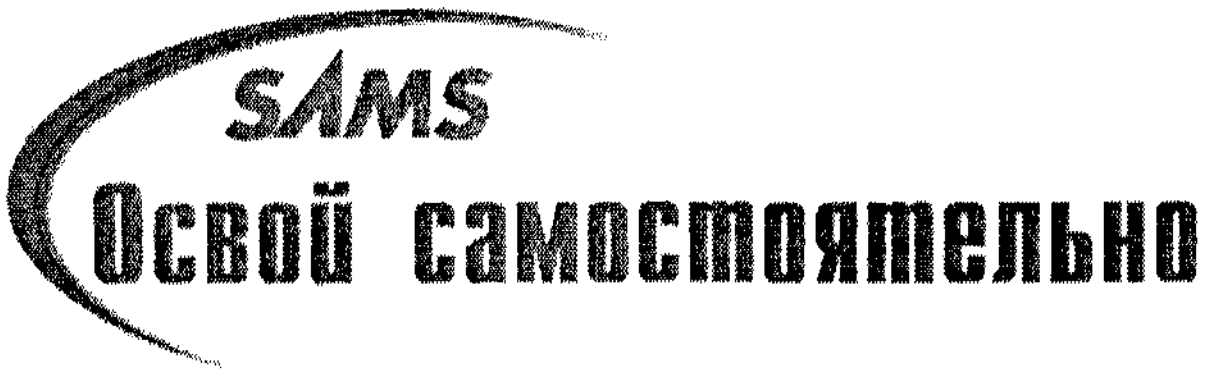
Параметрическую ссылку можно определить по знаку процента, указанному перед именем ссылки и отделенному от нее пробелом.

Упражнение

Проверьте свои знания, полученные в результате изучения материала настоящего урока. Ответ вы найдете в Приложении А, Ответы на вопросы".

Используйте экземпляр MCML, который вы создали еще при выполнении упражнения из урока 4, и измените его, добавив дочерний элемент `style` в каждый из элементов `cd`. Создайте ссылку `&style;` с подходящим текстовым описанием. Завершив эти действия, загрузите документ `cd.xml` в броузер, чтобы удостовериться в том, что все ссылки разрешены и обрабатываются должным образом.

Неделя II



Обработка XML

8. Пространства имен XML
9. Язык XPath
10. Язык XLink
11. Язык XPointer
12. Объектная модель документа XML — DOM
13. Simple API для XML
14. Использование каскадных таблиц стилей CSS

День 8-й

Пространства имен XML

Вы уже использовали пространства имен при изучении языков описания схем XDR и XSD и должны помнить, что обработчик XML использует пространства имен во время проверки документа для идентификации схемы XDR или XSD. Сегодня рассматриваются следующие темы.

- Что такое пространства имен и чем они полезны при программировании на XML.
- Синтаксис пространств имен XML.
- Различие между URI, URL и URN.
- Как явным образом объявить пространства имен для элементов и атрибутов в XML?
- Назначение пространства имен по умолчанию и его объявление.
- Стандартные пространства имен, используемые разными XML-технологиями.

Зачем необходимы пространства имен

Модель данных, характеризуемая разметкой XML, обычно представляет собой дерево элементов. Древовидная структура помогает выявить иерархическую природу документов XML. Вы создавали и исследовали древовидные диаграммы простых документов XML в уроке дня 2-го при изучении синтаксических правил XML.



Подробное рассмотрение древовидной структуры документов XML вы найдете в уроке дня 12-го, в котором рассматривается доступ к документам XML с использованием прикладного интерфейса API объектной модели документов DOM (Document Object Model). Прикладной интерфейс DOM API позволяет вам реализовать программный доступ к *узлам* — элементам, атрибутам и примечаниям, образующим дерево документа XML.

В разметке XML каждый элемент содержит имя типа или дескриптора, но может содержать и атрибуты. Как вы уже знаете, атрибуты определенного элемента указываются в его открывающем дескрипторе. Каждый атрибут состоит из имени и соответствующего значения. Консорциум *W3C* называет комбинацию имен типов элементов и атрибутом словарем разметки (markup vocabulary) (`http://www.w3.org/TR/REC-xml-names/`).

Приложения, созданные для использования данных XML, обычно используют словарь разметки для определения того, как обрабатывать тот или иной элемент. Элементы с разными именами обрабатываются по-разному, а элементы с одинаковыми именами — оди-

наково. Например, элементы `message` обрабатываются не так, как элементы `response`, но все элементы `message` в одном экземпляре XML обрабатываются совершенно одинаково.

Подобная обработка соглашений об использовании имен всегда сработает так как надо, если вы, как автор документа XML, обеспечите, чтобы все уникальные элементы и атрибуты обладали и уникальными именами. Однако созданные авторами документов имена потенциально могут приводить к проблемам в распределенных средах, таких как Web, в которых должен существовать общий подход к обработке документов XML. Например, многие транзакции электронной коммерции содержат слияние данных из разных источников. При совпадении имен элементов или атрибутов обработка данных усложняется, а в некоторых ситуациях становится вообще невозможной. Например, если два деловых партнера обмениваются данными XML, используя при этом элемент `message` для представления разных типов (или классов) данных, приложение, обрабатывающее документы, просто не сможет различить их. Если в одном документе XML под одним именем подразумеваются элементы различных типов, мы имеем дело с конфликтом имен.

Вы уже знаете, что XML позволяет вам размещать структурированные данные в текстовых файлах, а сама природа подобных структур "беспокоится" о должной разметке данных.

Как выглядит конфликт имен

Вы уже видели несколько примеров сложной разметки документов на языке XML, который позволяет вам создавать собственные дескрипторы. Конечно, люди, с которыми вы обмениваетесь данными, также могут создавать собственные дескрипторы, как уже отмечалось в примере из мира электронной коммерции раньше в настоящем уроке. Поэтому возможна такая ситуация, что вы и ваш партнер создадите различные элементы с одинаковыми именами; причем вероятность подобной ситуации на самом деле достаточно высока. В данном случае возникает *конфликт имен*, а приложение не различает элементы с одинаковыми именами.

Возможно, вы работаете над разметкой документа о структуре комитета, содержащей элемент `chair` для идентификации председателя комитета. В документе содержится список членов комитета. Думается, документ может выглядеть приблизительно так, как показано в листинге 8.1.

Листинг 8.1. Простой документ XML с описанием структуры комитета `committee.xml`

```
1: <?xml version="1.0"?>
2: <!-- Листинг 8.1 - committee.xml -->
3:
4: <committee>
5:  <chair>Kathy</chair>
6:  <member>Merrenna</member>
7:  <member>Doug</member>
8:  <member>Greg</member>
9:  <member>Kristen</member>
10: </committee>
```

Обратите внимание на то, что в этом экземпляре XML содержится простой список имен, одно из которых соответствует председателю комитета (элемент `chair`), а остальные — обычным членам комитета (элементы `member`).

В дальнейшем выполняется слияние документ с данными, предоставленными ответственным за проведение собрания, который определяет, кто именно будет сидеть за столом во время следующего рассмотрения насущных проблем, стоящих перед комитетом. Кроме того, в разметке, созданной ответственным за проведение собрания, есть элемент `chair`, который определяет место каждого из членов комитета за столом. Соответствующий документ приведен в листинге 8.2.

Листинг 8.2. Простой документ XML с описанием плана посадочных мест `seating.xml`

```
1:  <?xml version="1.0"?>
2:  <!-- Листинг 8.2 - seating.xml -->
3:
4:  <seating>
5:   <chair>head of table</chair>
6:   <chair>first on the left</chair>
7:   <chair>second on the left</chair>
8:   <chair>first on the right</chair>
9:   <chair>second on the right</chair>
10:  </seating>
```

Результат слияния этих документов приведет к недоразумениям, поскольку в каждом из документов есть имя `chair`, соответствующее различным элементам. Полученный в результате объединения документ приведен в листинге 8.3.

Листинг 8.3. Простой документ XML с описанием плана собрания `meeting.xml`

```
1:  <?xml version="1.0"?>
2:  <!-- Листинг 8.3 - meeting.xml -->
3:
4:  <meeting>
5:   <chair>Kathy</chair>
6:       <chair>head of table</chair>
7:   <member>Merrenna</member>
8:       <chair>first on the left</chair>
9:   <member>Doug</member>
10:      <chair>second on the 1eft</chair>
11:  <member>Greg</member>
12:      <chair>first on the right</chair>
13:  <member>Kristen</member>
14:      <chair>second on the right</chair>
15:  </meeting>
```

Конфликт имен `chair` приводит к тому, что приложение не сможет должным образом их обработать. Приложение не сможет различить элемент `chair` в строке 5, полученный из разметки структуры комитета, а также элементы `chair` из плана "посадки" в строках 6, 8, 10, 12 и 14. С точки зрения приложения, документ содержит один класс данных, к которому относятся все элементы `chair`. Чаще всего люди смогут без проблем различить различные элементы `chair` без какой-либо дополнительной информации. Таким образом, проблема будет оставаться до тех пор, пока вы не "проинформируете" приложение о существовании двух разновидностей элементов `chair` и способе обработки каждого из них.

Пространства имен предоставляют вам механизм уникальной идентификации элементов с одинаковыми именами, принадлежащих различным классам данных. Очень скоро вы увидите, как именно действует этот механизм; однако сначала мы рассмотрим еще один пример документа XML с коллизией имен.

Предположим, ваш документ XML описывает коллекцию элементов в офисе. В данном случае речь идет о книгах или о произведениях искусства. Таким образом, в документе элемент `item` будет соответствовать двум классам элементов — *книгам* и *произведениям искусства*.

Каждый элемент `item` в элементе `collection` содержит элемент `description`, `title`, а также элемент, идентифицирующий создателя элемента, такой как `author` для книги или `artist` для произведения искусства. Пример разметки коллекции на XML приведен в листинге 8.4. Если предположить, что элемент `item` приводит к коллизии имен, сможете ли вы определить еще подобные элементы?

Листинг 8.4. Документ XML с коллизией имен `collection.xml`

```
1: <?xml version="1.0"?>
2: <!-- Листинг 8.4 - collection.xml -->
3:
4: <collection>
5:
6:  <item number="1">
7:      <description>an XML book</description>
8:      <title>SAMS Teach Yourself XML in 21 Days, Second Edi
       tion</title>
9:      <author>Devan Shepherd</author>
10: </item>
11:
12: <item number="1">
13:     <description>a great painting</description>
14:     <title>Bermuda Longtails</title>
15:     <artist>E. Anthony</artist>
16: </item>
17:
18: </collection>
```

Потенциально к конфликту имен в этом примере "причастен" элемент `item`, который относится к таким типам данных, как книги и произведения искусства. Другими словами, элемент `item`, соответствующий книге, и элемент `item`, соответствующий произведению искусства, — это разные вещи. Элемент `description` для книги отличается от элемента `description` для произведения искусства. Элемент `title` — также еще один источник потенциальных проблем. Все эти элементы приводят к проблемам, поскольку приложение, использующее данные, не сможет различить такие элементы, как `item`, `description` и `title`. Дальше в настоящей главе вы узнаете, каким образом можно разрешить конфликт благодаря объявлению пространств имен.

Предотвращение конфликтов имен с помощью объявления пространств имен

Пространство имен XML выступает в качестве идентификатора, который уникальным образом определяет коллекцию имен, которые могут использоваться в качестве имен элементов или атрибутов в документе XML. Определив пространство имен для класса данных, вы информируете приложение XML о том, что данный элемент или атрибут *принадлежит* указанному пространству имен, или набору имен характеризующему определенный класс данных.

Пространство имен позволяет эффективно определять уникальные имена типов элементов и атрибутов в Web, а также в любой другой распределенной системе; так можно избежать коллизий имен элементов. В рассмотренном раньше примере планирования собрания вы можете создать пространство имен `committee` для элемента `chair`, соответствующего председателю комитета, пространство имен `seating` — для элемента `chair`, соответствующего посадочным местам. Все эти пространства имен вы создадите дальше в настоящей главе, но сначала вам следует познакомиться с синтаксисом и составными частями объявления пространства имен.

Синтаксис объявления пространства имен XML

Вы уже увидели несколько примеров объявлений пространств имен XML в предыдущих уроках. Вы можете сравнить эти пространства имен с теми, которые создадите сегодня с использованием стандартного синтаксиса объявлений пространства имен XML. Объявление пространства имен начинается с зарезервированного ключевого слова `xmlns`, после которого может следовать необязательное двоеточие. Двоеточие выступает в качестве префикса для имен элементов и атрибутов, связанных с этим пространством имен. К двум наиболее распространенным формам пространства имен относится простая форма, показанная ниже:

```
xmlns="URI"
```

Ниже приведен пример простого объявления пространства имен с использованием воображаемого идентификатора URI:

```
xmln="http://shepherdnamespace.com/XML"
```

Если вы хотите связать несколько элементов в экземпляре XML с конкретным пространством имен, имеет смысл использовать объявление, в котором используется префикс для этого пространства имен, как показано ниже:

```
xmlns:prefix="URI"
```

Ошибки при объявлении пространств имен

Ниже приведен пример объявления пространства имен с использованием определенного пользователем префикса, включенного в воображаемый идентификатор URI:

```
xmln:committee="http://shepherdnamespace.com/XML/ committee"
```

Пространства имен можно объявить как пространства имен по умолчанию с помощью первого варианта объявления — `xmlns="URI"`. Пространство имен по умолчанию относится ко всему экземпляру документа XML. Поскольку оно используется для объявления элементов в своей области, префикс не нужен. Пространство имен по умолчанию охватывает все элементы и атрибуты, которые не включены в явно определенное пространство имен.

В случае явного определения пространства имен (`xmlns:prefix="URI"`) вы объявляете сокращение, или префикс, для подстановки полного имени пространства имен в экземпляре документа XML. Вы используете этот префикс для определения всех имен элементов или атрибутов, принадлежащих определенному пространству имен, а не стандартному набору, определяемому пространством имен по умолчанию.

Как уже упоминалось в начале настоящего урока, документы XML образуют иерархические древовидные структуры, состоящие из различных узлов. Явные объявления полезны, если определенный узел документа содержит элементы или атрибуты из различных пространств имен.

Синтаксис элементов или атрибутов в экземпляре документа, принадлежащих определенному пространству имен, включает префикс, после которого следует двоеточие, а затем — имя элемента или атрибута, как показано ниже:

```
prefix:element
prefix:attribute
```

Пример:

```
<committee:chair>Kathy</committee:chair>
<filename committee:листинг="8.1">committee.xml</filename>
```

В приведенном примере элемент chair и атрибут листинг содержат префикс code, указывающий на то, что они принадлежат пространству имен xmln:committee= "http://shepherdnamespace.com/XML/ committee". Не забывайте, что это идентификатор URI, хотя и выглядит как Web-адрес, это просто метка, указывающая на коллекцию элементов и атрибутов. Приложение не будет даже пытаться подключиться к Web-узлу http://shepherdnamespace.com.

Префикс избавляет автора документа от необходимости явно объявлять пространство имен для каждого принадлежащего ему элемента или атрибута. Полностью определенный элемент или атрибут обрабатывается синтаксическим анализатором XML, который выполняет подстановку полных объявлений пространств имен, как показано ниже:

```
<name xmln:committee="http://shepherdnamespace.com/XML/ committee">A
Simple XML Document Depicting a Committee Structure</name>
<filename committee:листинг="8.1"
xmln:code="http://shepherdnamespace.com/XML/ committee".
>committee.xml</filename>
```

Уникальная идентификация пространств имен XML

Идентификатор URI пространства имен XML выступает в роли *уникальной метки* для коллекции элементов и атрибутов, которые он идентифицирует. Вы должны понимать, что идентификатор URI — это просто последовательность символов, используемая для различения имен.



> Идентификаторы URI используются в XML в качестве меток, которые уникальным образом идентифицируют пространства имен. Совершенно не играет роли то, на что именно (если это так) указывает идентификатор URI. Идентификатор URI всего лишь выступает в качестве "прозвища" имен элементов и атрибутов, которые он характеризует.

Некоторые читатели могут запутаться в формах представления идентификаторов URI, используемых в некоторых популярных XML-технологиях. Например, некоторые идентификаторы URI имеют вид адресов URL. Как вы помните из урока дня 6-го, идентификатор URI, который указывает схему XSD для синтаксического анализатора XML, определяется в корневом элементе схемы и всегда имеет следующую форму:

```
xmlns:xsd="http://www.w3.org/2000/10/XMLSchema"
```

Значение атрибута xmlns:xsd явно имеет форму адреса URL. Попытайтесь ввести это значение в адресной строке броузера и посмотрите на полученный результат. Если вы будете просматривать Web-узел с помощью Internet Explorer 5.0 или более поздней версии, вы увидите преобразованное представление схемы XML, используемой для проверки схемы XSD. Другими словами, консорциум *W3C* разместил по указанному адресу документ XML, чтобы у людей, которые попытаются разрешить адрес URL, представляющий часть объявления пространства имен, не возникло недоразумений.

Считаете ли вы, что синтаксический анализатор XML разрешит этот адрес URL при проверке документа на соответствие ограничениям, указанным в схеме XSD? Другими словами, загрузит ли синтаксический анализатор XML схему, которую вы видели по адресу http://www.w3.org/2000/10/XMLSchema, при проверке документа? Ответом на оба варианта будет "Нет". Вы должны помнить, что пространство имен — это всего лишь метка. Идентификатор URI — это уникальная метка. Если идентификатор URI содержит адрес URL, синтаксический анализатор XML не будет разрешать Web-адрес при объявлении пространства имен.

Зачем же тогда использовать адрес URL для идентификации пространства имен? На самом деле, адрес URL может быть глобально уникальным во всей Web. Когда вы

регистрируете доменное имя или имя узла, вы добавляете его к 450 миллионам существующих доменных имен (на момент написания книги; источник — *Internic News*, Network Solutions, Inc., 2001). Служба регистрации доменных имен *Internic* на данный момент разрешает регистрацию имен, содержащих вплоть до 22 букв до точки в адресе URL. Допускается использование любых буквенно-цифровых символов английского языка, а также знака "минус" или косой, если только эти знаки не указаны первыми. В результате в адресах URL допускается использование 37 символов на 22 позициях, что соответствует $3{,}17*10^{34}$ возможным комбинациям.

Применив эти сведения к вашему собственному уникальному доменному имени, вы всегда сможете создать уникальную метку для пространства имен. Именно по этой причине консорциум *W3C* определил уникальное пространство имен для схем XSD, которое вы использовали в уроке дня 6-го.

Идентификатор URI также можно создать в форме имени *URN* (Uniform Resource Name — унифицированное имя информационного ресурса), а не адреса URL. В уроке дня 5-го вы познакомились с созданием схем с использованием языка описания схем XML Data Reduced (XDR), предложенным компанией *Microsoft*. Вы можете вспомнить, что вы объявляли два пространства имен в корневом элементе каждой схемы XSD. Это были пространства имен URN, а не URL; они не имели формы, используемой для представления обычных Web-адресов. По определению URN — это любой идентификатор URI, который не выглядит как адрес URL. Это кажется несколько запутанным, однако именно так можно отличить имена URN от адресов URL. В случае схем XDR указываемые вами пространства имен содержали следующие имена URN:

```
xmlns="urn:schemas-microsoft-com:xml-data"
xmlns="urn:schemas-microsoft-com:datatypes"
```

Первое объявление определяет пространство имен, которое относится ко всему документу XDR, т. е. является пространством имен по умолчанию. Подробно с пространствами имен по умолчанию вы познакомитесь в разделе "Использование объявлений пространств имен" в настоящем уроке. Как вы помните, второе пространство имен определяет префикс имен типов данных, используемого для идентификации атрибутов типов данных в ограничениях схемы XDR. Дальше в документ XDR вы могли включить атрибуты `dt:type`, задающие ограничения для типов данных элементов или атрибутов. Например, в приведенном ниже примере обязательный атрибут `purchase_date` для определенного элемента действителен только в том случае, если значение атрибута выражено в виде строки, соответствующей определенному раньше синтаксису. В данном случае к пространству имен типов данных относится только атрибут `type`. Элемент `AttributeType`, а также атрибуты `name` и `required` относятся к пространству имен по умолчанию — `urn:schemas-microsoft-com:xml-data`:

```
<AttributeType name="purchase_date" dt:type="date" required="yes"/>
```



> Рабочая группа IETF (Internet Engineering Task Force — проблемная группа проектирования Internet) работает над созданием заранее определенного синтаксиса имен URN, базирующегося на предложении Р. Моатса (R. Moats) из компании *AT&T*, сделанного в мае 1997 года. Синтаксис обеспечивает разрешение имен с использованием протоколов Internet. Если работа группы окажется успешной, то имена URN могут на самом деле стать схемами URI.

Автор документа может использовать практически любое имя пространства имен, за исключением зарезервированного имени `xml`, являющегося интеллектуальной собственностью консорциума *W3C*.

Каким образом приложения используют пространства имен

В рассмотренных в настоящем уроке примерах заранее определенные пространства имен для XDR и XSD предоставляют важные сведения приложениям XML. Использование этих пространств имен необходимо для обеспечения самой возможности проверки документов XML. Даже если синтаксический анализатор XML не разрешает адрес URL во время проверки выполнения ограничений XDR, все равно пространство имен должно указываться. На самом деле, при столкновении с пространством имен синтаксический анализатор использует правила XSD, встроенные в само приложение. Пространство имен — это метка, которая указывает приложению на необходимость обработки элементов и атрибутов, которые относятся к определенной коллекции. Правила обработки элементов и атрибутов встроены в приложение еще до обработки документа.

Использование объявлений пространств имен

Теперь, когда вы познакомились с синтаксисом и теорией объявления пространств имен, запустите текстовый редактор и создайте пространства имен для устранения коллизий имен, показанных раньше в настоящем уроке.

Начните с примера собрания комитета, объявив пространства имен для элементов из документа `committee.xml`, а также элементов из документа `seating.xml`. В случае элементов из документа `committee.xml` создайте пространство имен с префиксом `cmte`, показанное ниже:

```
http://shepherdnamespace.com/XML/committee
```

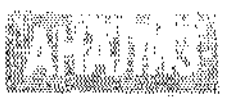
Не забывайте о том, что, несмотря на то, что этот идентификатор URI напоминает адрес URL, он необязательно указывает на реально существующий Web-узел. Идентификатор URI — это всего лишь метка, используемая синтаксическим анализатором XML. В случае элементов из документа `seating.xml` создайте пространство имен с префиксом `seat`, показанное ниже.

```
urn:XML_in_21_days:seating
```

Сохраните результат в файле `meeting02.xml`. Вы должны получить результат, приведенный в листинге 8.5.

Листинг 8.5. Простой документ XML с описанием плана собрания, содержащий объявления пространств имен, `meeting02.xml`

```
1: <?xml version="1.0"?>
2: <!-- Листинг 8.5 - meeting02.xml -->
3:
4: <meeting xmlns:cmte="http://shepherdnamespace.com/XML/committee"
5:                  xmlns:seat="urn:XML_in_21_Days:seating">
6:   <cmte:chair>Kathy</cmte:chair>
7:       <seat:chair>head of table</seat:chair>
8:   <cmte:member>Merrenna</cmte:member>
9:       <seat:chair>first on the left</seat:chair>
10:   <cmte:member>Doug</cmte:member>
11:       <seat:chair>second on the left</seat:chair>
12:   <cmte:member>Greg</cmte:member>
13:       <seat:chair>first on the right</seat:chair>
14:   <cmte:member>Kristen</cmte:member>
15:       <seat:chair>second on the right</seat:chair>
16: </meeting>
```



Указанные вами пространства имен объявляются как значения атрибутов `xmlns` корневого элемента `meeting`. Пространство имен `http://shepherdnamespace.com/XML/committee` содержит префикс `cmte`, который добавляется в открывающий и закрывающий дескрипторы каждого элемента, относящегося к этому пространству имен. Соответствующие элементы указаны в строках 6, 8, 10, 12 и 14.

Показанная ниже строка 5 содержит объявление пространства имен для элементов из документа `seating.xml`. Префикс `seat` добавляется к каждому из элементов `chair`, относящихся к пространству имен `seat`. Соответствующие элементы указаны в строках 7 9, 11, 13 и 15:

```
5:                    xmlns:seat="urn:XML_in_21_Days:seating">
```

Затем выполните те же действия для документа XML, приведенного в листинге 8.4. Вы должны помнить, что он содержит элементы двух классов, относящиеся к книгам и произведениям искусства. Используя текстовый редактор, создайте новый документ `collection02.xml`, содержащий пространство имен по умолчанию для книг с идентификатором URI `http://www.devan.org/books`, а также явное пространство имен с префиксом `art` и идентификатором URI `urn:mystuff:artwork` для произведений искусства. Полученный результат должен напоминать приведенный в листинге 8.6.

Листинг 8.6. Простой документ XML с описанием коллекции, содержащий объявления пространств имен, `collection02.xml`

```
1: <?xml version="1.0"?>
2: <!-- Листинг 8.6 - collection02.xml -->
3:
4: <collection xmlns="http://www.devan.org/books"
5:                    xmlns:art="urn:mystuff:artwork">
6:
7:    <item number="1">
8:        <description>an XML book</description>
9:        <title>SAMS Teach Yourself XML in 21 Days, Second Edi
          tion</title>
10:        <author>Devan Shepherd</author>
11:    </item>
12:
13:    <art:item number="1">
14:        <art:description>a great painting</art:description>
15:        <art:title>Bermuda Longtails</art:title>
16:        <art:artist>E. Anthony</art:artist>
17:    </art:item>
18:
19: </collection>
```



В строках 4–5 показан корневой элемент документа XML с пространством по умолчанию (стока 4), а также явное пространство имен с префиксом `art` (строка 5).

Поскольку первый элемент `item` (строки 7–11) и его дочерние элементы принадлежат пространству имен по умолчанию, в дескрипторах элемента никакой префикс не указывается. В то же время второй элемент `item` (строки 13–17) и его дочерние элементы принадлежат пространству имен `urn:mystuff:artwork`; таким образом, в открывающем и закрывающем дескрипторах элемента указан префикс `art`.

Избранные стандартные идентификаторы URI пространств имен XML

Вы уже познакомились с пространствами имен XDR и XSD, а также несколькими другими. В табл. 8.1 перечислено несколько широко используемых стандартных пространств имен XML.

Таблица 8.1. Избранные стандартные идентификаторы URI пространств имен XML

Приложение	*Идентификатор URI пространства имен*
Схема XDR	`urn:schemas-microsoft-com:xml-data`
Типы данных XDR	`urn:schemas-microsoft-com:datatypes`
Схема XSD	`http://www.w3.org/2000/10/XMLSchema`
Словарь форматирования XSL	`http://www.w3.org/1999/XSL/Format`
Словарь преобразований XSLT	`http://www.w3.org/1999/XSL/Transform`
XHTML	`http://www.w3.org/1999/XHTML`

Резюме

Сегодня вы познакомились с коллизиями имен, которые могут возникнуть при объединении документов из различных источников. Авторы документов могут спокойно создавать собственные имена элементов и атрибутов, а значит, могут существовать элементы с одинаковыми именами, но относящиеся к различным типам данных. Пространства имен предотвращают коллизии имен, при указывании уникальных идентификаторов для элементов с одинаковыми именами, но относящихся к различным типам данных. Таким образом, пространство имен представляет собой коллекцию имен элементов и атрибутов, характеризующую определенный класс данных.

Пространства имен идентифицируются с помощью идентификаторов URI, которые могут представлять собой адреса URL или имени URN. Авторы документов используют идентификаторы URI для обеспечения "глобальной" уникальности каждого пространства имен. URI, которые повторяют по форме адреса URL, приложением XML не разрешаются; вместо этого все идентификаторы URI представляют собой метки, указывающими приложению, как именно следует обрабатывать элементы и атрибуты.

Для каждой определенной технологии XML существуют свои стандартные пространства имен. Использование этих стандартных пространств имен в ряде случае оказывается обязательным.

Вопросы и ответы

Что такое конфликт имен?

Конфликт имен возникает в том случае, когда в одном документе предоставлены элементы или атрибуты с одинаковыми именами, но относящиеся к различным классам данных.

Когда чаще всего возникают конфликты имен?

Чаще всего конфликты имен возникают в том случае, когда данные XML из разных источников объединяются в одном экземпляре документа. В этом случае возможна ситуация, когда элементы или атрибуты с одинаковыми именами, но относящиеся к различным классам данных.

Как можно избежать конфлитка имен?

Пространства имен предоставляют разработчику очень простой механизм уникальной идентификации дескрипторов с одинаковыми именами для избежания конфликтов имен.

Чем синтаксис пространства имен по умолчанию отличается от синтаксиса явно определенного пространства имен в том же экземпляре документа?

В пространстве имен по умолчанию нет префикса использованного в оставщейся части экземпляра документа XML. Имена элементов и атрибутов, которые не содержат префикс, относятся к пространству имен по умолчанию. Явно указанное пространство имен включает префикс, который указывается в открывающем и закрывающем дескрипторах всех элементов и атрибутов, относящихся к этому пространству имен.

Зачем использовать стандартное пространство имен для идентификации элементов XML, принадлежащих схеме XSD?

Стандартное пространство имен (`http://www.w3.org/2000/10/XMLSchema`) распознается приложениями и синтаксическими анализаторами XML, содержащими встроенными синтаксическими правилами XSD. Действительно, только стандартное пространство имен идентифицирует элементы и атрибуты документа XML, принадлежащие схеме XSD; таким образом, ничто иное не позволит синтаксическому анализатору проверить экземпляр документа XML в соответствии с ограничениями, указанными в схеме XSD.

Упражнение

Проверьте свои знания, полученные в результате изучения материала настоящего урока. Ответ вы найдете в Приложении A, "Ответы на вопросы".

Во время изучения уроков первой недели вы создали документ XML, соответствующий всем синтаксическим правилам и стандартам, который представляет собой разметку коллекции музыкальных компакт-дисков. Затем вы проверяли этот документ с использованием схем DTD, XSD и XDR. Сегодня вы добавите новый класс данных в язык разметки MCML. Новый класс данных используется для разметки записей на обычных виниловых пластинках, а не на компакт-дисках.

Выполните следующие действия.

1. Добавьте в язык MCML запись, соответствующую пластинке *King of Blue* исполнителя *Miles Davis*, содержащей *11* записей.
2. Укажите, что новый класс данных относится к пространству имен с идентификатором URI `urn:myoldies:LPs`.
3. Объявите префикс `vinyl` и используйте его должным образом в разметке MCML.
4. Создайте новый тип элемента для использования при разметке данных о пластинке во избежание коллизии имен с данными, относящимися к записям на компакт-дисках.

День 9-й

Язык XPath

Вы уже изучили, что XML предоставляет эффективный способ описания высокоструктурированных данных с помощью самоописывающихся дескрипторов. Однако сам по себе XML не обеспечивает средства поиска определенных наборов данных, сохраненных в документе. Язык XPath (XML Path Language) позволяет находить определенные части данных, сохраненных в экземпляре документа.

Сегодня рассматриваются следующие темы.

- Важность языка XPath для обнаружения определенных компонентов данных в структуре XML.
- Основы синтаксиса и использования языка XPath.
- Как обнаружить семь типов узлов, выраженных с помощью XML.
- Основы перемещения с помощью языка XPath.
- Использование выбранных операторов и функций XPath.

Что такое XPath

Язык XPath позволяет обнаруживать определенные узлы в дереве документа XML. В отличие от XML, XPath не считается структурным. Вместо этого он базируется на языке выражений, применяемом в других технологиях XML для обнаружения определенных узлов структуры XML и перехода между ними. На данный момент консорциум *W3C* предлагает версию 1.0 языка XPath.

XPath разрабатывался для работы с преобразованиями XSLT, которые позволяют преобразовывать документы XML к другим формам, например другим документам XML или HTML, а также с языком XPointer (XML Pointer Language), который предоставляет метод "указания" на определенную информацию в документе XML. В итоге члены консорциума *W3C* разработали отдельный язык выражений для обеих технологий. Спецификация языка XPath опубликована 16 ноября 1999 года, в тот же день, когда была объявлена и спецификация XSLT. Подробно о роли языка XPath в преобразованиях XSLT вы узнаете в уроке дня 16-го, когда создадите собственный вариант преобразований документов с языка XML на язык HTML. Язык XPointer подробно рассматривается в уроке дня 11-го.

XPath выступает в роли подъязыка преобразований XSLT и языка XPointer. Выражение XPath используется для манипуляции строками, в числовых вычислениях, а также в булевой логике. Однако основное назначение языка XPath, как и следует из его названия, — возможность адресации частей документа с помощью представления путей, это

похоже на разрешения URL при перемещении по иерархии документа XML. XPath состоит из простых выражений, аналогичных выражениям, используемых в других языках; он содержит символы путей, имена узлов, а также зарезервированные слова, причем при этом используется синтаксис, отличный от синтаксиса XML. XPath обнаруживает компоненты документа XML, однако функционирует он на абстрактном уровне структуры документа, а не на уровне синтаксиса. Вы поймете, что это означает, по мере исследования документов XML с использованием выражений XPath в настоящем уроке.

Узлы дерева документа XML

Как вы уже знаете, данные, сохраненные в документе XML, представлены в виде иерархической древовидной структуры узлов. Некоторые из этих узлов содержат другие узлы. Например, элемент может содержать другие элементы. Элементы также содержат текстовые данные; однако в подобных случаях текстовые данные представляются в виде отдельного дочернего узла. Доступ к данным в элементе требует от вас прохождения по структуре документа к текстовому дочернему узлу узла элемента. Вы уже знакомы с узлами элементов и атрибутов, а также знаете о существовании узлов для комментариев в экземпляре документа XML. На самом деле, XPath имеет дело с семью различными видами узлов. Использование XPath приводит к созданию структурированной иерархии узлов в экземпляре документа, поддерживающей функции поиска.

Итак, в основном XPath предназначен для перемещения по документу XML для достижения его определенного узла. Для перехода используются выражения, построенные в соответствии с синтаксическими правилами XPath. При вычислении выражений XPath результатом оказывается объект данных, который можно отнести к одной из следующих категорий.

- **Узел.** Находится один выбранный узел.
- **Набор узлов.** Выбираются несколько узлов, обладающих общими характеристиками.
- **Булево выражение.** Выражение, принимающее значение `true` или `false`.
- **Строка.** Разрешение выражения приводит к строке, зависящей от содержания одного или нескольких узлов.
- **Число.** Результат вычислений, полученный при определении значений выражений XPath, может представлять собой число с плавающей запятой.

Семь узлов документа в XPath

Дерево узлов, полученное с помощью XPath, очень похоже на древовидную структуру, соответствующую объектной модели документа DOM. Объектная модель DOM предоставляет в ваше распоряжение программный интерфейс API, который позволяет вам получать доступ, получать сведения, изменять и добавлять узлы документа программным образом. Подробно с объектной моделью документа DOM вы познакомитесь в уроке дня 12-го.

XPath содержит следующие семь типов узлов: корень, элемент, атрибут, комментарий, текст, инструкция обработки, пространство имен. Дерево XPath содержит только один корневой узел, объединяющий все остальные узлы дерева. Не путайте корневой узел с корневым элементом документа XML. Корневой узел содержит все остальные узлы, в том числе корневой элемент, все другие элементы, инструкции обработки, пространства имен, комментарии или текстовые данные всего экземпляра документа.



Корневой узел в дереве XPath — это не одно и то же, что корневой элемент в экземпляре документа XML. Корневой узел содержит все остальные узлы, в том числе корневой элемент, все другие элементы, инструкции обработки, пространства имен, комментарии и текстовые данные.

Взаимосвязи узлов

Как уже отмечалось раньше, корневой узел выступает в роли контейнера для всех остальных узлов в экземпляре документа XML. Корневой узел и узлы элементов содержат упорядоченные списки дочерних узлов. Каждый узел, за исключением корневого, имеет и родительский узел. Родительские узлы могут не содержать ничего, но могут содержать и любое количество дочерних узлов и узлов более низкого уровня (внучатых). Интересно, что дочерними могут быть только такие узлы, как элементы, комментарии, инструкции обработки и текстовые данные. Вспомним, что атрибут в чем-то аналогичен прилагательному; он дополняет элемент, который мы сравниваем с существительным в обычном языке. Атрибут не является дочерним по отношению к элементу; он просто предоставляет дополнительные сведения об элементе. Атрибут связан с элементом, но не является его дочерней конструкцией. Хотя это и покажется странным и противоречивым, но атрибут может иметь "родителя", но не будет по отношению к нему дочерним. "Родитель" содержит свои дочерние элементы. Как вы узнали из урока дня 2-го, даже пустые элементы, которые не имеют никакого содержания, могут иметь атрибуты. Атрибут не считается содержанием элемента. С учетом подобной логики говорят, что узел пространства имен может иметь "родителя", но не будет по отношению к нему дочерним. Пространство имен просто предоставляет дополнительные сведения о "родителе". В действительности узел пространства имен — это специальный экземпляр узла атрибута.



Узлы атрибутов и пространств имен имеют родительские узлы, однако не являются дочерними по отношению к ним в XPath. Узлы атрибутов и пространств имен предоставляют сведения о родительских узлах, однако не будут их содержанием.

Знакомство с узлами экземпляра документа XML

Для того чтобы лучше разобраться с компонентами дерева узлов XPath, документ XML нужно представить в виде диаграммы узлов XPath, соответствующих документу. Вы увидите версию документа, с которым работали в первых главах настоящей книги, сначала в виде XML, а затем в виде дерева узлов, в котором можно проводить поиск с помощью выражений XPath. В листинге 9.1 приведен документ XML `message01.xml`, который вы использовали раньше. Для представления всех семи узлов XPath теперь документ содержит инструкции обработки и комментарии.

Листинг 9.1. Документ XML, содержащий семь узлов XPath `message01.xml`

```
1: <?xml version="1.0"?>
2: <!-- Листинг 9.1 - message01.xml -->
3:
4: <note xmlns="urn:STY_XML_in_21Days:XPath">
5:
6:     <?MessageProcessor command = "AcceptMessage" ?>
7:     <!-- Это пример инструкции обработки -->
```

```
8:
9:      <message type="phone message">
10:          Remember to buy milk on the way home from work
11:          <status>urgent</status>
12:      </message>
13: </note>
```



Этот относительно простой экземпляр документа содержит корневой элемент `note`, начинающийся со строки 4; он содержит пространство имен с идентификатором URI `urn:STY_XML_in_21Days:XPath`. Строка 6 образует инструкцию обработки. В уроке дня 14-го вы будете использовать инструкции обработки для связывания экземпляров документов XML с каскадными таблицами стилей CSS для придания результатам обработки документа, отображаемым на стороне клиента, более "привлекательного" вида. В уроке дня 15-го вы будете использовать подобные инструкции обработки для связывания таблиц стилей XSL с документами XML. Инструкции обработки предоставляют специальные указания приложению, обрабатывающему документ XML. Пример в строке 6 инициализирует команду `MessageProcessor`, которая применяется к данным сообщения, содержащимся в экземпляре документа XML.

Строки 2 и 7 представляют собой комментарии. Обратите внимание, что строка 2 находится до корневого элемента экземпляра документа XML, а строка 7 — в его рамках. Так как корневой узел находится за пределами коневого элемента, XPath способен обнаружить комментарий в строке 2 точно так же легко, как и в строке 7.

Объявление XML в строке 1 не содержится в узле документа. Вы могли предположить, что это инструкция обработки, но технически это не так, поэтому XPath не умеет находить эти сведения.

Остальные узлы документа представляют собой узлы элементов и атрибутов, которые XPath легко находит.

Для того чтобы лучше проиллюстрировать дерево узлов, полученное после использования XPath, посмотрите на рис. 9.1. Подобное дерево содержит все типы узлов вместе с их содержанием. Прямоугольники представляют каждый тип узла, известный XPath, вместе с содержанием узлов. Линии, соединяющие прямоугольники, иллюстрируют отношения между узлами. Вы можете связать представляющие узлы прямоугольники с действительными узлами, показанными в листинге 9.1. Внимательно изучите схему, воспользовавшись подробным описанием, которое следует дальше.

Показанный на рис. 9.1 элемент `Root` содержит все остальные узлы дерева XPath. Объявление XML в строке 1 листинга 9.1 не представлено на диаграмме по той причине, что оно отсутствует в окончательном дереве XPath. Первый дочерний элемент элемента `Root` — элемент `Comment`, который в экземпляре документа указан до корневого элемента `note`. Затем следует узел `Element` для корневого элемента `note`.

Линии, исходящие из узла корневого элемента `note`, заканчиваются на узле пространства имен, узле инструкции обработки, узле комментария, а также узле элемента `message`. С точки зрения XPath все эти узлы относятся к узлу `Element`. Узел пространства имен `urn:STY_XML_in_21Days:XPath` не является дочерним по отношению к узлу элемента `note`, даже несмотря на то, что последний выступает родительским по отношению к нему. Узел инструкции обработки `MessageProcessor command = "AcceptMessage"`, узел комментария, а также узел элемента `message` являются дочерними по отношению к узлу элемента `note`.

Узел `message` — родительский по отношению к узлу атрибута `type`, несмотря на то, что сам атрибут не будет дочерним по отношению к своему "родителю". Узел `message` содержит узел `Text`, а также узел элемента `Status`. Узел элемента `Status` содержит узел `Text`.

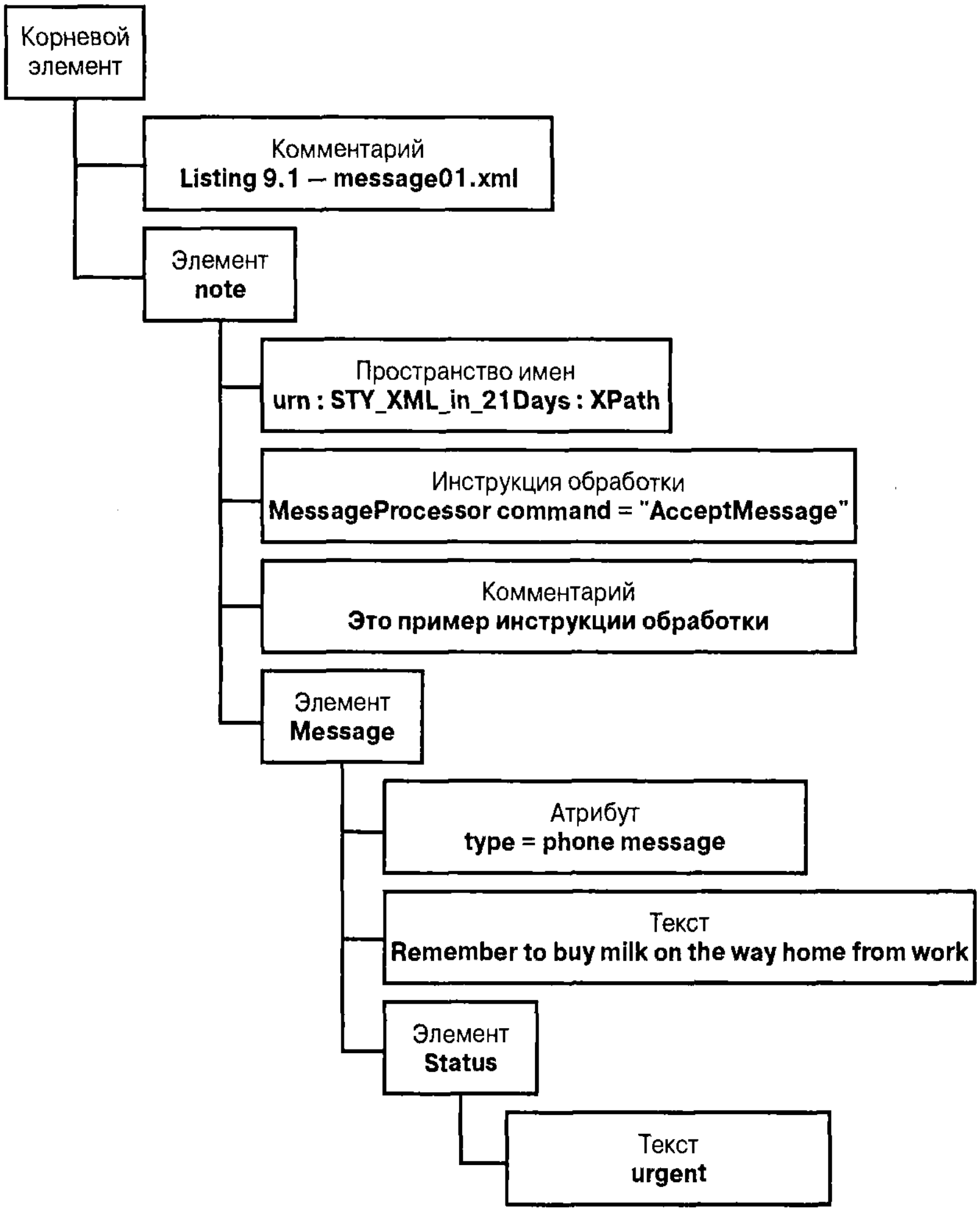


Рис. 9.1. Дерево XPath для документа `message01.xml` *из листинга 9.1*

Сложные отношения узлов

Вы уже познакомились с простыми отношениями между узлами в экземпляре документа `message01.xml`. Однако некоторые отношения могут, по крайней мере на первый взгляд, показаться менее очевидными. Вы уже познакомились с родительскими и дочерними элементами. Теперь вы познакомитесь с отношениями предков и потомков, также широко используемыми в XPath.



Порядок узлов в дереве XPath определяется порядком, в котором они следуют в исходном документе XML. Этот порядок известен как *порядок документа*. Он начинается с корневого узла и "спускается" вниз по дереву XPath по всем элементам, комментариям, инструкциям обработки, тексту, атрибутам и пространствам имен дерева документа.

Каждый узел в дереве XPath содержит строковое значение, которое используется XPath при сравнении узлов. Строковое значение элемента представляет собой полное текстовое содержание элемента, плюс текстовое содержание всех его потомков. Например, строка 11 из листинга 9.1

```
11:          <status>urgent</status>
```

содержит разметки и содержание элемента `status`. Теперь рассмотрим узел `Text` элемента `status`, строковое значение которого `urgent`. Узел элемента `status` содержит только связанный с ним в качестве потомка узел `Text`, поэтому только он и включен в строковое значение этого узла. Текстовые значения узла элемента `status` и узла `Text` совпадают.



> Строковое значение узла любого элемента определяется после объединения строковых значений всех текстовых узлов, являющихся его потомками.

Строковое значение элемента `message` из рис. 9.1 представляет собой более интересный вариант. Это текстовое содержание узла элемента `message`, который, в свою очередь, включает текстовые узлы своих узлов-потомков согласно порядку документа. Таким образом строковым значением узла элемента `message` будет `Remember to buy milk on the way home from work urgent`. Эта строка получена после объединения строковых значений узла и всех его узлов-потомков. Строковое значение дочернего узла по отношению к узлу элемента `message` равно `Remember to buy milk on the way home from wor`, а строковое значение узла элемента `status` остается равным `urgent`.



> Узлами-потомками являются все узлы, которые следуют после определенного узла согласно порядку документа.

Используя полученные сведения, сможете ли вы определить строковое значение корневого элемента в документе из листинга 9.1? Оно должно быть определено после объединения строковых значений всех текстовых узлов-потомков. Таким образом, оно равно `Remember to buy milk on the way home from work urgent` точно так же, как и значение узла элемента `message`, поскольку в экземпляре документа больше не существует других текстовых узлов-потомков.

Строковое значение узлов атрибутов содержит только значения атрибутов. Таким образом, строковое значение узла атрибута `type` равно `phone message`. Строковые значения узлов комментариев равны текстам самих комментариев, или `Листинг 9.1 - message01.xml` для узла комментария из строки 2 листинга 9.1.

Значения узлов инструкций обработки равны инструкциям после первого аргумента. Так, строка 6 в документе `message01.xml`, равная

```
6:       <?MessageProcessor command = "AcceptMessage" ?>
```

содержит разметку инструкции обработки. Первый аргумент инструкции равен `MessageProcessor`, поэтому значение узла инструкции обработки равно `command = "AcceptMessage"`. Это особенно важно для инструкций обработки, которые содержат URL, как в случае связанных таблиц стилей. Подробно с таблицами стилей вы познакомитесь в уроках дней 14-го и 15-го.

Строковые значения узлов пространств имен совпадают с их идентификаторами URI. Например, строковое значение узла пространства имен, связанное с элементом `note`, равно `urn:STY_XML_in_21Days:XPath`.

В табл. 9.1 подытожены основные сведения, полученные вами обо всех семи типах узлов XPath.

Таблица 9.1. Семь типов узлов XPath

Тип узла	*Описание*	*Строковое значение*
Root	Существует только один корневой узел, который содержит все остальные узлы документа	Объединенные строковые значения все текстовых узлов-потомков согласно порядку документа
Element	Узлы элементов соответствуют элементам документа XML и могут содержать другие узлы	Объединенные строковые значения все текстовых узлов-потомков согласно порядку документа
Attribute	Узлы атрибутов представляют атрибуты XML; родительскими по отношению к ним будут узлы элементов, однако сами они не будут дочерними по отношению к последним	Значение атрибута
Namespace	Узлы пространств имен представляют пространства имен XML; родительскими по отношению к ним будут другие узлы, однако сами они не будут дочерними по отношению к родительским узлам	Идентификатор URI пространства имен
Processing instruction	Узлы инструкций обработки представляют инструкции обработки XML	Значение, указанное после первого аргумента инструкции обработки
Text	Узлы `Text` содержат текстовые данные элементов	Текстовые данные, содержащиеся в текстовом узле
Comment	Узлы комментариев содержат текстовые данные элементов	Текст комментария

Разновидности отношений XPath

Вы узнали о существовании дочерних и родительских узлов, а также о том, что некоторые из узлов имеют родительские узлы, но не являются по отношению к ним дочерними. Например, узлы атрибутов и пространств имен предоставляют информацию о родительских узлах, но не являются их содержанием, а значит, не будут дочерними по отношению к ним. Вы уже познакомились с концепцией дочерних и родительских отношений, используемых в выражениях XPath.

XPath различает одиннадцать основных разновидностей отношений узлов в документе XML. При создании выражений XPath вы предоставляете адрес, а также путь его поиска. Это *ветвь*, по которой можно перемещаться и в обратном направлении. Например, для того чтобы попасть от текущего узла к необходимому, путь поиска нужно выразить с помощью подобных отношений ветвей.

Для того чтобы вы лучше разобрались во всех этих отношениях, вы изучите каждое из них в отдельном подразделе дальше в настоящем уроке. Вам представят как текстовое описание, так и графическое отображение узлов, находящихся в определенных взаимных отношениях. Выбранные узлы — это узлы, полученные в результате разрешения дескриптора отношений или в результате перехода в определенном направлении поиска. Программа находит узлы, которые характеризуются отношениями, определяемыми выражениями XPath. Это означает, что один и тот же путь поиска, нача-

тый с различных узлов, приведет к разным результатам. Изучая приведенные ниже примеры, вы увидите различные варианты подобной концепции. Однако сначала очень важно разобраться, что же понимается под контекстным узлом и выбранными узлами. Изучите пример простого документа XML, приведенный в листинге 9.2.

Листинг 9.2. Документ XML, содержащий сложные типы элементов, `driveway.xml`

```
1:  <?xml version="1.0"?>
2:  <!-- Листинг 9.2 - driveway.xml -->
3:
4:  <driveway>
5:              <car>
6:                   <mfgr>Ford</mfgr>
7:                   <model>Mustang</model>
8:                   <color>green</color>
9:              </car>
10: </driveway>
```

Предположим, что контекстный узел для выражения XPath, применяемого к этому документу XML, — это узел элемента `car`, а вам необходимо определить родительский узел по отношению к нему. Выбранным узлом, в данном случае родительским, является узел элемента `driveway`. Точно также, если вы рассматриваете тот же контекстный узел (узел элемента `car`), то выбор его дочерних узлов приведет к обнаружению узлов таких элементов, как `mfgr`, `model` и `color`. В каждом случае выбранные узлы определяются по их относительной связи с контекстным узлом. Естественно, что в силу подобной природы XPath, очень важно помнить, какой именно узел выбран контекстным при использовании выражений для обнаружения тех или иных узлов.

Концепция self

В выражении XPath контекстный узел может быть именно тем узлом, который вам необходим. Тогда контекстный узел и выбранный узел — это одно и то же. На рис. 9.2 показано дерево, представляющее узлы документа XML. Оно очень похоже на те древовидные структуры, с которыми вы уже сталкивались раньше при изучении предыдущего материала книги. В рамках нашего рассмотрения указывать названия узлов необязательно. Линии, соединяющие узлы, иллюстрируют их взаимные отношения. Очевидно, что узлы подчиняются всем узлам, по отношению к которым в приведенном на рисунке дереве они занимают положение ниже. Кроме того, те из них, которые находятся на одном уровне, считаются "братскими" и характеризуются одинаковым количеством родительских узлов, расположенных на более высоких уровнях дерева. Некоторые родительские узлы содержат несколько дочерних.

Вы можете видеть, что контекстный узел находится во второй ветви дерева узлов. В соответствии с легендой, представленной в верхнем правом углу, выбранный узел отображается серым. В данном случае выбранный и контекстный узел совпадают. Другими словами, вычисление выражение XPath для поиска узлов `self` возвращает нас к контекстному узлу. Концепция `self` очень важна в XPath, поскольку ее можно объединить с другими видами отношений, когда контекстный узел входит в группу выбранных узлов, полученную вычислением выражения. Дальше в настоящем уроке вы увидите еще целый ряд подобных примеров.

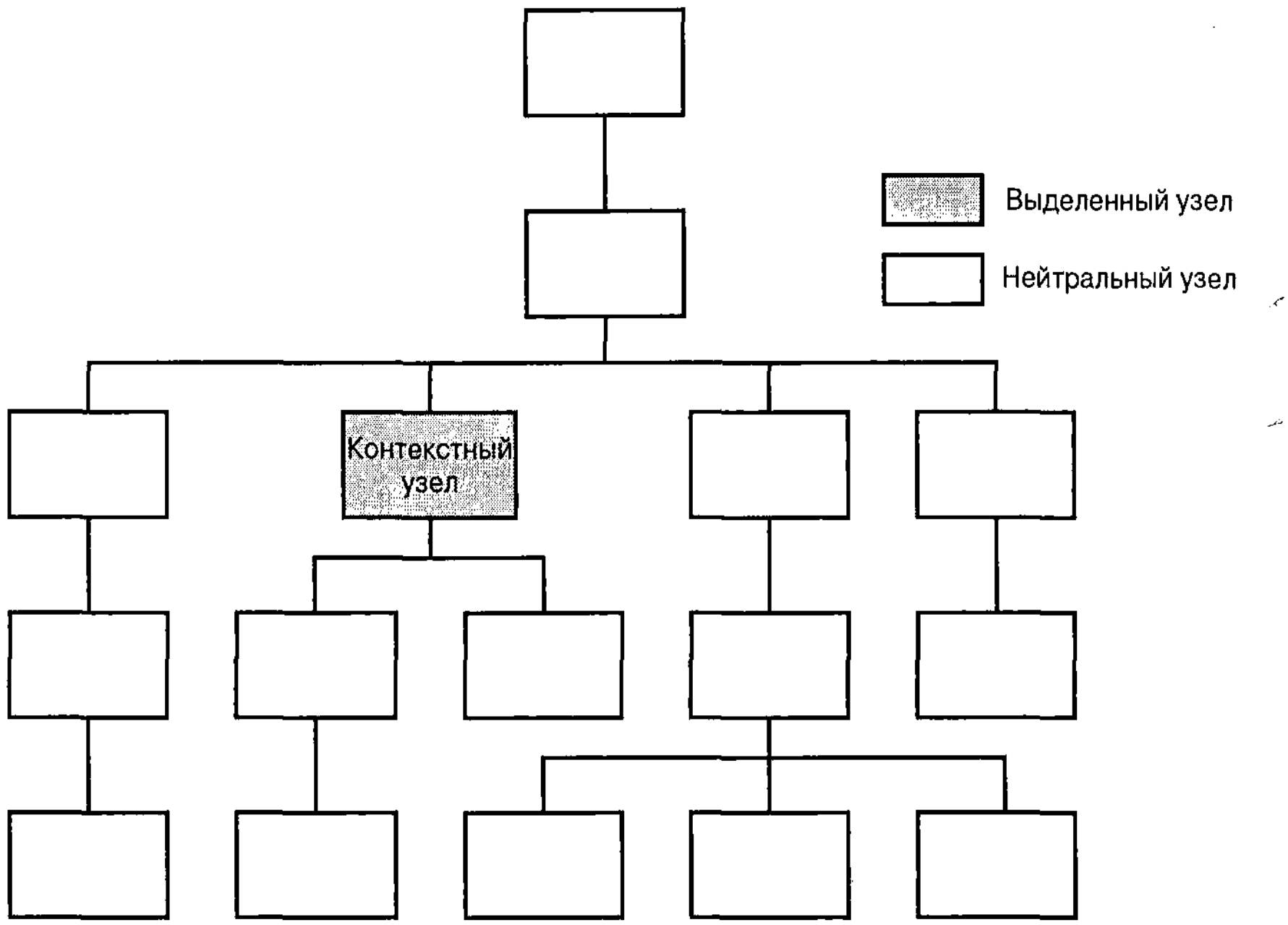


Рис. 9.2. Дерево узлов XPath, в котором выбран только контекстный узел

Родительский узел

Вы уже сталкивались с родительскими узлами сегодня. Вы уже знаете, что обычно такие узлы содержат дочерние, за исключением атрибутов и пространств имен, о чем также говорилось раньше. Родительские узлы занимают более высокое место в дереве по отношению к своим дочерним узлам. По отношению к контекстному узлу родительский узел находится на одно поколение (или уровень) выше; таким образом, выбранный узел, полученный после вычисления выражения XPath для поиска родительского узла, должен представлять собой один узел, расположенный уровнем выше. Соответствующий пример приведен на рис. 9.3.

Обратите внимание, что правильный выбор в случае родительских отношений всегда должен привести к одному узлу. В отношениях, рассматриваемых нами дальше, вы будете сталкиваться и с несколькими узлами, так называемыми наборами узлов, выбранными после вычисления выражения XPath.



При использовании XPath выражение для поиска может привести к целой группе узлов с одинаковыми характеристиками по отношению к другим узлам. Подобная коллекция называется *набором узлов*.

Дочерние отношения

Как уже отмечалось раньше, дочерний узел содержится в родительском. Выражение XPath для поиска дочерних узлов находит только те узлы, которые находятся на уровень ниже по отношению к контекстному. Контекстный узел может содержать несколько дочерних, но дочерние узлы всегда находятся ровно на один уровень ниже родительского. Пример выбранных дочерних узлов приведен на рис. 9.4.

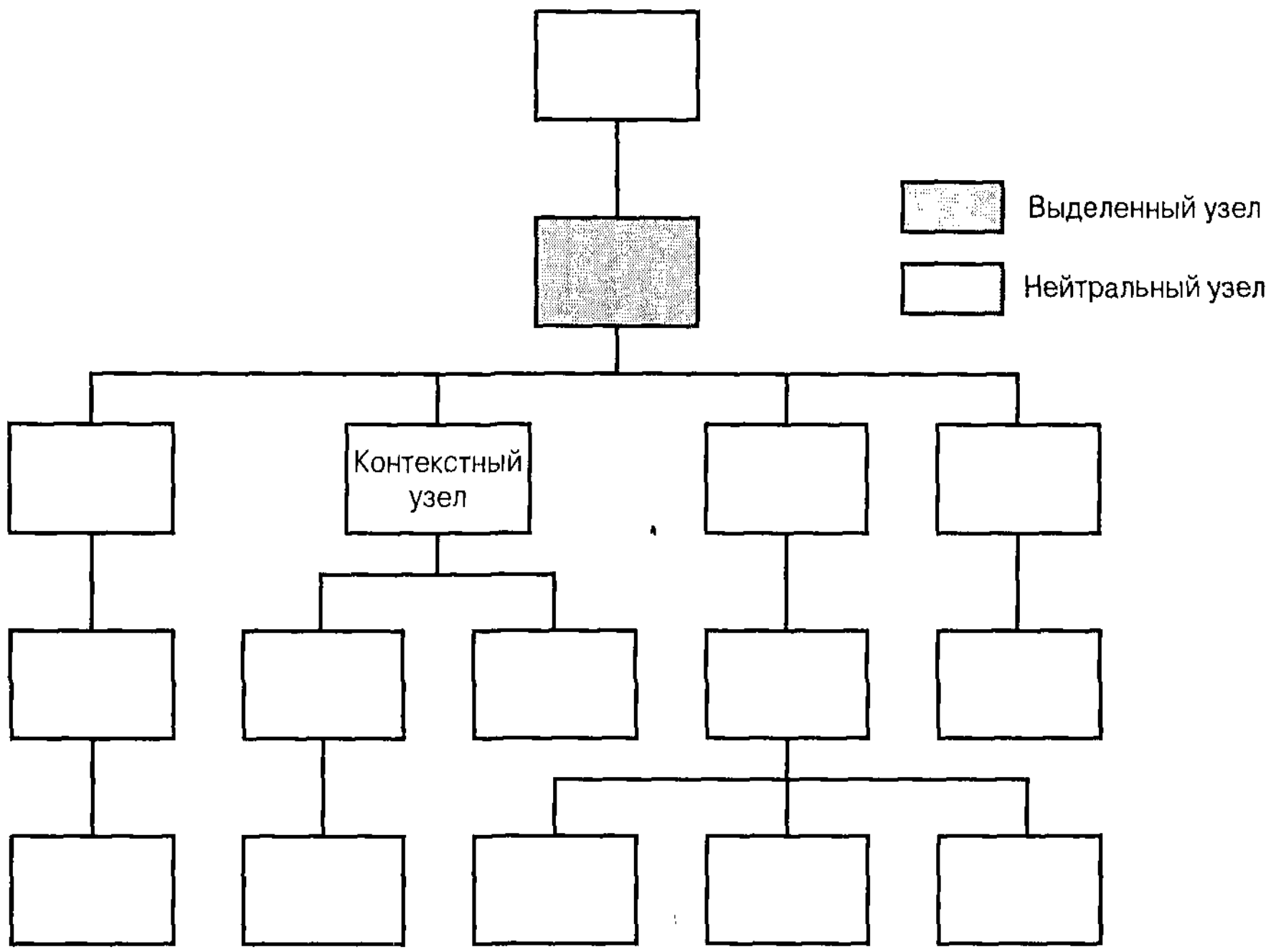


Рис. 9.4. Дерево узлов XPath, отражающее родительское отношение узлов

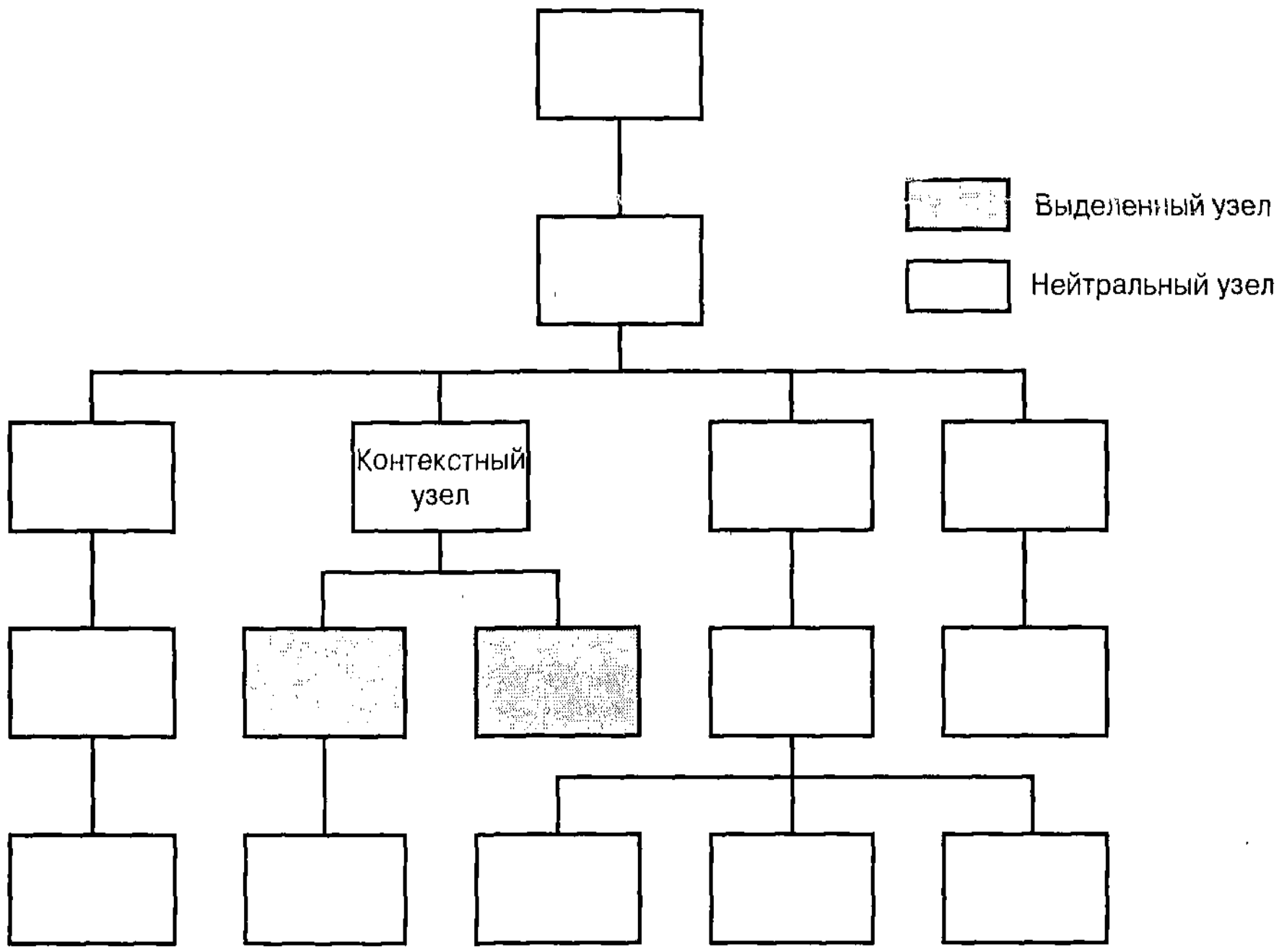


Рис. 9.4. Выбраны два дочерних узла контекстного узла

Обратили ли вы внимание на то, что один из дочерних узлов, в свою очередь, также содержит дочерний узел? Этот потомок не был выбран в результате вычисления выражения XPath для поиска дочерних узлов по той причине, что он находится больше, чем на один уровень ниже по отношению к контекстному узлу. Скоро вы узнаете, как можно задать выбор всех узлов-потомков.

Родительские и узлы более высокого уровня

К узлам-предкам относятся как родительский узел, так и родительский узел по отношению к нему и дальше до самого верха дерева XPath, а в обратном порядке документа — вплоть до корневого узла. На рис. 9.5 приведен пример выбранного узла и всех его узлов-предков.

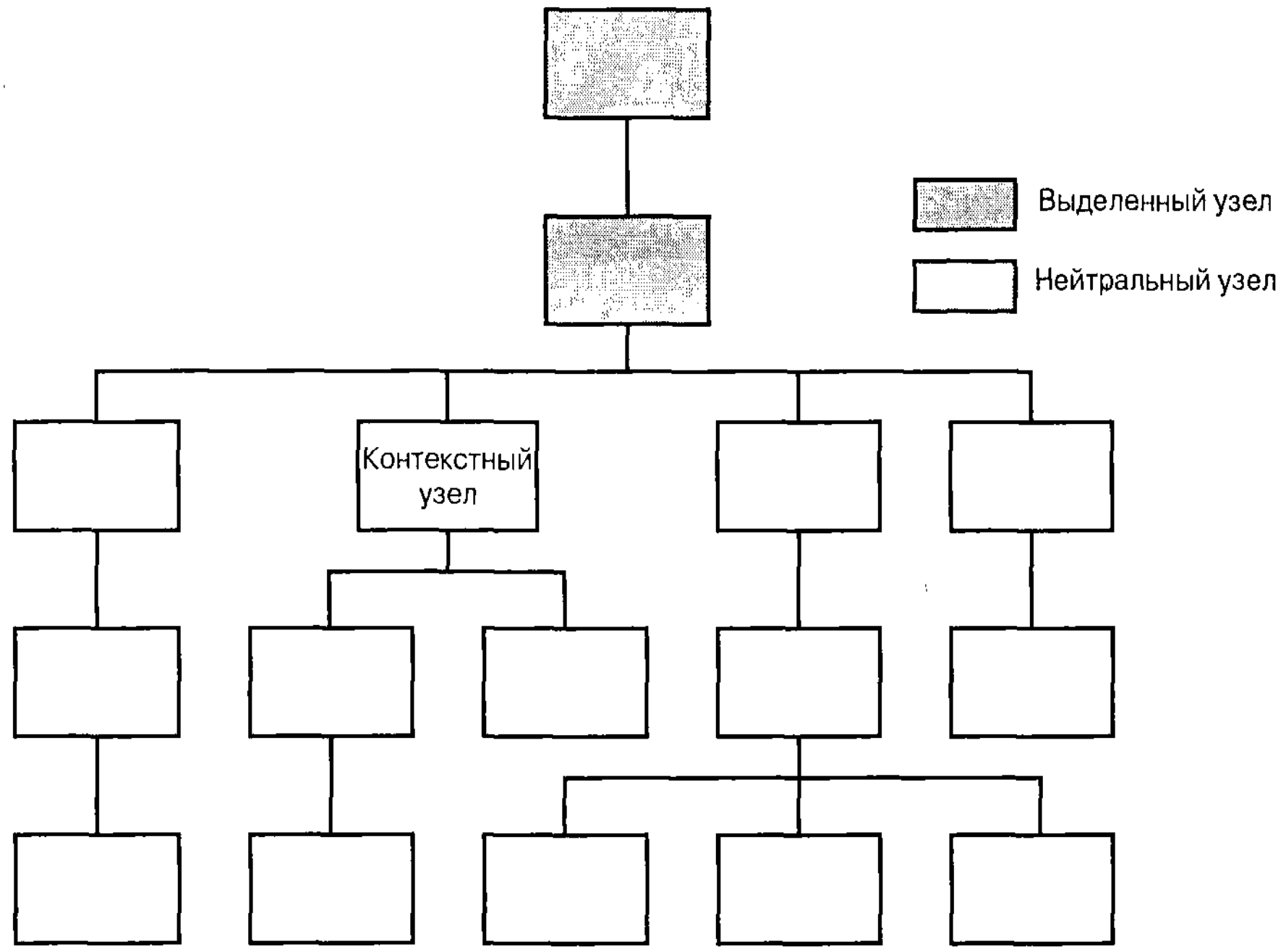


Рис. 9.5. Дерево XPath, в котором показаны отношения выбранного узла и узлов-предков

Вы уже знаете, что выражение XPath для поиска родительского узла приводит только к одному узлу, а выражение XPath для поиска узлов-предков приводит к выбору всех узлов более высокого уровня, которые непосредственно предшествуют контекстному узлу. Размер выбранного набора узлов определяется только количеством узлов-предков между контекстным и корневым узлами. Корневой узел — это узел-предок по отношению ко всем остальным узлам в дереве.

Предок самого себя

Отношение "предок самого себя" включает все узлы, обнаруженные с помощью выражения для нахождения узлов-предков, а также собственно сам контекстный узел. В случае нашего простого дерева XPath, после вычисления выражения "предок самого себя" будут обнаружены три узла, как показано на рис. 9.6 приведен пример выбранного узла и всех его узлов-предков.

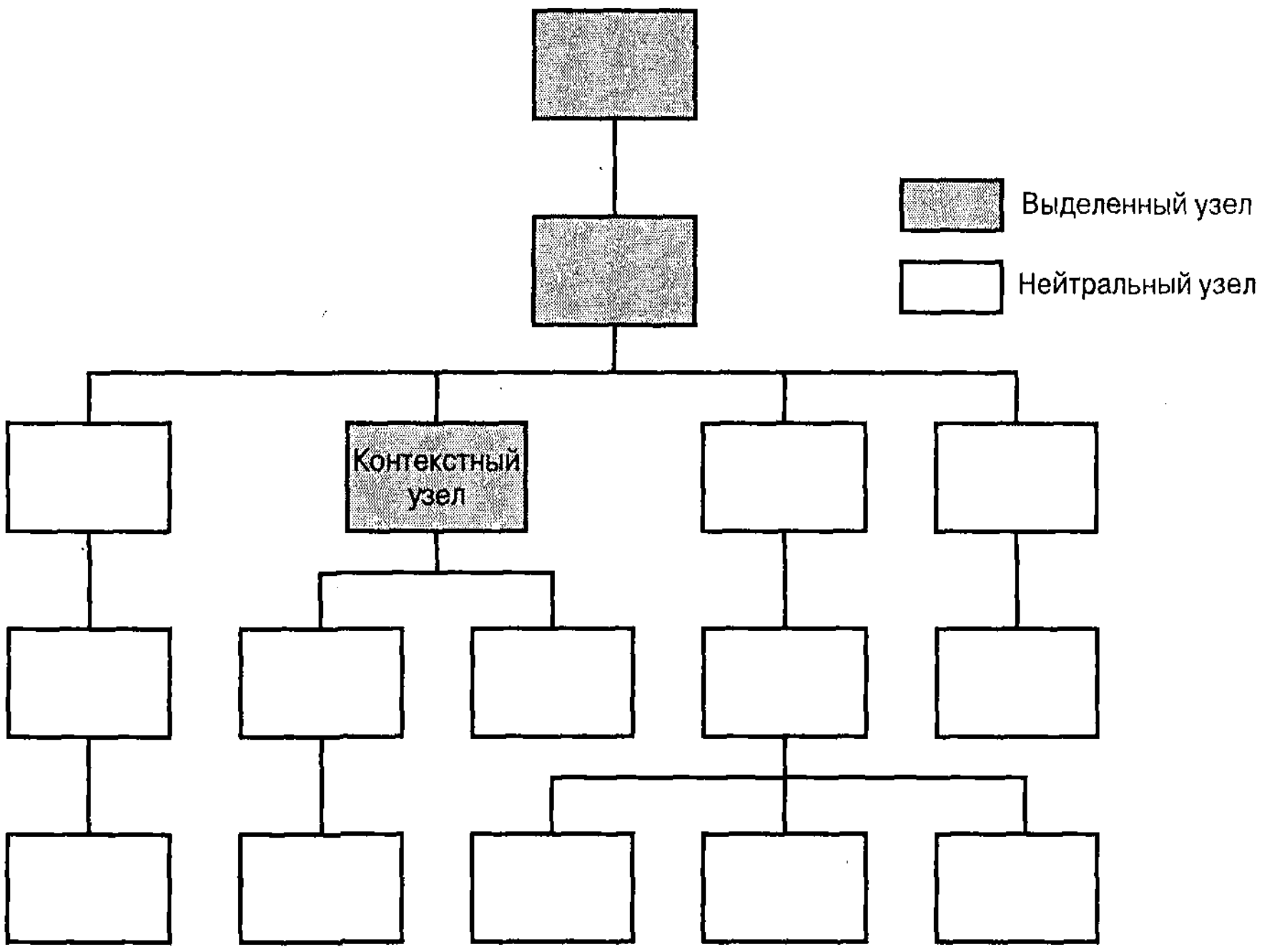


Рис. 9.6. Дерево XPath, в котором показаны отношения "предок самого себя"

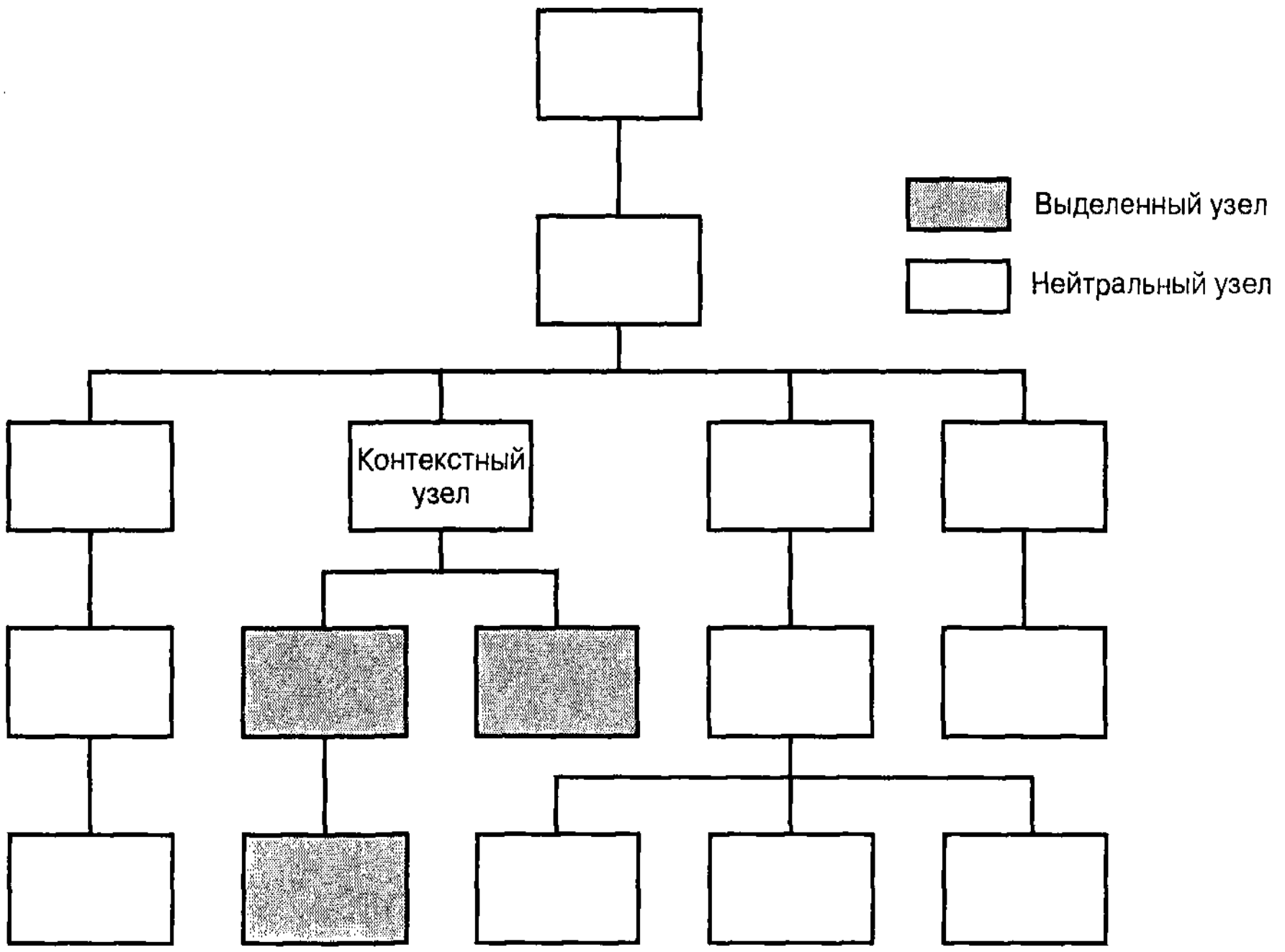


Рис. 9.7. Дерево XPath, в котором показаны отношения потомков

Потомок

Для выбора дочернего узла, а также всех внучатых узлов, а также всех узлов более низкого уровня, вам необходимо использовать отношения потомков. На рис. 9.7 показаны три узла, выбранные после вычисления соответствующего выражения. Каждый из этих узлов вложен в контекстный.

Отношение *потомков* может включать в себя любое количество узлов. Набор узлов начинается с контекстного узла и продолжается до тех пор, пока не будут достигнуты узлы самого нижнего уровня в дереве.

Потомок самого себя

Как и в случае отношений "предок самого себя", существует возможность выбрать набор узлов, который будет включать в себя все узлы-потомки, а также сам контекстный узел. Подобное отношение называется "потомок самого себя". Пример подобных отношений приведен на рис. 9.8, на котором выбраны четыре узла в нашем дереве XPath.

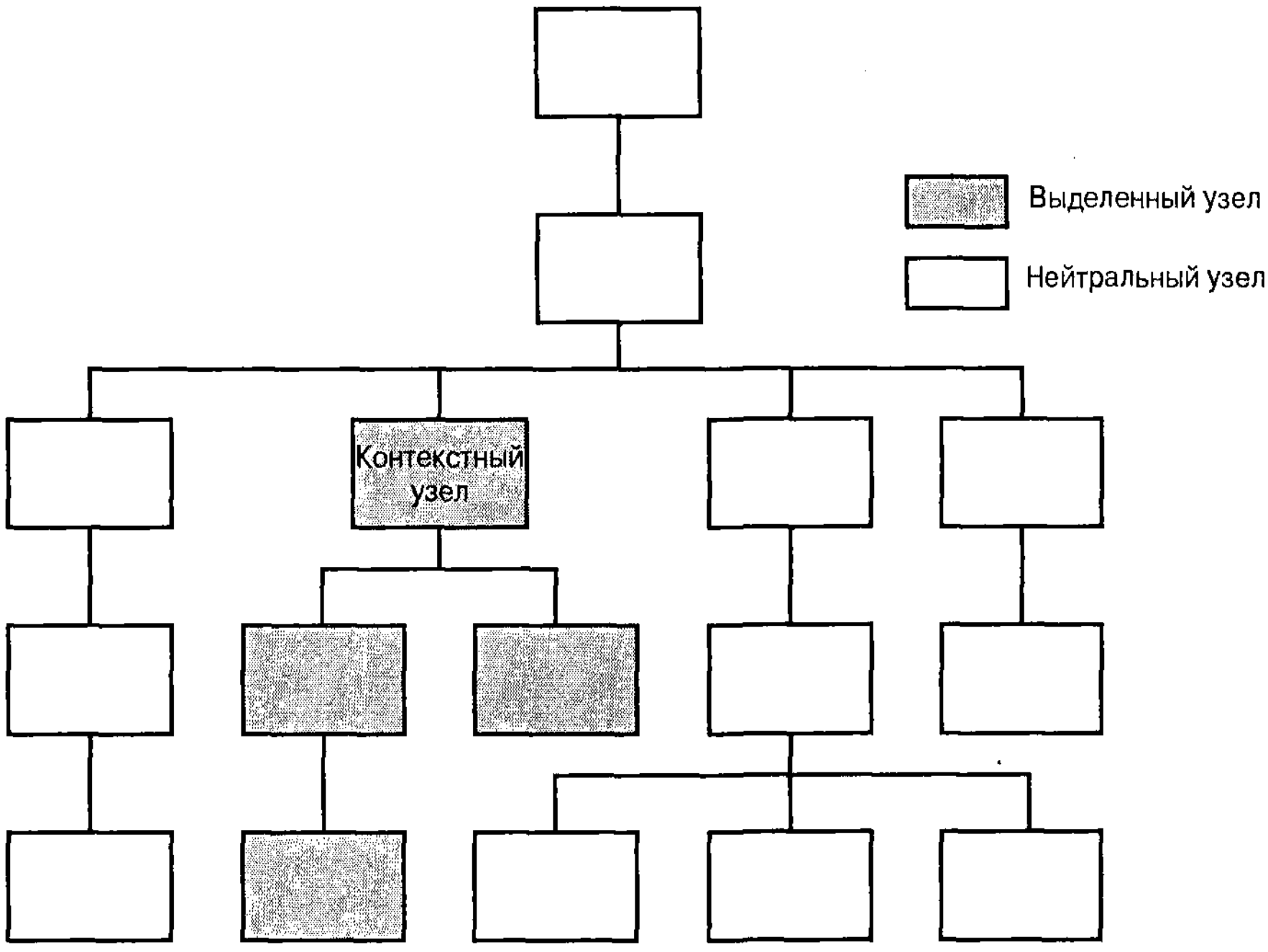


Рис. 9.8. Дерево XPath, в котором показаны отношения "потомок самого себя"

Отношение "потомок самого себя" включает контекстный узел, а также все его вложенные узлы, вплоть до самого нижнего уровня дерева.

Выбор узлов, следующих после контекстного узла

Вы уже узнали, что XPath может просматривать дерево узлов в документе как в прямом, так и обратном порядке. Концепция *последующих* узлов позволяет выбрать все узлы, которые существуют после контекстного. Следующие узлы начинаются с узлов одного

уровня с контекстным и включают в себя все потомки этих узлов. То, какие узлы считаются следующими, определяется порядком документа. Все узлы, которые расположен до контекстного узла, исключаются из отношений следующих узлов. Пример подобных отношений приведен на рис. 9.9, на котором выбраны семь узлов в нашем дереве XPath.

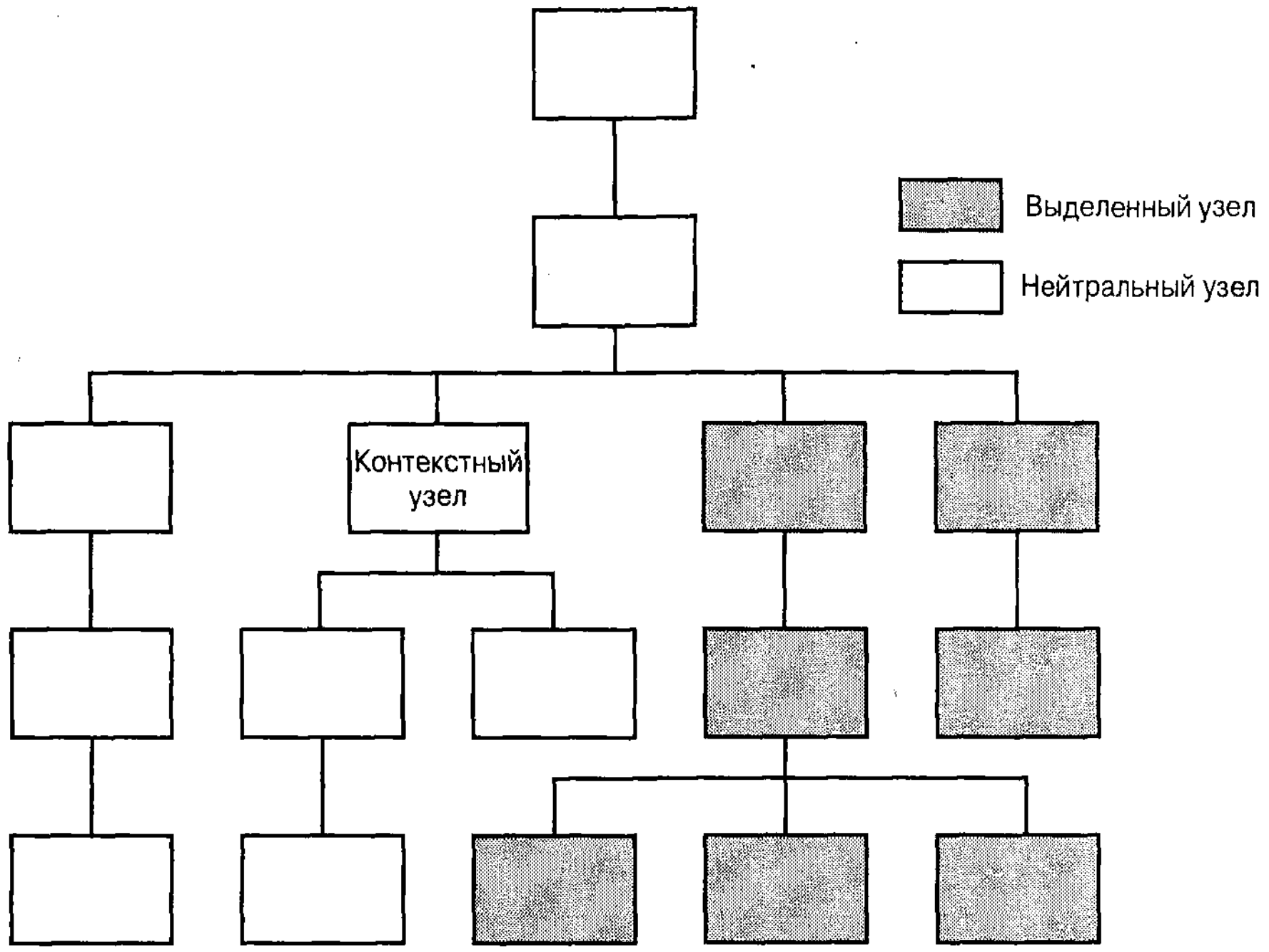


Рис.9.9. Дерево XPath, в котором показаны отношения последующих узлов

Вы обратили внимание на то, что отношения следующих узлов не включают в себя родительские и другие узлы-потомки по отношению к контекстному? Они включают в себя только узлы одного с ним уровня, а также все их узлы потомки.

Выбор последующих узлов одного уровня

Отношения последующих узлов одного уровня включают узлы, которые являются следующими узлами одного уровня в соответствии с порядком документа. Выбираются узлы только одного уровня с контекстным. Потомки выбранных узлов одного уровня из полученного набора исключаются. Соответствующий пример приведен на рис. 9.10.

Предыдущие узлы

Выбор предыдущих узлов реализуется практически так же, как и выбор следующих узлов, за исключением того, что выбираются узлы одного уровня и их потомки, которые следуют до контекстного узла в соответствии с порядком документа. Соответствующий пример приведен на рис. 9.11.

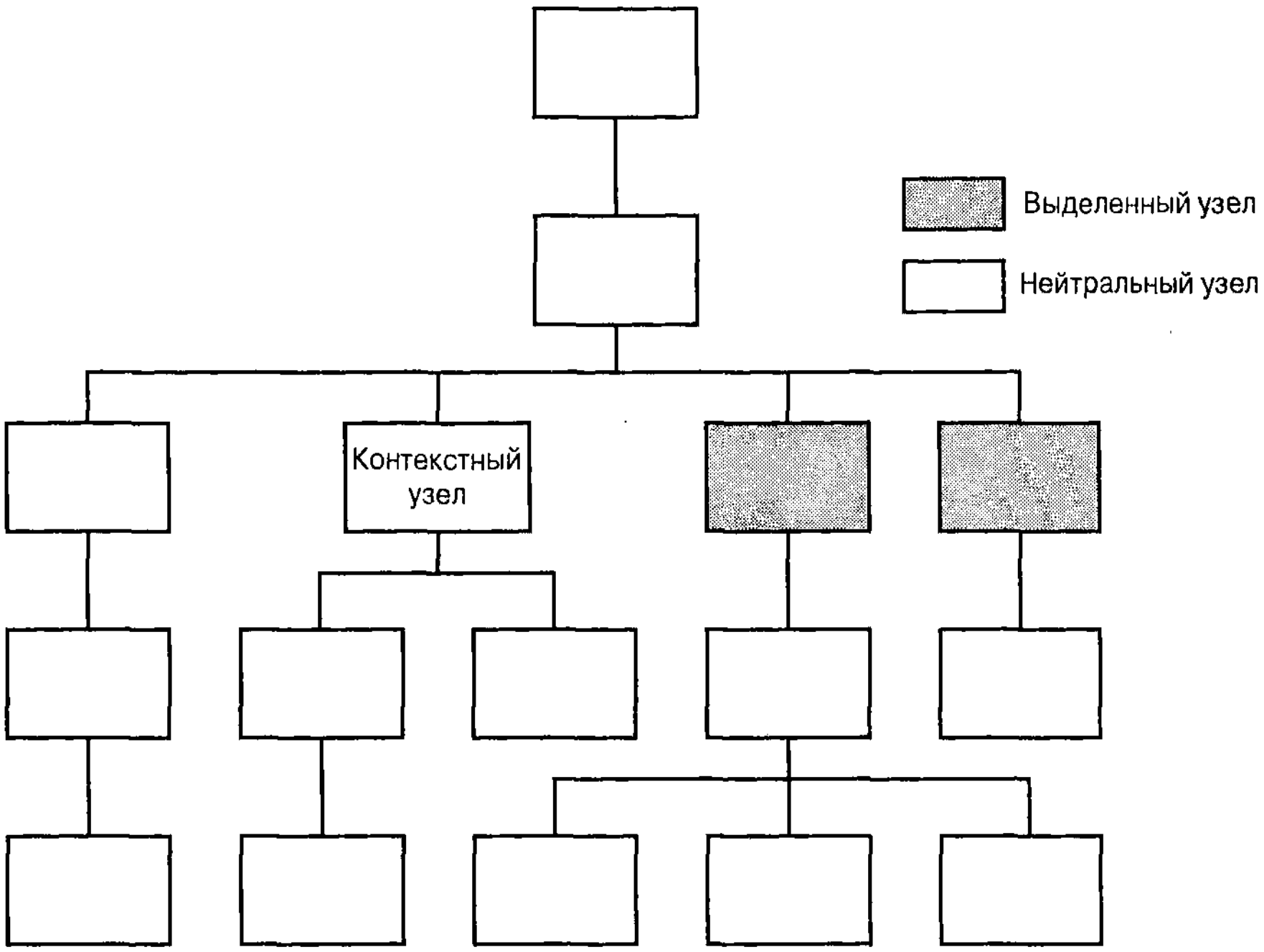


Рис. 9.10. Дерево XPath, в котором показаны отношения следующих узлов одного уровня

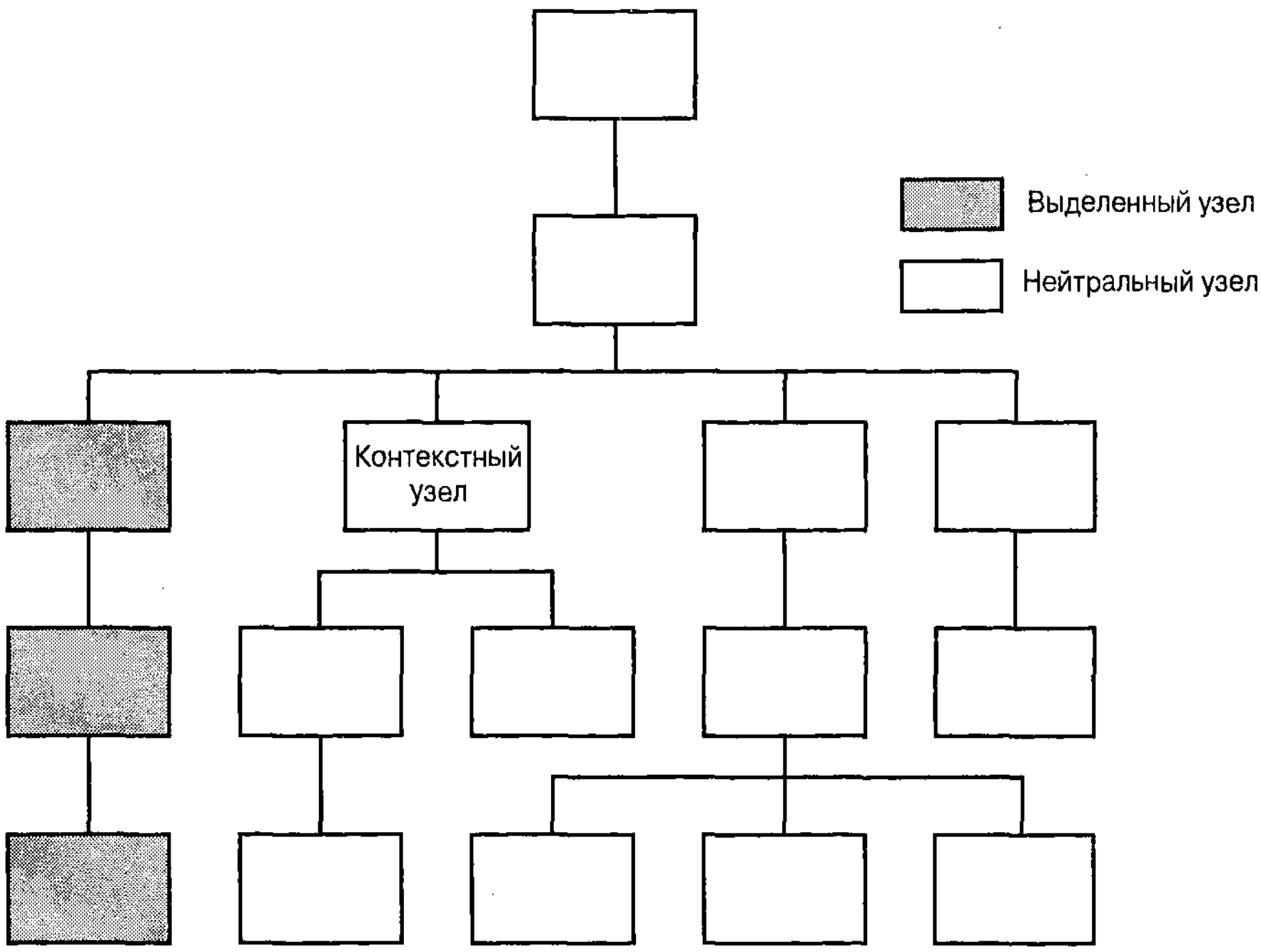


Рис. 9.11. Дерево XPath, в котором показаны отношения предыдущих узлов

Обратите внимание на то, что к предыдущим относятся только те узлы, которые находятся на том же уровне или на более низких уровнях, но до контекстного узла. Узлы, которые находятся на более высоких уровнях, такие как родительские узлы и узлы-предки, предыдущими не считаются.

Предыдущие узлы одного уровня

Не удивительно, что существуют и отношения предыдуших узлов одного уровня, которые позволяют выбрать только узлы одного уровня по отношению к контекстному в соответствии с порядком документа.

Соответствующий пример приведен на рис. 9.12.

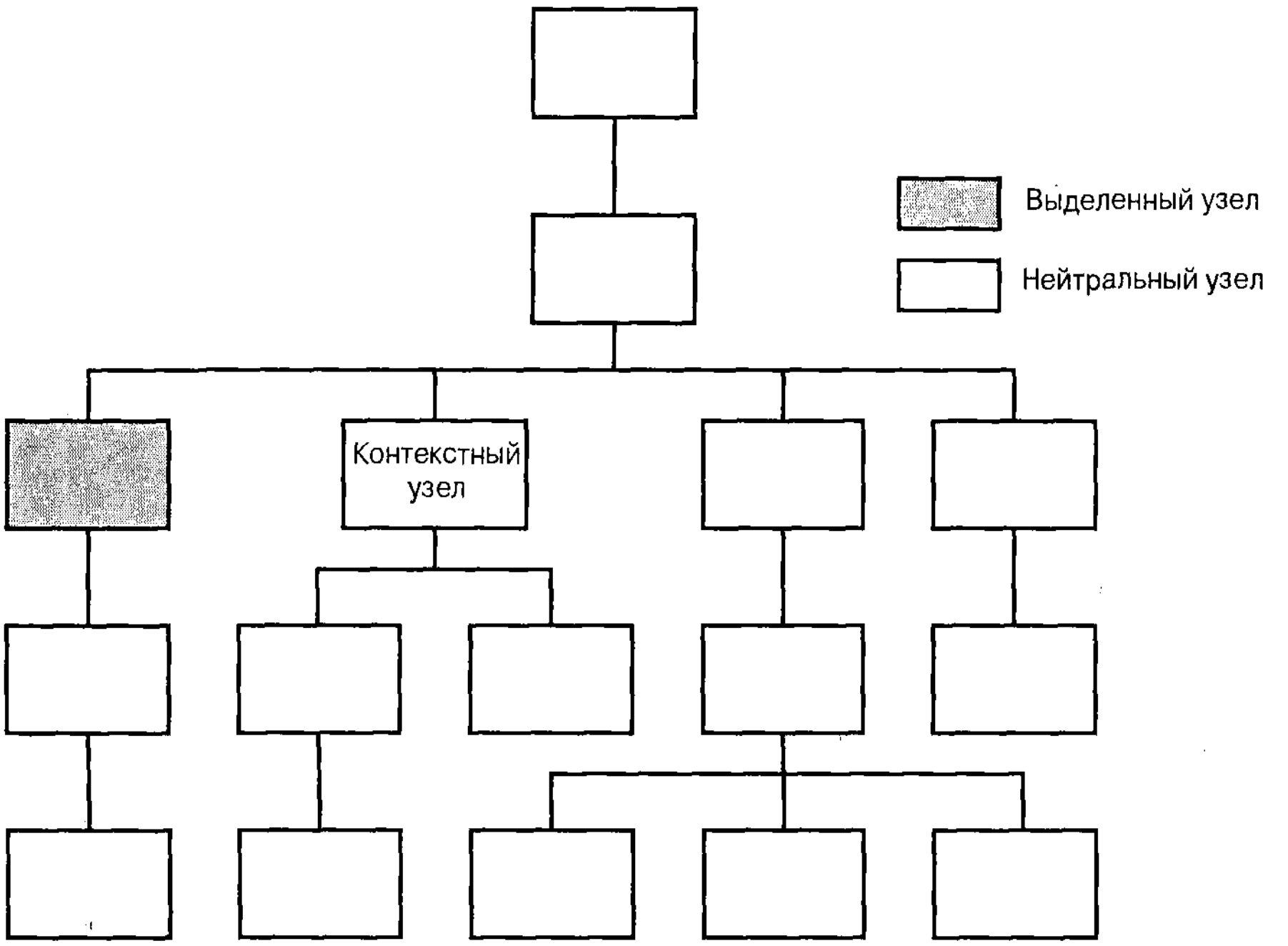


Рис. 9.12. Дерево XPath, в котором показаны отношения предыдущих узлов одного уровня

В этом разделе вы познакомились со стандартными типами отношений, существующими в XPath. Вы увидели, что выбирались узлы, находящиеся в том или ином положении по отношению к контекстному. Особенно это полезно при работе с очень сложным и большим документом. Например, предположим, что вы имеете дело с большим финансовым документом, созданным с использованием XML, содержащим различные налоги для разных товарных групп и регионов страны. Каждому региону соответствует несколько дочерних элементов, отражающих налоги для торговли бензином, алкоголем, сигаретами и т.д. Для обнаружения подобных дочерних узлов очень удобно использовать представления об отношениях.

Вы убедились в том, что XPath просматривает документ как в прямом, так и обратном порядке, что отражается на том, какие узлы будут считаться предыдущими, а какие — следующими. В следующем разделе вы научитесь создавать выражения XPath, отражающие те или иные отношения между узлами. Это позволит вам более тонко контролировать процесс выбора, а также обнаруживать определенные узлы.

Создание и проверка выражений XPath

Вы уже убедились в относительной природе XPath, а также различных отношениях, в которых находятся узлы. В настоящем разделе вы познакомитесь с созданием абсолютных выражений путей, которые позволяют выбирать определенные элементы, атрибуты или узлы других типов. Затем вы увидите, как можно объединить эти выражения с осями отношений, о которых я рассказывал раньше в настоящем уроке. Основная цель настоящего раздела — познакомиться с синтаксисом наиболее популярных выражений XPath, а не со всеми существующими вариантами.

Для выбора определенных узлов вы будете работать с настоящим документом XML, содержащим несколько вложенных элементов, примените к нему ряд самостоятельно созданных выражений XPath. Начните с создания простого документа XML. Представьте себе, что вы нашли в ящике своего рабочего стола несколько ручек и карандашей различного типа. Пусть твердость каких-то карандашей будет H, а других — HB. Вы также найдете несколько коробок, достаточно больших для того, чтобы в них поместилось по несколько карандашей. Все эти отношения формируют иерархию документа XML, содержащего дочерние и родительские узлы, а также узлы-предки и узлы-потомки. Создайте документ, представленный в листинге 9.3, сохранив его под именем `drawer.xml`.

Листинг 9.3. Содержимое доски для рисования в виде документа XML `drawer.xml`

```
1: <?xml version="1.0"?>
2: <!-- Листинг 9.3 -drawer01.xml -->
3:
4: <drawer>
5:       <pencil type="HB"/>
6:       <pen/>
7:       <pencil type="H"/>
8:       <pencil type="HB"/>
9:       <pencil type=" HB "/>
10:       <box>
11:            <pencil type="HB"/>
12:       </box>
13:       <tray>
14:            <box>
15:                 <pencil type="H"/>
16:                 <pencil type="HB"/>
17:            </box>
18:       </tray>
19:       <pen/>
20: </drawer>
```

В соответствии с нашим сценарием, элемент `drawer`, который начинается со строки 4, содержит несколько элементов `pencil`, `pen`, `box`, а также элемент `tray`. Элементы `pencil` содержат атрибуты `type` со значением `H` или `HB`.

Для проверки выражений XPath, которые вы создали, загрузите из Internet необходимый инструмент. Для проверки выражений XPath, а также представления полученного набора узлов замечательно подходит такой инструмент, как XPath Visualizer версии 1.3. Загрузите копию этого программного продукта с Web-узла `http://www.vbxml.com/xpathvisualizer/default.asp`. Вы сможете использовать это приложение для проверки всех примеров, представленных в настоящем разделе. Программа XPath Visualizer разрабатывалась как инструмент обучения XPath; однако она замечательно подходит и в тех ситуациях, когда вам необходимо создать сложные выражения XPath для их дальнейшего использования в XSLT или XPointer.

Программа XPath Visualizer написана в виде коллекции простых HTML-страниц, содержащих сценарии JavaScript для манипулирования узлами, найденными с помощью XPath. Каскадная таблица стилей CSS и таблица стилей XML включены для отображения выбранных узлов различными цветами. На вашем компьютере должен быть установлен броузер Microsoft Internet Explorer версии 5 или более поздней, поскольку XPath Visualizer инициализирует объект MSXML, поставляемый вместе с броузером. С синтаксическим анализатором MSXML вы познакомились в уроке дня 3-го. На момент написания настоящей книги последней версией синтаксического анализатора MSXML была версия 4.0, которую вы сможете найти по адресу `http://msdn.microsoft.com/downloads`. Для запуска XPath Visualizer достаточно MSXML версии 2.0, поэтому, если у вас есть Microsoft Internet Explorer версии 5 или более поздней, то загружать MSXML версии 4.0 необязательно.

После загрузки файлов программы XPath Visualizer сохраните их в отдельной папке на жестком диске вашего компьютера. Для запуска приложения откройте с помощью Microsoft Internet Explorer файл `XPathMain.htm`. после загрузки страницы щелкните на кнопке Browse (Обзор) и найдите созданный раньше файл `drawer.xml`. Путь к файлу указан в качестве значения поля слева от кнопки. Щелкните на кнопке Process File (Обработать файл). Это должно привести к обновлению содержимого окна броузера, в котором теперь представлен листинг из файла `drawer.xml`, причем каждый из узлов будет выделен желтым цветом. Вы увидите, что, помимо выделения цветом каждого из узлов, документ XML выглядит так, как будто к нему применяли преобразования XSLT со стороны Internet Explorer. Причем выполняются не только эти преобразования, но и добавляются стили для выделения выбранных узлов XPath. Подробно с преобразованиями XSLT вы познакомитесь в уроке дня 16-го.

После щелчка на кнопке Process File (Обработать файл) в окне XPath Visualizer программа автоматически добавит выражение `//*` в поле, расположенное под именем файла. Это поле называется полем выражения XPath. В частности, указанное выражение XPath позволяет выбрать все узлы в дереве. Подробно об использовании подстановочного знака "`*`" вы узнаете дальше в настоящем разделе. При обработке этого выражения приложение выделит все узлы в дереве. Вы сможете добавлять все рассматриваемые дальше выражения прямо в поле выражения XPath, чтобы выбрать определенные узлы в документе XML. Окно броузера Microsoft Internet Explorer, в котором загружена программа XPath Visualizer для обработки документа `drawer.xml`, приведено на рис. 9.13.

Выбор узлов по имени с помощью косой

В последующих разделах вы изучите синтаксис самых популярных выражений XPath. В каждом случае вы сможете ввести строку в поле выражения XPath программы XPath Visualizer для просмотра выбранных узлов.

Косая (/) используется XPath для ссылки на корневой узел в дереве XPath. Когда вы начнете создавать шаблоны XSLT с помощью выражений XPath в уроке дня 16-го, вы будете использовать косую во многих своих программах. В настоящей главе вы будете использовать этот знак для ссылки на корневой узел при обнаружении необходимых узлов на более нижних уровнях.

Выбор корневого элемента

Основной синтаксис выражения XPath очень похож на систему адресации, принятую в различных файловых системах. В нашем примере вы введете в поле выражения XPath программы XPath Visualizer строку `/drawer` для выбора корневого элемента в документе `drawer.xml`. После того как вы ввели эту строку, щелкните на кнопке Select Nodes (Выбрать узлы), чтобы выбрать корневой элемент `drawer`, выделенный желтым. Прило-

жение также сообщит о количестве совпадений, обнаруженных им в соответствии с введенным вами выражением. В данном случае оно сообщит вам только об одном совпадении корневого элемента в экземпляре документа XML. И это совершенно логично — ведь документ XML может содержать только один корневой элемент.

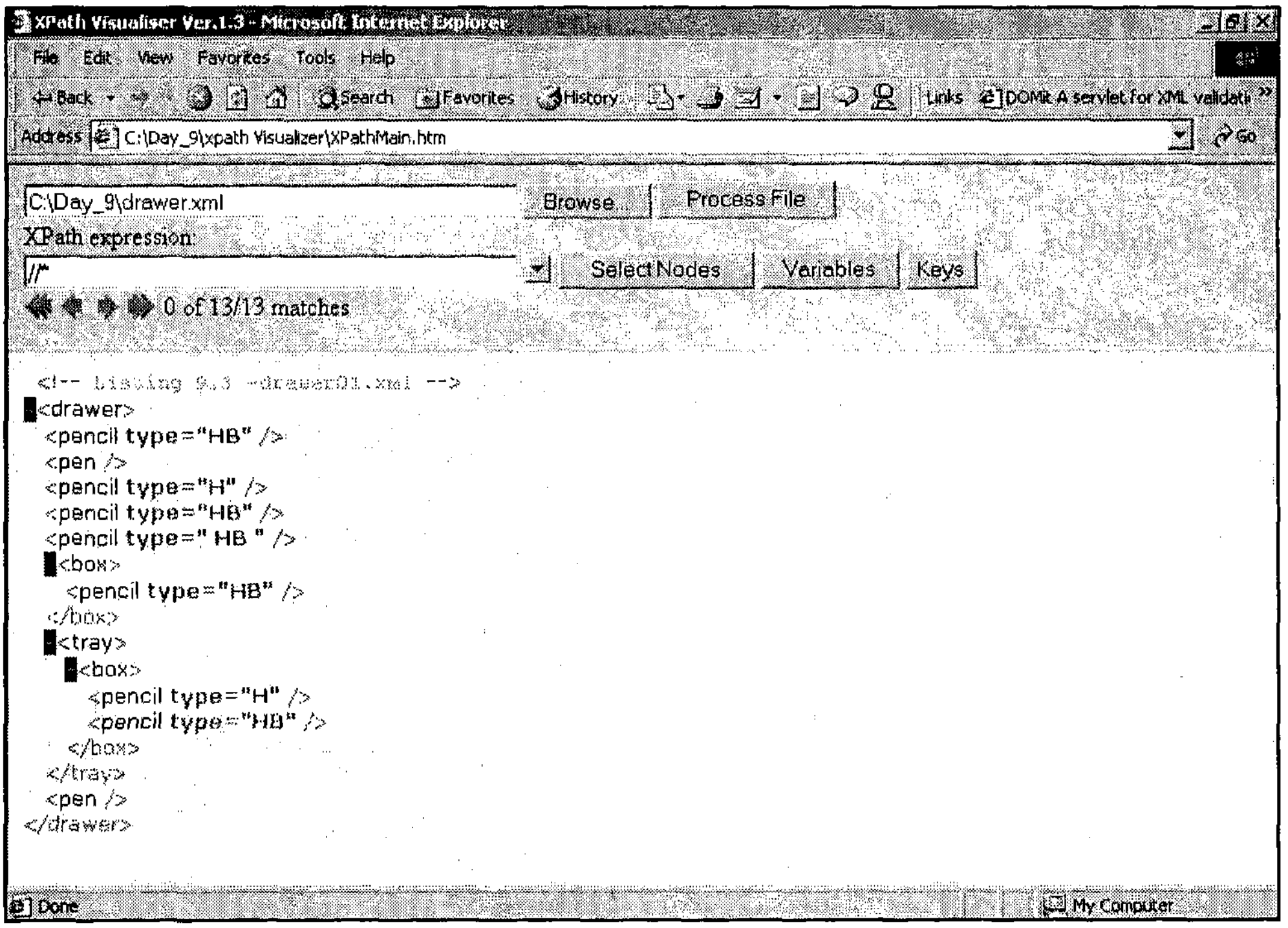


Рис. 9.13. Приложение XPath Visualizer отображает все узлы в документе `drawer.xml` *желтым цветом*

В соответствии с введенным вами выражением вы выбираете дочерний элемент `drawer` корневого узла. В результате мы имеем дело с выбором известного корневого элемента по имени.

В качестве проверки попробуйте ввести строку `/pen` и щелкнуть на кнопке Select Nodes (Выбрать узлы). После вычисления выражения XPath не будет обнаружено ни одного совпадения; приложение сообщит вам о нулевом числе совпадений, и не один из узлов в дереве не выделяется цветом. В следующем примере вы увидите синтаксис выражения XPath для поиска узлов `pen` в рассматриваемом нами экземпляре документа XML.

Выбор дочернего элемента корневого элемента

Поскольку XPath необходимо знать определенное отношение или ветвь для поиска узлов, вам необходимо создать выражение, которое будет содержать корневой узел, а после него будет указан корневой элемент, затем — целевой элемент. Другими словами, вам необходимо ввести строку `/drawer/pen`, если вы пытаетесь обнаружить все элементы `pen`. Попробуйте выполнить эти действия и посмотрите на полученные результаты. Вы должны увидеть два элемента `pen`, выделенных желтым цветом, а счетчик программы должен сообщить вам о выборе двух совпадающих узлов. Выражение интерпретируется следующим образом: "Выберите все элементы `pen`, дочерние по отношению к корневому элементу `drawer`".

Попробуйте выбрать все элементы `pencil` в экземпляре документа, используя тот же синтаксис. Введите следующее выражение XPath: `/drawer/pencil`. После вычисления этого выражения приложение сообщит вам о четырех совпадениях, а также выделит желтым цветом четыре первых элемента `pencil`. Оставшиеся два элемента `pencil` не были выбраны потому, что они не являются дочерними по отношению к корневому элементу. Введенное вами выражение выбирает только дочерние по отношению к корневому элементы, но не другие его элементы-потомки. О выборе элементов-потомков мы поговорим дальше.

Выбор элементов по известному направлению

Следуя той же логике, с которой вы уже познакомились, вы должны иметь возможность выбрать все элементы `pencil`, дочерние по отношению к определенному элементу `box`, который является дочерним по отношению к элементу `tray` в рассматриваемом нами экземпляре документа. Для того чтобы это сделать, постройте выражение XPath, которое будет следовать по документу в определенном направлении до достижения необходимого элемента. Введите следующее выражение XPath: `/drawer/tray/box/pencil`, после чего щелкните на кнопке Select Nodes (Выбрать узлы).

В результате будут выбраны два элемента `pencil`, дочерние по отношению к определенному элементу `box`, который является дочерним по отношению к элементу `tray`.

Выбор всех элементов по имени

Предположим, что вам действительно надо выбрать все элементы `pencil`, независимо от их расположения. Для выполнения этой задачи XPath позволяет использовать двойную косую (//). Этот знак интерпретируется следующим образом: "Выбери все элементы, удовлетворяющие определенному критерию". Для выбора всех элементов `pencil` введите выражение `//pencil` и исследуйте полученный результат. XPath обнаружит семь элементов `pencil`, независимо от их предков.

Выбор всех элементов по их предкам

Вы можете использовать оператор в виде двойной косой вместе с отношением родитель –дочерний для выбора всех дочерних элементов определенного родительского элемента. В данном случае порядок следования элементов оказывается очень важным, поэтому в выражении родительский документ должен обязательно предшествовать дочерним. Попробуйте создать соответствующее выражение, интерпретируемое так: "Выбери все элементы `pencil`, дочерние по отношению к элементам `box`.". Если вы изучите экземпляр документа XML, вы убедитесь в том, что в нем существует три подобных элемента `pencil`. Первый из них содержится в строке 11 листинга 9.3 ; два остальных — в строках 15 и 16 соответственно.

Выражение XPath для выбора этих элементов выглядит так: `//box/pencil`. Как уже отмечалветвь раньше, оно означает "Выбери все элементы `pencil`, дочерние по отношению к элементам `box`".

Использование подстановочных символов

Подстановочные символы в XPath указываются с помощью звездочки (*). Этот знак действует практически так же, как при поиске файлов. Для того чтобы разобраться, каким образом подстановочные символы можно объединить с другими выражениями, в табл. 9.2 приведено несколько примеров, касающихся документа `drawer.xml`. Попробуйте применить эти примеры в XPath Visualizer, а также поэкспериментируйте с созданием собственных выражений.

Таблица 9.2. Примеры использования подстановочных символов

Пример выражения XPath	*Интерпретация*	*Результат*
`/drawer/*`	Выбор всех элементов, дочерних по отношению к элементу `drawer` (или его потомками)	Возвращает все дочерние по отношению к элементу `drawer` элементы (общее количество — 8)
`/*/*/*/pencil`	Выбор всех элементов `pencil` третьего уровня, независимо от их элементов-предков	Возвращает два элемента `pencil`, которые содержатся в элементах `box`, `tray` корневого элемента
`//*`	Выбор всех элементов-потомков корневого элемента	Возвращает все элементы в экземпляре документа
`/*`	Выбор всех дочерних элементов корневого элемента	Возвращает только корневой элемент

Выбор определенных элементов из полученного набора узлов

В XPath квадратные скобки используются для включения в выражение XPath определенных формулы или функции. Например, предположим, что вам необходимо обнаружить третий элемент `pencil`, дочерний по отношению к корневому элементу. Вы можете использовать квадратные скобки для определения третьего элемента согласно порядку документа. Синтаксис подобного выражения приведен ниже:

```
Путь[выражение или функция]
```

Для нахождения третьего элемента `pencil` по отношению к корневому выражение XPath примет следующий вид:

```
/drawer/pencil[3]
```

Попробуйте ввести это выражение, экспериментируя с различными числовыми значениями в квадратных скобках, а также изменяя путь перед ними.

Функцию `last()` можно использовать в выражении в качестве счетчика для выбора последнего узла в соответствии с указанными условиями поиска. Таким образом, для выбора последнего элемента `pen`, содержащегося в корневом элементе, вы можете использовать выражение `/drawer/pen[last()]`.

Элемент `pencil` встречается в документе несколько раз. Каким образом можно выбрать последний элемент `pencil` в каждом из родительских элементов, в которых он содержится? Для решения этой задачи вам необходимо выбирать элементы независимо от их предков. Для этого можно использовать двойную косую. После этого вам следует указать интересующие вас элементы, в данном случае элементы `pencil`. И наконец, вам необходимо выбрать последний элемент из каждого найденного набора, используя функцию `last()`. Когда вы объедините все это, выражение примет следующий вид:

```
//pencil[last()]
```

В результате будет выбран последний элемент `pencil` в каждом из родительских элементов, в которых он содержится. Таких элементов три, и содержатся они в строках 9, 11 и 16.

Выбор атрибутов

В документе `drawer.xml` элементы `pencil` обладают атрибутами `type`. Вы можете использовать целый ряд подходов в XPath для выбора атрибутов и их значений, а также комбинировать функции и синтаксические правила, с которыми уже познакоми-

лись, для сужения области поиска. В выражении XPath для указания включения именно атрибута, а не элемента, используется специальный знак атрибута — `@`.

Для выбора всех атрибутов в экземпляре вы можете объединить оператор `//` со знаком атрибута и подстановочным символом: `//@*`. Попробуйте ввести это выражение в окне программы XPath Visualizer; тогда должны возвратиться все семь атрибутов, которые встречаются в документе XML. Вы также можете выбрать атрибуты по имени, для чего следует указать их имя. Например, для выбора всех атрибутов `type` введите `//@type` в поле выражения XPath. Конечно же, в связи с тем, что документ `drawer.xml` содержит только атрибуты `type`, вы также получите семь атрибутов, как и при выборе всех атрибутов в документе.

В XPath используется еще одна функция — `not()`, которая позволяет исключать определенные узлы. Например, если вам необходимо выбрать все элементы в документе `drawer.xml`, у которых нет атрибутов, введите следующее выражение XPath.

```
//*[not(@*)]
```

Это выражение интерпретируется следующим образом: "Выбери все элементы, которые являются потомками корневого элемента и не содержат атрибуты". В документе `drawer.xml` это приведет к выбору корневого элемента `drawer`, а также элементов `pen`, `tray` и `box`.

XPath также позволяет использовать в выражениях в качестве критериев выбора значения атрибутов точно так же, как при использовании инструкции SQL `WHERE`. Например, для выбора всех элементов `pencil`, значение атрибута `type` которых равно `HB`, введите `//pencil[@type="HB"]`. Это выражение возвратит вам четыре совпадения, если созданный вами документ `drawer.xml` полностью соответствует варианту, предложенному в листинге 9.3. Давайте изучим строки 8 и 9 этого листинга:

```
8:      <pencil type="HB"/>
9:      <pencil type=" HB "/>
```

Вы видите, что две эти строки отличаются между собой. В значении атрибута в строке 9 есть пробел. Таким образом, выражение XPath возвратит только четыре совпадения, поскольку значение атрибута из строки 9 не соответствует указанному критерию. Поскольку XPath — достаточно мощный язык, он предоставляет вам возможность опускать лишние пробелы в значения в значениях атрибутов. Выражение `//pencil[normalize-space(@type)="HB"]` возвратит пять узлов, содержащих атрибуты, поскольку в результате нормировки значения `"HB"` и `" HB"` стали равны.

Дополнительные функции XPath

В табл. 9.3 приведены примеры нескольких дополнительных функций, которые можно использовать в выражениях XPath. В каждом случае приводится соответствующий пример, однако вы можете подставлять другие имена элементов, цифры или буквы для создания собственных выражений.

Таблица 9.3. Дополнительные функции в выражениях XPath

Пример выражения XPath	*Интерпретация*	*Результат*
`//*[count(pencil)=2]`	Выбор всех элементов, содержащих ровно два дочерних элемента `pencil`	Возвращает второй элемент `box` в документе `drawer.xml`
`//*[count(*)=1]`	Выбор всех элементов, содержащих только один дочерний элемент	Возвращает элементы `box` и `tray`

Пример выражения XPath	Интерпретация	Результат
`//*[name()="pen"]`	Выбор всех элементов `pen`	Возвращает два элемента `pen`. Эквивалентно выражению `//pen`
`//*[start-with(name(),"p")]`	Выбор всех элементов, имя которых начинается с буквы `p`	Возвращает все элементы `pen` и `pencil`, т. е. всего девять элементов
`//*[contains(name(),"x")]`	Выбор всех элементов, в имени которых встречается буква `x`	Возвращает два экземпляра элемента `pen`
`//*[string-length(name())=3]`	Выбор всех элементов, в имени которых содержится ровно три буквы	Возвращает все элементы `pen` и `box`, т. е. всего четыре элемента
`//*[string-length(name())>3]`	Выбор всех элементов, в имени которых содержится больше трех букв	Возвращает девять элементов, в именах которых содержится больше трех букв

Комбинирование осей XPath

Вы можете объединить несколько выражений XPath, используя оператор `|`, чтобы отделять их. Например, если вы хотите выбрать все элементы `box` и все элементы `pen`, создайте выражение XPath, которое выглядит следующим образом: `//box|//pen`. Количество объединений выражений неограниченно; просто не забывайте, что их действие кумулятивно.

Явное объявление относительных осей

Вы уже знаете, что можно выбрать корневой элемент, выбрав дочерний элемент корневого узла. Синтаксис соответствующего выражения следующий:

```
/имя_элемента
```

На самом деле, это краткая форма полностью определенного выражения, которое выглядит следующим образом:

```
/child::имя_элемента
```

Таким образом, полностью определенный вариант выражения, необходимого для возвращения корневого элемента документа `drawer.xml`, имеет вид `/child::drawer`. Ветвь `child` является ветвью по умолчанию, поэтому ее можно опустить, что приведет к краткой форме выражения. При необходимости в одном выражении XPath допускается совмещение краткой и полной форм. Таким образом, выражение `/child::drawer/pencil` возвратит все дочерние элементы `pencil` элемента `drawer`, который, в свою очередь, является дочерним по отношению к корневому узлу.

Другие отношения элементов, с которыми вы познакомились в настоящей главе, также можно включить в выражения XPath, указав соответствующие оси, разделяя их двумя двоеточиями. Например, для выбора всех потомков элемента `tray` вы можете использовать следующее выражение:

```
<drawer/tray/descendent::*
```

Это выражение указывает на необходимость выбора всех потомков элемента `tray`, дочернего по отношению к элементу `drawer`, который сам является дочерним элементом корневого узла.

Предположим, вам необходимы только те элементы `pencil`, среди предков которых есть элемент `tray`. Для этого вам достаточно ввести следующее выражение: `//tray/descendent::pencil`.

Выражение `//box/following-sibling::*` позволяет обнаружить элемент `tray`, а также следующий элемент `pen` в порядке документа; кроме того, в документе существуют и следующие элементы одного уровня с элементом `box`. Например, выражение `//box/following::*` возвращает все узлы после контекстного узла (`box`) согласно порядку документа.

Выражения предоставляют и целый ряд других возможностей, некоторые из которых достаточно сложны. Например, выражение `//pencil[position()=floor(last()div 2 +0,5) position()=ceiling(last()div 2 +0,5)]` обнаружит только средние элементы `pencil` в образце документа. Однако эти выражения выходят за рамки рассмотрения настоящего урока; с ними вы познакомитесь несколько позже.

Резюме

Сегодня вы познакомились с некоторыми основными концепциями языка XPath (XML Path Language). XPath — это язык выражений, а не диалект XML, который разрабатывался как удобный и эффективный инструмент. Он используется для адресации частей документа XML, называемых *узлами*. Подробно с XPath вы познакомитесь при изучении технологий, с которыми он должен взаимодействовать, такими как XPointer и XSLT. Более подробно с XPointer вы познакомитесь в уроке дня 11-го, а с XSLT — в уроке дня 16-го. В настоящем уроке вы познакомились с семью видами узлов, образующих дерево XPath: корень, атрибут, пространство имен, инструкция обработки, текст и комментарий. Вы узнали, каким образом все эти элементы соотносятся с корневым узлом, а также вы узнали, что корневой узел — это не одно и то же, что корневой элемент в документе XML. Вы познакомились с 11 видами отношений элементов, предназначенных для выбора узлов и их наборов при использовании XPath. После этого вы создали и проверили выражения XPath для выбора различных комбинаций элементов и атрибутов, а также познакомились с использованием функций и подстановочных символов.

Вопросы и ответы

Какова природа объекта данных, полученного после использования выражений XPath?

При вычислении выражений XPath полученный объект данных может быть отнесен к одной из следующих категорий.

- **Узел.** Выбирается один узел.
- **Набор узлов.** Выбирается набор узлов с некоторыми общими характеристиками.
- **Логическое значение.** Выражение приводит к результату `true` или `false`.
- **Строка.** В результате применения выражения получается строка, зависящая от значений ряда узлов.
- **Число.** В результате применения выражения получается число с плавающей запятой.

Какие типы узлов можно обнаружить с помощью XPath?

XPath позволяет находит следующие семь типов узлов: корень, атрибут, пространство имен, инструкция обработки, текст и комментарий.

Для чего в XPath используется концепция порядка документа?

Выражения XPath приказывают обработчику просматривать дерево узлов в том порядке, в котором узлы следуют в документе. Таким образом оси отношений означают расположение узлов в дереве XPath по отношению к контекстному узлу, начальной точке или направлению, указанному в выражении XPath. Предыдущие узлы — это узлы, которые следуют до контекстного узла в порядке, определяемом структурой документа. Следующие узлы — это узлы, которые идут после контекстного узла в порядке, определяемом структурой документа.

Упражнение

Проверьте свои знания, полученные в результате изучения настоящего урока. Ответ вы найдете в Приложении А "Ответы на вопросы".

Обработайте файл `cd.xml` с помощью программы XPath Visualizer и создайте выражения для выбора следующих узлов.

- Все элементы, дочерние по отношению к корневому узлу.
- Все элементы `tracks`, значение атрибута `total` которых равно 16.
- Все элементы, в именах которых содержится буква `i`.
- Все элементы, в именах которых содержится больше 11 букв.
- Все элементы одного уровня по отношению к первому элементу `cd`.

День 10-й

Язык XLink

Популярность Web отчасти связана с широкой поддержкой гиперссылок, реализуемых с помощью дескриптора анкера (<A>) языка HTML. Гиперссылки позволяют эффективно объединять содержимое Web-страниц. Однако ненаправленных, простых ссылок, характерных для современной Web, совершенно недостаточно для удовлетворения растущих потребностей мира XML. Рекомендация консорциума *W3C*, касающаяся ссылок в XML, называется XML Link Language (XLink) и датируется 27 июля 2001 года.

Сегодня рассматриваются следующие темы.

- Связывание ресурсов в XML с помощью Xlink.
- Различие между ссылками XLink и HTML
- Связывание элементов с помощью ссылок и описание удаленных ресурсов.
- Значение и синтаксис расширенных ссылок.
- Различные виды ссылок.

Связывание гипертекста

Гипертекст — это термин, введенный еще в начале 1960-х годов Тэдом Нельсоном, на то время аспирантом Гарвардского университета, работающим на проектом, в дальнейшем получившим название *Xanadu*. Ранние упоминания о концепции понятной для компьютера текстовой информации, не последовательной, но организованной таким образом, чтобы связанные элементы информации объединялись, можно найти в работе Ванневара Баша (Vannevar Bush) под названием "As We May Think" ("Как мы можем думать"), опубликованной еще в 1945 году (`http://www.theathlantic.com/unbound/flashbks/computer/bushf.htm`). Баш назвал свою систему Memex, что является сокращением от *microfilm indexing*. Нельсон первый уже полностью описал свою концепцию компьютерных гиперссылок на национальной конференции, проводимой ассоциацией по вычислительной технике ACM (Association for Computing Machinery) в 1965 году.

Фундаментальная идея гипертекста заключается в том, что документы, сохраненные в электронном виде, можно представить в виде коллекции *узлов*. В свою очередь, узлы могут содержать перекрестные ссылки, или *ссылки*, которые (конечно, с помощью специальной программы, например, броузера) позволяли бы читателю легко переходить от одного элемента к другому. Именно для этого и используется гипертекст в Web. В соответствии с консорциумом *W3C* (`http://www.w3.org/Terms.html`) расширением гипертекста можно считать звук, видео и графические изображения (все

это также называется *гипермедиа*), однако очень часто любые мультимедийные дополнения также называют просто гипертекстом.

Действительно, с момента изложения самой концепции, Web никогда не представлялась набором изолированных страниц информации. Вместо этого предоставлялись специальные механизмы для указания отношений между страницами, для активизации связующих их ссылок. Наиболее очевидный пример из HTML — это анкер, т.е. дескриптор, позволяющий программисту определить ссылку, которая активизируется после щелчка на ней мышью в окне броузера. В настоящем уроке вы познакомитесь с рядом ограничений, связанных с использованием анкерных ссылок HTML, а также узнаете, каким образом XLink позволяет преодолеть ограничения, характерные для "старого" подхода.

XLink использует стандартизированный словарь XML, который легко добавить к экземпляру документа XML для описания простых или расширяемых ссылок на Web-ресурсы. Основное назначение XLink — расширение концепции ссылок, которые раньше были ограничены однонаправленными анкерными ссылками в HTML или ссылками ID/IDREF для внутреннего связывания элементов в документе XML. Вы уже, наверно, знакомы с ссылками HTML, но все равно в следующем разделе я предлагаю их краткий обзор. О расширении возможностей ссылок ID/IDREF мы говорили в уроках дней 4-го, 5-го и 6-го.

Ссылки HTML

Используя анкерный дескриптор (<A>), HTML позволяет внедрение гипертекстовых ссылок в документы. После активизации анкерный дескриптор, связанный, например, с внешним документом, инициирует процесс загрузки содержания этого документа. Ссылка HTML функционирует в однонаправленном режиме, поскольку синтаксис подобных ссылок не разрешает, например, вернуться к исходному документу. Эта проблема решается, например, с помощью кнопки Back (Назад) на панели инструментов большинства современных броузеров. Целевой объект, на который указывает ссылка HTML, выражается как значение атрибута HREF дескриптора HTML <A>. Дескриптор HTML <A> содержит текст или другой объект, выступающий в роли ссылки на странице. Ссылка обычно активизируется в тот момент, когда пользователь, просматривающий страницу, наводит указатель на объект ссылки и щелкает на нем. Простейшая форма анкерного дескриптора HTML имеет следующий вид:

```
<A HREF="[URL]">[текст|другой элемент HTML]</A>
```

Дескриптор HTML <A> в качестве значения атрибута HREF принимает любое выражение. Содержанием анкерного дескриптора может быть текст или другой элемент. Все, что содержится в анкерном дескрипторе, и является тем элементом, который приводит к активизации ссылки. В большинстве реализаций броузеров именно активатор ссылки позволяет пользователю инициировать функционирование ссылки после щелчка на ней мышью. Пример HTML-страницы с внедренной гиперссылкой приведен в листинге 10.1.

Листинг 10.1. HTML-страница с внедренной гиперссылкой HTMLlink01.html

```
1:  <!DOCTYPE XHTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
2:   "http://www.w3.org/TR/html4/loose.dtd">
3:
4:  <!--  Листинг 10.1 - HTMLlink01.html -->
5:
6:  <HTML>
7:
8:    <HEAD>
```

```
 9:      <TITLE>HTML Link Document</TITLE>
10:    </HEAD>
11:
12:    <BODY>
13:      <P>This is a simple HTML document with an embedded link to the
14:      <A HREF="http://architag.com">Architag</a> website.</P>
15:    </BODY>
16:
17: </HTML>
```

АНАЛИЗ В строках 1 и 2 объявляется, что настоящая HTML-страница соответствует стандарту HTML 4.01 и поддерживает переходы. В строке 6 показан корневой элемент HTML-страницы, который всегда называется `HTML`. В строках 8–10 объявляется элемент `HEAD`, содержащий элемент HTML `TITLE`, а также текстовые данные. Строки 12–15 содержат элемент `BODY`. Дочерний элемент `<P>` (строки 13–14) существует в рамках элемента `BODY`. В HTML элемент `<P>` соответствует абзацу. В строке 14

```
14:      <A HREF="http://architag.com">Architag</a> website.</P>
```

вы видите, что ссылка внедрена в строку текста в качестве дочернего элемента `<A>` по отношению к элементу `<P>`. Пользователь, просматривающий эту страницу с помощью соответствующего Web-броузера, сможет активизировать ссылку, чтобы загрузить страницу, находящуюся по адресу `http://architag.com`, подведя указатель мыши к слову `Architag` и щелкнув на нем.

На рис. 10.1 показана типичная ссылка, которая с помощью анкерного дескриптора HTML `<A>` устанавливает взаимосвязь между двумя Web-страницами. Исходный документ содержит элемент `<A>`, значение атрибута `HREF` которого равно действительному адресу URL. После активизации ссылки, как правило после щелчка мышью, загружается содержимое страницы с указанным адресом URL.

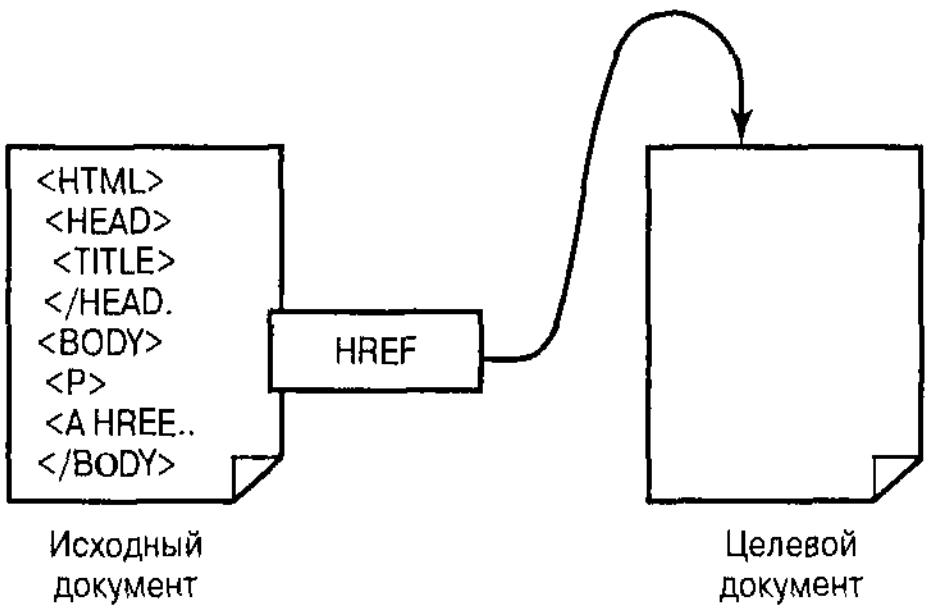


Рис. 10.1. Ссылка HTML, установленная с помощью внедренного дескриптора <A>

Как вы уже увидели, ссылки HTML представляют собой достаточно простые, однонаправленные механизмы. Несмотря ни на что, нельзя переоценить их вклад в распространение информации через Internet, что послужило серьезным толчком к популяризации World Wide Web. Простой анкерный дескриптор HTML можно использовать для вставки изображений, предоставления пользователю возможности перехода от одного документа к другому (как было показано раньше) или просто перехода к другой части того же документа.

Несколько специальных атрибутов анкерного дескриптора предоставляют достаточно ограниченные возможности контроля за ссылками. Например, атрибут `rel` используется для указания названия отношений между документом-источником и целевым документом. Соответствующий пример приведен ниже:

```
<A HREF=http://architag.com rel="goto_Arhitag">Architag</a> website.</P>
```

Атрибут `rel` определяет имя ссылки для работы программиста, возможно для улучшения документирования, и выполняет ряд других функций. Преимущественно атрибут `rel` используется в качестве метки. Атрибут `rev` позволяет описать обратную связь, описывая отношение целевого документа к документу-источнику. И опять, мы имеем дело со меткой, пример которой приведен ниже:

```
<A HREF=http://architag.com rel="goto_Arhitag" rev="example.doc">Architag
</a> website.</P>
```

Атрибут `rev` предоставляет метку для отношения обратной связи, символизирующей путь, который надо преодолеть, чтобы вернуться к документу-источнику, однако соответствующие функциональные возможности в современных реализациях броузеров отсутствуют.

Еще один связующий элемент в HTML реализуется с помощью дескриптора `<LINK>`. Этот дескриптор описывает расположение документа в иерархии, однако указанная ссылка не обрабатывается; обрабатываются только те ссылки, которые указаны с помощью анкерного дескриптора `<A>`. Ссылки, указанные дескриптором `<LINK>`, определяют документ для работы указанного процесса или агента пользователя. Например, дескриптор `<LINK>` используется для связывания каскадных таблиц стилей с HTML-страницей для обработки последней соответствующим образом.

Для связей HTML характерен целый ряд ограничений. Например, адреса URL могут указывать только на один документ. Для того чтобы указать на определенный раздел в целевом документе, потребуется вручную добавить именованный дескриптор перед тем, как ссылка активизируется. Если вам необходима ссылка на четвертое предложение в восьмом абзаце в третьем разделе документа, именованный дескриптор должен уже находиться в соответствующем документе. Если после этого вам потребуется ссылка на пятнадцатое предложение на три абзаца раньше, там также должен быть именованный дескриптор. Другими словами, этот метод никогда не сработает с документами, сохраненными на сервере без соответствующих именованных дескрипторов, или с теми, к которым у вас есть доступ с правом “только чтение”.

Даже несмотря на то, что кнопка Back (Назад) и средство History (Журнал) в большинстве современных броузеров позволяют вам возвращаться к посещенным раньше страницам, этот подход не всегда оказывается надежным. Ссылка на другую страницу всегда однонаправлена. Документ-источник содержит сведения о целевом документе, но не наоборот.

Существует возможность использовать ссылки HTML в документах XML. Вы сможете это сделать, воспользовавшись языком XSLT для преобразования экземпляра XML к HTML с целью отображения в окне броузера. Этот подход обеспечивает очень простое связывание; однако XLink предлагает гораздо более широкие возможности.

XLink расширяет возможности ссылок HTML

XLink — это рекомендация консорциума W3C для создания ссылок между документами с гораздо более широкими возможностями. XLink проектировали для работы с XML, чтобы достигнуть максимально возможного при использовании ссылок HTML, а также для поддержки многонаправленных ссылок. Это означает, что ссылки могут указывать не только в одном направлении. Используя XLink, вы сможете указать две однонаправленные ссылки между документами, многонаправленную ссылку или даже несколько подобных ссылок. В настоящем уроке вы узнаете, каким образом использовать XLink для реализации различных соединений между документами.

XLink реализован в форме атрибутов элементов. Таким образом, в экземпляре XML любой элемент может функционировать как ссылка, в отличие от HTML, в котором в этой роли может выступать только анкерный дескриптор <A>. Ссылки могут сохраняться как в экземпляре XML, так и в отдельном специальном файле ссылок.

При объединении XLink с XPointer полученная комбинация предоставляет еще больше возможностей. XPointer позволяет вам устанавливать ссылки на произвольные части в документе XML, избавляя от предварительного указания именованных анкерных дескрипторов в целевых документах. Подробно с языком XPointer вы познакомитесь в следующем уроке, завтра.

XLink можно использовать для связывания не только документов. Он устанавливает связи между записями базы данных, видео и аудиоданными и любыми другими объектами, которые считаются *ресурсами*. Например, предположим, что вы работаете в рекламном агентстве. Вам необходимо связать рекламный материал со всеми пресс-релизами, брошюрами и другими подобными документами. После этого вам понадобится связать рекламный материал с аудиоданными для распространения через Internet. Возможно, вы захотите связать рекламный материал с видео и аудиоданными для получения полноценной мультимедийной презентации. Затем, предположим, полученный результат нужно предоставить международной аудитории, а значит, видеоданные надо связать с аудиоданными на разных языках. Вы уже начинаете понимать, что все эти различные ресурсы можно связать с помощью ссылок. XLink как раз и позволяет вам представить все эти ресурсы в виде связанной коллекции.

XLink-совместимые приложения способны "перемещаться" по ссылкам для обработки ресурсов, независимо от того, локальные те или удаленные. Это означает, что вы можете совершенно спокойно работать с ресурсами, находящимися на жестком диске вашего компьютера, в локальной сети или в Internet. Эти ресурсы, в свою очередь, можно связать с другими ресурсами и т.д.

Ограничения XLink

Самое значительное ограничение XLink состоит а том, что он еще не может похвастаться широкой поддержкой со стороны компаний —разработчиков Web-броузеров. Недостаток поддержки частично связан с тем, что ссылки XLink выходят за функциональные возможности броузеров. Однако в настоящее время разрабатываются новые классы устройств и агенты пользователей, которые смогут задействовать сложные многонаправленные ссылки. В настоящее время такие броузеры, как Internet Explorer или Opera, не обеспечивают никакой поддержки XLink. Netscape версии 6.0 и броузер Mozilla предлагают ограниченную поддержку XLink.

Связующие элементы

В XML элемент, который использует атрибуты XLink для выражения ссылки и описания ее природы, называется *связующим элементом*. В отличие от HTML, где ссылка устанавливается с помощью дескриптора <LINK>, в XML в качестве связующего может выступать любой элемент. Элемент XML может считаться связующим при выполнении следующих двух требований.

- Элемент должен включать атрибуты XLink, начинающиеся с уникального префикса. По общему соглашению, чаще других используется префикс `XLink:`.
- Префикс должен быть связан с идентификатором URI пространства имен XLink `http://www.w3.org/1999/xlink`.



Несмотря на то, что вы можете выбрать любой префикс для связывания атрибутов XLink с пространством имен, хорошим тоном в программировании считается использование префикса `xlink:`.

В описании ссылки все атрибуты XLink выполняют строго определенную роль.

XLink определяет восемь атрибутов для описания связующего элемента. Эти атрибуты, принадлежащие к пространству имен XLink (`http://www.w3.org/1999/xlink`), добавляются к элементам документа XML. В табл. 10.1 перечислены все атрибуты связующих элементов XLink.

Таблица 10.1. Атрибуты связующих элементов XLink

Атрибут	*Описание*
`type`	Строка, определяющая тип ссылки
`href`	Любой действительный адрес URL, определяющий расположение целевого ресурса
`from`	Строка, объявляющая ресурс, исходный по отношению к `arc`
`to`	Строка, объявляющая ресурс, целевой по отношению к `arc`
`show`	Строка, объявляющая, каким образом надо получить ресурс
`actuate`	Строка, объявляющая, каким образом надо инициализировать ссылку
`role`	Строка, используемая приложением для описания функции содержания ссылки
`title`	Метка ссылки

Дальше вы увидите целый ряд примеров использования большей части этих атрибутов.

Связующие элементы XLink содержат атрибут `XLink:type`, который описывает природу ссылки. Связующие элементы бывают типов `simple` или `extended`. *Простая ссылка* (`xlink:type="simple"`) — это ссылка, которая связывает один ресурс с другим точно так же, как это делает ссылка HTML. *Расширенная ссылка* (`xlink:type="extended"`) — это ссылка, которая позволяет связать между собой несколько локальных или удаленных ресурсов. С примерами обеих ссылок вы познакомитесь дальше в настоящей главе. Атрибут `xlink:type` может принимать одно из следующих действительных значений: `locator`, `arc`, `resource`, `title` или `none`. Все эти значения атрибута `xlink:type` кратко рассматриваются дальше в настоящем уроке.

Значение атрибута `xlink:href` связующего элемента представляет собой адрес URL соответствующего ресурса. Это может быть как локальный, так и удаленный ресурс, поэтому в качестве значения атрибута задается любой действительный адрес URL.

Пример использования XLink

Вы уже достаточно хорошо познакомились с синтаксисом XLink, поэтому в состоянии создать простой связующий элемент XLink для языка разметки обработки сообщений, которые мы рассматривали в нескольких предыдущих уроках. На протяжении предыдущих девяти дней вы создали несколько документов XML для разметки сообщений. Данные записываются приложением, получающим сообщения, когда получатель отсутствует. Предположим, вам необходимо создать связь между полученным сообщением и удаленным ресурсом, который содержит сведения об отправителе сообщения. Например, вы получаете доступ к ресурсу, которые содержит такие идентифицирующие отправителя сведения, как имя, адрес электронной почты или номер телефона.

Предположим, вы создаете связь из элемента message (который, безусловно, является локальным ресурсом) языка разметки сообщений. Это простая ссылка на удаленный ресурс, адрес которого — http://www.architag.com/devan/ks.html. Вам необходимо использовать два атрибута, с которыми вы уже познакомились раньше в настоящем уроке:

```
xlink:type="simple"
xlink:href="http://www.architag.com/devan/ks.html">
```

Первый атрибут (xlink:type="simple") указывает, что атрибут type относится к пространству имен xlink, объявленному раньше в разметке документа или одном из его предков. Значение simple указывает на простую связь между ресурсами. Атрибут xlink:href определяет адрес целевого ресурса.

Все эти атрибуты помещаются в связующий элемент, а пространство имен xlink должно быть обязательно объявлено в экземпляре документа XML. Полный вариант экземпляра документа XML приведен в листинге 10.2.

Листинг 10.2. Пример простого связывания в XML `message01.xml`

```
1: <?xml version="1.0"?>
2: <!-- Листинг 10.2 - message01.xml -->
3:
4: <note xmlns:xlink="http://www.w3.org/1999/xlink">
5:  <message xlink:type="simple"
6:           xlink:href="http://www.architag.com/devan/ks.html">
7:                  Remember to buy milk on the way home from work
8:     </message>
9: </note>
```

АНАЛИЗ Пространство имен XML для XLink (http://www.w3.org/1999/xlink) объявляется в строке 4 с префиксом xlink. Атрибуты type и href указываются в локальном ресурсе (элемент message) для создания простой связи с удаленным ресурсом по адресу http://www.architag.com/devan/ks.html.

Обратили ли вы внимание на то, что у элемента message есть имя (message), которое описывает его содержание? В HTML связующий элемент описывает функцию, а не содержание.

Проверка документа

Поскольку все атрибуты указаны в документе XML, вам необходимо учесть их в схеме, если вы предполагаете проверять документ XML на соответствие стандартам. Если для проверки экземпляра документа используется схема DTD, вам нужен операнд #FIXED для указания обязательного значения атрибутов. Например, допускается только одно пространство имен, которое может использоваться вместе с XLink, поэтому оно должно быть строго указано в схеме DTD. Кроме того, вы можете зафиксировать и значения других атрибутов. В листинге 10.3 приведена схема DTD, которую вы сможете использовать для проверки документа XML из листинге 10.2. После связывания документа со схемой вы проверьте документ, воспользовавшись броузером Microsoft Internet Explorer версии 5 или более поздней.

Листинг 10.3. Схема DTD для проверки документа `message01.dtd`

```
1: <!-- Листинг 10.3 - message01.dtd -->
2:
3: <!ELEMENT note (message)>
4: <!ATTLIST note
```

```
5:  xmlns:xlink CDATA #FIXED "http://www.w3.org/1999/xlink"
6: >
7: <!ELEMENT message (#PCDATA)>
8: <!ATTLIST message
9:  xlink:type CDATA #FIXED "simple"
10: xlink:href CDATA #REQUIRED
11: >
```

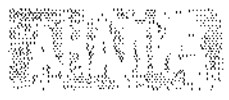


Элемент `note` содержит элемент `message`, а также атрибут `xmlns:xlink`, у которого единственное допустимое значение `http://www.w3.org/1999/xlink`. Элемент `message` содержит только текстовые данные, а также два атрибута — `xlink:type` и `xlink:href`. Атрибут `type` может иметь только одно значение: `simple`, а атрибут `href` — любое строковое значение.

В настоящее время при проверке документов вместе с XLink преимущественно используются схемы DTD; однако вы можете выполнить те же действия, используя другой язык описания схем, такой как XDR или XSD. Например, в листинге 10.4 приведена схема XSD для проверки рассматриваемого нами экземпляра документа.

Листинг 10.4. Схема XSD для проверки документа `message01.xsd`

```
1:  <?xml version="1.0" encoding="UTF-8"?>
     <!-- листинг 10.4 - message01.xsd -->
2:  <xsd:schema xmlns:xsd="http://www.w3.org/2000/10/XMLSchema">
3:   <xsd:element name="note">
4:       <xsd:complexType>
5:           <xsd:sequence>
6:               <xsd:element name="message" type="messageType"/>
7:           </xsd:sequence>
8:           <xsd:attribute name="xmlns:xlink" type="xsd:string"
             use="fixed" value="http://www.w3.org/1999/xlink"/>
9:       </xsd:complexType>
10: </xsd:element>
11: <xsd:complexType name="messageType">
12:     <xsd:simpleContent>
13:         <xsd:restriction base="xsd:string">
14:         <xsd:attribute name="xlink:type" type="xsd:string"
            use="fixed" value="simple"/>
15:             <xsd:attribute name="xlink:href" type="xsd:string"
                use="required"/>
16:         </xsd:restriction>
17:     </xsd:simpleContent>
18: </xsd:complexType>
19: </xsd:schema>
```



В строке 2 назначается стандартное пространство имен `http://www.w3.org/2000/10/XMLSchema` для экземпляра документа XSD. Объявления элемента `note` приведены в строках 3–10. Элемент `note` содержит элемент `message`, о чем говорит объявление в строке 6, а также атрибут `xmlns:xlink`, для которого единственное допустимое значение (`use="fixed"`) `http://www.w3.org/1999/xlink`. В соответствии с объявлением элемента `message` в строке 6 его дочерний элемент типа `complexType` называется `messageType`, что показано в строках 11–18. Атрибут `xlink:type` элемента `message` определен как атрибут типа `xsd:string` с фиксированным значением `simple`. Атрибут `xlink:href` может принимать любое значение типа `xsd:string`. Оба этих атрибута обязательны в проверяемом экземпляре документа XML.

Описательные атрибуты XLink

Атрибуты `xlink:title` и `xLink:role`, представленные в табл. 10.1, помогают дать описание ссылке. Атрибут `title` представляет собой описательный заголовок, данный ссылке. Атрибут `role` — это зависящий от конкретного приложения идентификатор URI, который используется для описания функции, выполняемой содержанием ссылки. В листинге 10.5 приведен тот же элемент `message`, но с добавленными атрибутами `xlink:title` и `xLink:role`.

Листинг 10.5. Простой пример Xlink `message02.xml`

```
1: <?xml version="1.0"?>
2: <!-- Листинг 10.5 - message02.xml -->
3:
4: <note xmlns:xlink="http://www.w3.org/1999/xlink">
5:  <message
6:       xlink:type="simple"
7:       xlink:href="http://www.architag.com/devan/ks.html"
8:       xlink:role="message"
9:       xlink:title="message processing example">
10:                 Remember to buy milk on the way home from work
11:     </message>
12: </note>
```

АНАЛИЗ В строках 8 и 9 показаны добавленные атрибуты `xlink:title` и `xLink:role`.

Поведение простых ссылок

Вы уже познакомились с простыми ссылками. Добавив атрибуты поведения ссылок, вы сможете контролировать переходы по ссылкам XLink-совместимого приложения. Точно так же вы можете проинструктировать приложение связать удаленные ресурсы с локальным несколькими заранее определенных способов.

Представление

Атрибут `xlink:show` используется для определения того, каким образом будет представлен удаленный ресурс XLink-совместимым обработчиком. Рассмотрите в качестве примера замены стиля анкерной ссылки HTML. После активизации ссылки пользователем щелчком мышью в нужном месте окна броузера, содержимое текущей страницы заменяется содержимым страницы с указанным адресом URL. В XLink атрибут `xlink:show` может принимать значение `replace`, которое приведет к тем же действиям при использовании XLink-совместимого броузера. Другими словами, локальный ресурс заменяется удаленным при активизации ссылки XLink с атрибутом `xlink:show="replace"`.

Возможные значения атрибута `xlink:show` перечислены в табл. 10.2.

Таблица 10.2. Возможные значения атрибута `xlink:show`

Значение	*Описание*
`replace`	Загружает удаленный ресурс в том же окне, в котором содержался локальный ресурс
`new`	Открывает новое окно приложения, в которое будет загружаться содержание удаленного ресурса

Значение	Описание
embed	Добавляет содержание удаленного ресурса к содержанию локального ресурса. Например, графический или звуковой файл можно внедрить в экземпляр документа
other	Определяет один из вызовов приложения. Например, вызов, который не ограничивается спецификацией XLink, но поддерживается приложением
none	Возможные действия не ограничиваются

Приведение в действие

Еще один атрибут поведения, `xlink:actuate`, используется для определения времени перехода к удаленному ресурсу. И опять, продолжая аналогию с HTML, давайте рассмотрим пример приведения в действия анкерной ссылки. Без использования дополнительных сценариев анкерная ссылка приводится в действие щелчком мышью в определенной части экрана, например на анкерном дескрипторе. В XLink для выполнения тех же действий предназначено значение `onRequest`. Однако атрибут `xlink:actuate` может принимать и несколько других значений, приведенных в табл. 10.3.

Таблица 10.3. Возможные значения атрибута `xlink:actuate`

Значение	Описание
onRequest	Подобно действию анкерной ссылки HTML; пользователь инициирует приложение, например щелкнув мышью, которое начинает обработку ссылки
onLoad	Загружает удаленный ресурс сразу после считывания приложением локального ресурса в память. XLink-совместимый броузер обнаруживает удаленный ресурс сразу после загрузки страницы со ссылкой
other	Определяет один из вызовов приложения. Например, вызов, который не ограничивается спецификацией XLink, но поддерживается приложением
none	Действия не ограничены спецификацией XLink

Исследование поведений XLink

На момент написания настоящей книги спецификацию XLink не поддерживали ни популярные броузеры, ни приложения общего характера. Частичную поддержку обеспечивал только броузер Netscape версии 6.0 (`http://home.netscape.com/browsers/index.html?cp=hop06d16`), который и использовался для проверки приведенных ниже нескольких примеров. Вы не сможете выполнять код XLink, используя Internet Explorer.

Измените документ `message02.xml`, добавив в него атрибут `xlink:actuate` со значением `onRequest`, а также атрибут `xlink:show` со значением `replace`. Новые атрибуты должны выглядеть следующим образом:

```
xlink:actuate="onRequest"
xlink:show="replace">
```

Сохраните измененный документ под именем `message03.xml`. Полный вариант документа представлен в листинге 10.6.

Листинг 10.6. Экземпляр XML, содержащий атрибуты поведения Xlink, `message03.xml`

```
1: <?xml version="1.0"?>
2: <!-- Листинг 10.6 - message03.xml -->
3:
4: <note xmlns:xlink="http://www.w3.org/1999/xlink">
5:  <message
6:       xlink:type="simple"
7:       xlink:href="http://www.architag.com/devan/ks.html"
8:       xlink:role="message"
9:       xlink:title="message processing example"
10:      xlink:actuate="onRequest"
11:      xlink:show="replace">
12:                  Remember to buy milk on the way home from work
13:     </message>
14: </note>
```



В строках 10 и 11 содержатся атрибуты `xlink:actuate` и `xlink:show`. Ссылка в действие приводится значение `onRequest`, т.е. пользователь должен инициировать соответствующую команду приложения — например, щелкнув мышью на гипертекстовой ссылке в окне броузера, — чтобы заставить обработчик обработать ссылку. После этого содержание локального ресурса заменяется содержанием удаленного ресурса.

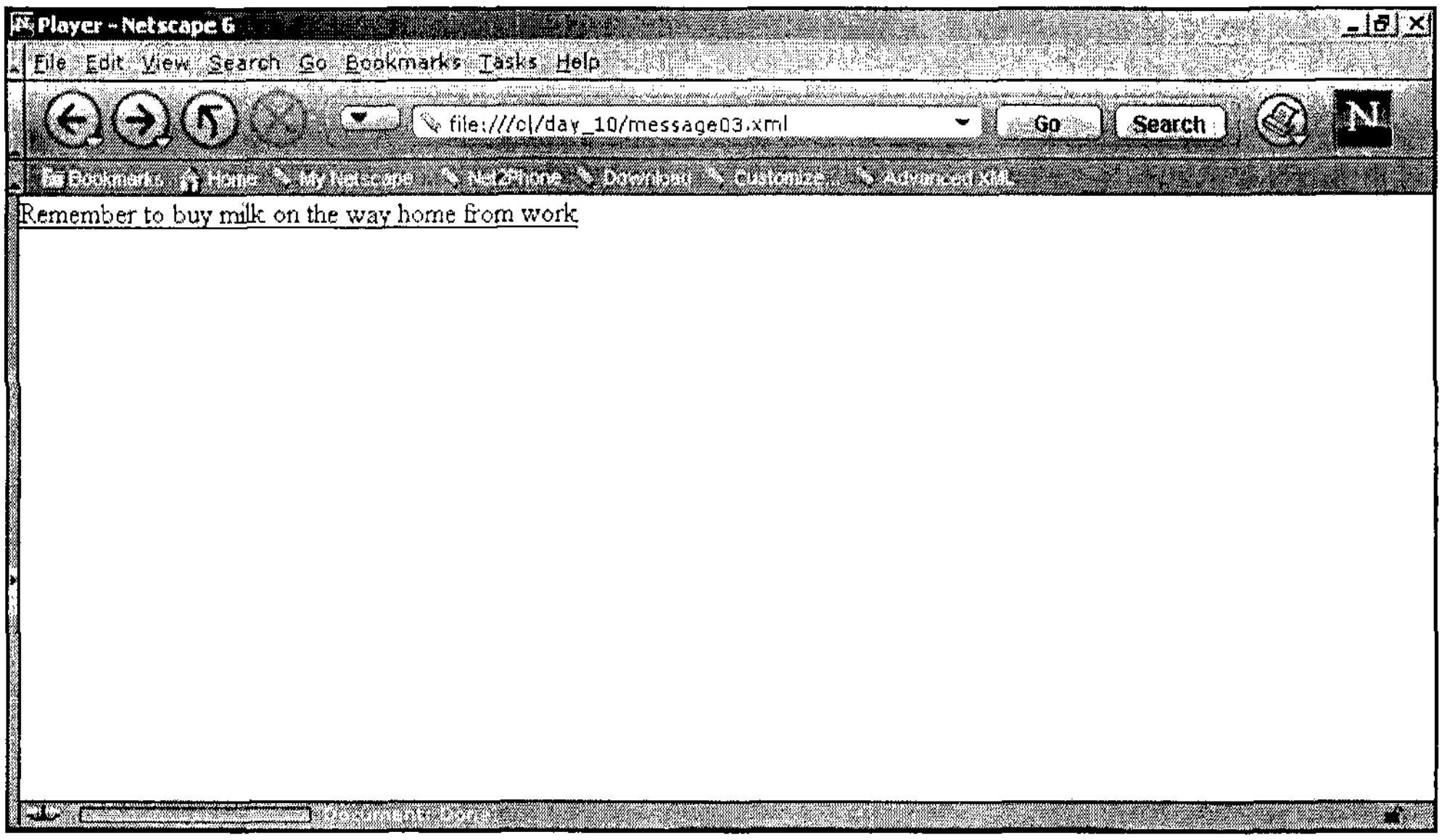


Рис. 10.2. Документ `message03.xml`, отображенный в окне Netscape 6

Загрузив полный код в окно Netscape 6, вы должны увидеть только текстовое содержание документа `message03.xml`. Оно подчеркнуто точно так же, как содержание элемента `<A>` в HTML. Обработчик XLink представляет гипертекстовые ссылки именно таким образом. Не забывайте о том, что другой обработчик может представлять ссылки совершенно другим способом. После щелчка на ссылке содержание локальной страницы (показано на рис. 10.2) должно замениться содержимым страницы с Web-адресом `http://www.architag.com/devan/ks.html`, как показано на рис. 10.3.

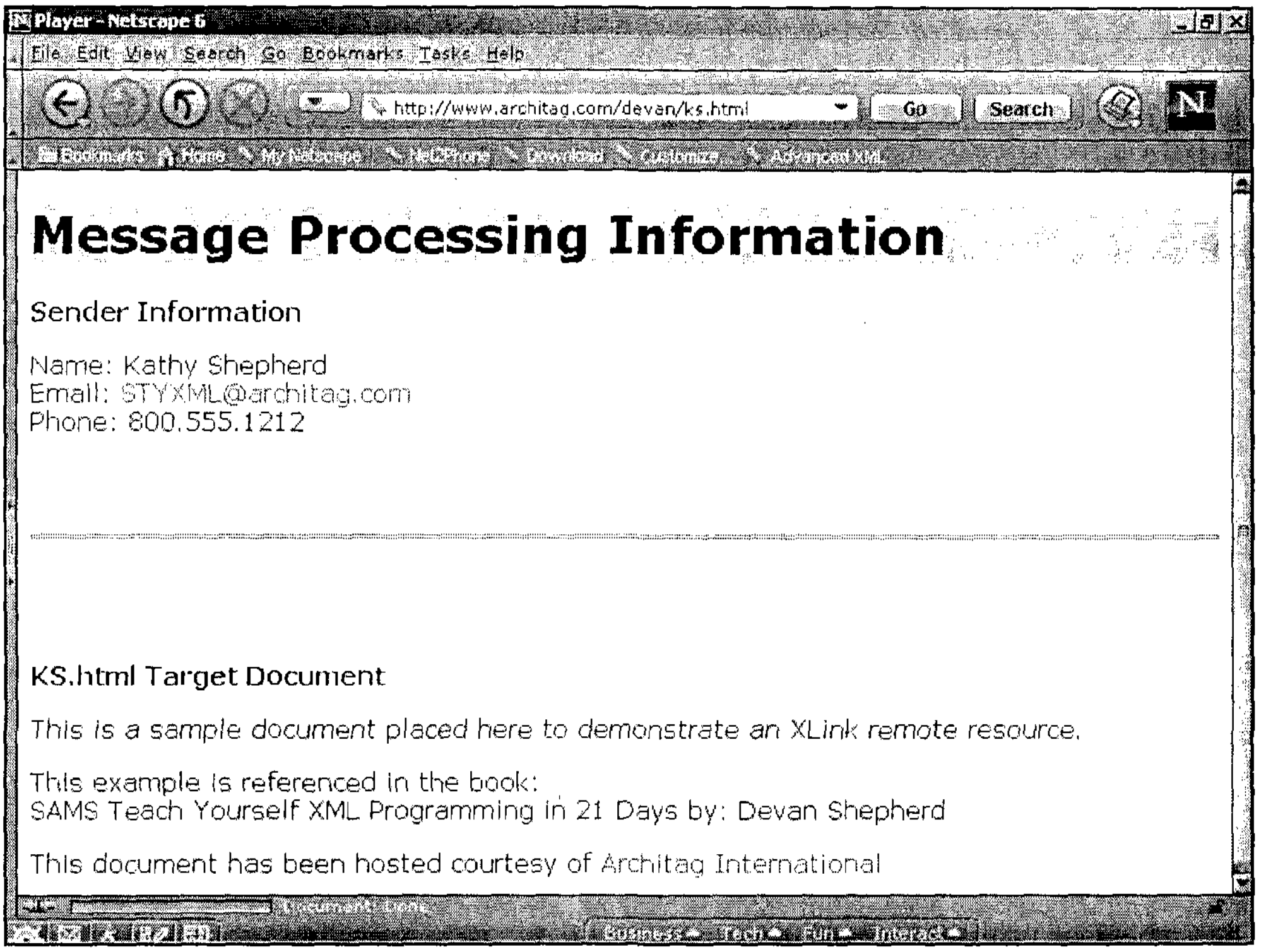


Рис. 10.3. Результат активизации ссылки XLink

Измените документ `message03.xml`, присвоив атрибуту `xlink:show` в строке 11 значение `new`. Сохраните измененный документ под именем `message04.xml` и активизируйте ссылку в окне Netscape 6. Броузер откроет новое окно, в котором представлен результат загрузки содержания удаленного ресурса. Вы также можете попробовать присвоить атрибуту `xlink:actuate` в строке 10 значение `onLoad`. Тогда ссылка приводится в действие слишком быстро для вас; вы даже не заметите того, что на загрузку содержания локального ресурса было затрачено какое-то время. Остальные значения этих двух атрибутов Netscape 6 не поддерживает.

Расширенные ссылки

Как уже отмечалось раньше, возможности XLink выходят за пределы простых ссылок. *Расширенные ссылки* обеспечивают несколько комбинаций локальных и удаленных ресурсов в нескольких направлениях. Расширенные ссылки состоят из набора ресурсов и соединений между ними. Локальные ресурсы — это часть связующего элемента, в то время как удаленные ресурсы выходят за его пределы. Удаленные ресурсы обычно относятся к другим документам, но они не обязательно должны быть удаленными по отношению к связующему элементу. Каждый из ресурсов может быть источником или целью ссылки, но может быть и тем, и другим. Если ссылка не содержит локальных ресурсов, она называется внешней.

Расширенная ссылка задается с помощью значения `extended` атрибута `xlink:type`. Расширенные ссылки обычно указывают на несколько удаленных ресурсов. Они также могут происходить из нескольких источников. Концепция локальных и удален-

ных ресурсов достаточно сложна в силу относительности этих понятий. Один ресурс может быть локальным по отношению к одной ссылке XLink и удаленным — по отношению к другой. В настоящее время ни один программный продукт не обладает поддержкой расширенных ссылок. Как только такие средства станут доступными, расширенные ссылки превратятся в очень мощные средства.

Элементы-локаторы

Как уже отмечалось раньше, удаленные ресурсы — это ресурсы, которые находятся за пределами элемента расширенной ссылки. XLink обеспечивает концепцию элементов-локаторов, упрощающих определение месторасположения удаленных ресурсов. Элементы-локаторы — это дочерние элементы по отношению к элементу расширенной ссылки. Этим элементам можно присвоить любое имя, но их атрибут `xlink:type` обязательно должен иметь значение `"locator"`. Атрибут `xlink:href` обязателен для элемента-локатора и используется для указания на удаленный ресурс. С элементами `locator` связаны элементы `resource`, которые обладают подобной структурой, но используются для указания локальных, а не удаленных ресурсов.

Предположим, в системе обработки сообщений, с которой вы уже имели дело в настоящем уроке, отслеживаются не только сообщения и их отправители, но и предпринимаемые над ними действия. Например, представьте, что ваша автоматизированная система контролирует следующие действия, связанные с сообщениями.

- Просьба позвонить.
- Просьба о встрече.
- Ответ на ваш приглашение.
- Предложение проанализировать сообщение.

Развивая этот сценарий, можно сказать, что с сообщением связан целый ряд различных действий. Расширенные ссылки XLink поддерживают отношения один-ко-многим, а также многие-ко-многим. Предположим, что элемент `message` связан с несколькими элементами `action`. Вы можете отразить подобные связи с помощью документа, представленного в листинге 10.7. Однако на момент написания настоящей книги броузеры еще не поддерживают работу с расширенными ссылками; приведенные дальше примеры не будут функционировать и в Netscape 6.

Листинг 10.7. Расширенный элемент XLink, содержащий локаторы `message05.xml`

```
<?xml version="1.0"?>
<!-- листинг 10.7 - message05.xml -->

<note xmlns:xlink="http://www.w3.org/1999/xlink"
            xlink:type="extended">
 <message xlink:type="resource">
                  Remember to buy milk on the way home from work
    </message>

     <action xlink:type="locator"
                  xlink:href="phoned.xml"/>

     <action xlink:type="locator"
                  xlink:href="see_you.xml"/>

     <action xlink:type="locator"
                  xlink:href="Will_call_back.xml"/>

</note>
```

АНАЛИЗ Элемент `note` описывает расширенную ссылку (строка 5) с четырьмя ресурсами. Первый из них — это локальный ресурс `message` (строки 6–8). Три локатора `action` представляют удаленные ресурсы (строки 10, 13 и 16). Обратите внимание, что все четыре ресурса являются дочерними элементами по отношению к элементу-контейнеру расширенной ссылки `note` (строки 4–19). Так как элемент `message` — это локальный ресурс, значение его атрибута `xlink:type` равно `resource` (строка 6), а не `locator`, как в случае удаленных ресурсов. Каждый из элементов `action` содержит адрес URL (строки 11, 14 и 17) удаленного ресурса.

Элементы arc

При создании простых ссылок раньше в настоящей главе вы использовали атрибуты `xlink:show` и `xlink:actuate` для определения того, как и когда будут обрабатываться ссылки. В случае внешних ссылок ситуация оказывается посложнее из-за количества ссылок, которые можно активизировать в любой момент времени. Например, рассмотрите 16 возможных ссылок между ресурсами в документе `message05.xml` (листинг 10.7).

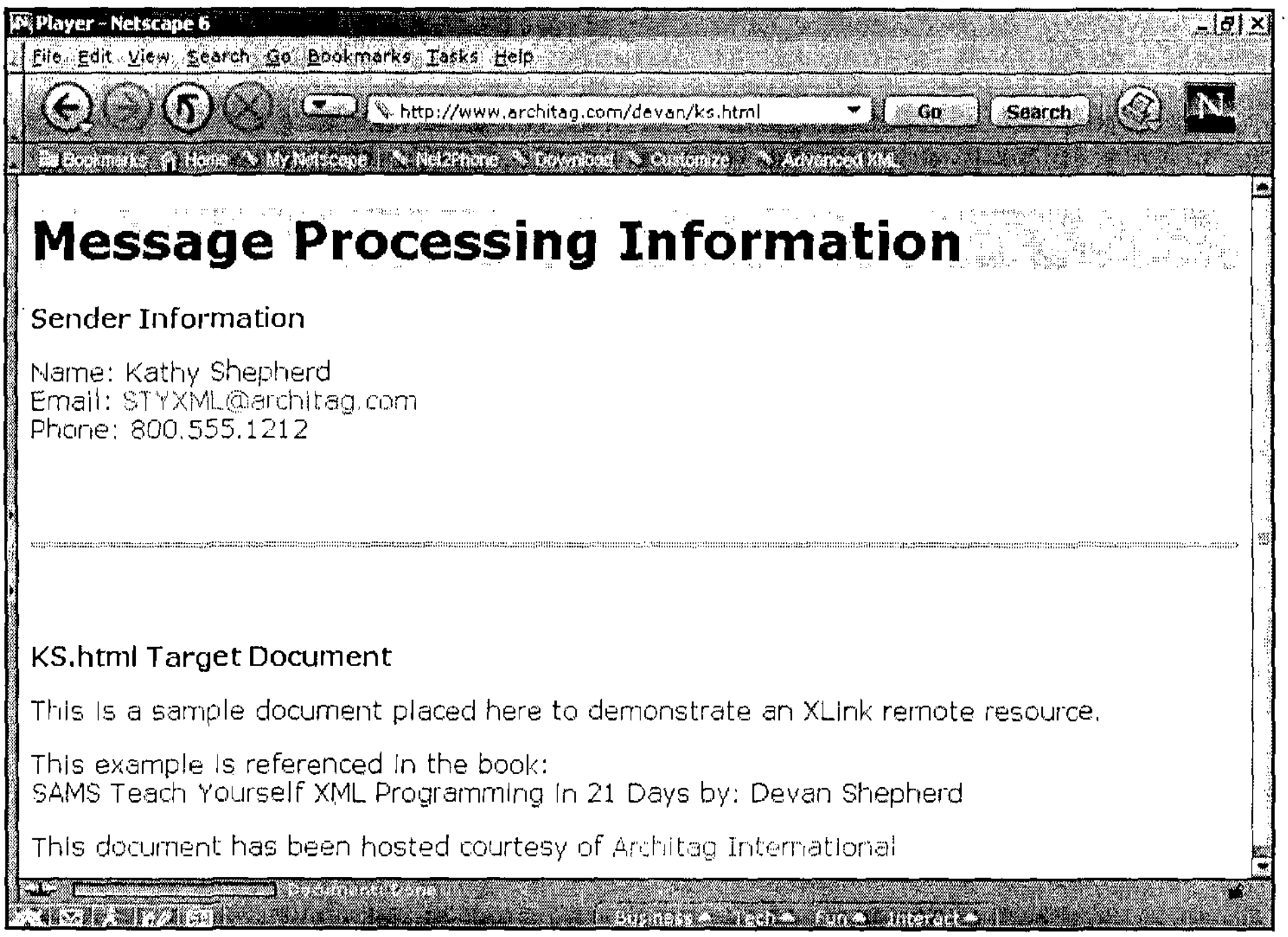


Рис. 10.4. Расширенная модель XLink, содержащая локальные ресурсы, удаленные ресурсы и элементы `arc`

Каждая из ссылок между ресурсами характеризуется различными способами отображения и приведения в действие. В языке XLink каждое отношение указывается с помощью значения `arc` атрибута `xlink:type`. Правила поведения вместе с каждым объявлением `arc` описывают, как и когда обрабатываются ссылки. Кроме того, элементы `arc` обладают атрибутами `xlink:from` и `xlink:to`, которые описывают, с какими именно ресурсами они связаны. Для того чтобы все это сработало, ресурсы должны обладать атрибутами `xlink:label`, выступающими в роли ссылок на указатели `from` и `to`. Расширенная модель XLink представлена на рис. 10.4.

Пример сложной расширенной ссылки

Предположим, ваша система обработки сообщений способна отслеживать сообщения, с которыми связаны несколько действий, как уже отмечалось раньше, а также внешние ресурсы, связанные с несколькими отправителями. Например, удаленные ресурсы `sender` содержат сведения об отправителях. Все эти отношения можно представить в виде сложной расширенной ссылки. Например, в листинге 10.8 по сравнению с листингом 10.8 добавлено два элемента `sender` и `arc`. Каждый из элементов `sender` на самом деле является элементом `locator`, используемым для адресации удаленного ресурса.

Листинг 10.8. Сложный элемент Xlink `message06.xml`

```
1:  <?xml version="1.0"?>
2:  <!-- листинг 10.8 - message06.xml -->
3:
4:  <note xmlns:xlink="http://www.w3.org/1999/xlink"
5:            xlink:type="extended">
6:
7:      <message xlink:type="resource"
8:                    xlink:role="message"
9:                    xlink:href="msg.xml"/>
10:
11:     <sender xlink:type="locator"
12:                    xlink:href="ks.xml"
13:                    xlink:role="sender"/>
14:
15:     <sender xlink:type="locator"
16:                    xlink:href="gs.xml"
17:                    xlink:role="sender"/>
18:
19:     <action xlink:type="locator"
20:                   xlink:href="phoned.xml"
21:                   xlink:role="action"/>
22:
23:     <action xlink:type="locator"
24:                   xlink:href="see_you.xml"
25:                   xlink:role="action"/>
26:
27:     <action xlink:type="locator"
28:                   xlink:href="Will_call_back.xml"
29:                   xlink:role="action"/>
30:
31:     <relate xlink:type="arc"
32:                   xlink:from="message"
33:                   xlink:to="sender"/>
34:
35:     <relate xlink:type="arc"
36:                   xlink:from="message"
37:                   xlink:to="action"/>
38:
39:     <relate xlink:type="arc"
40:                   xlink:from="sender"
41:                   xlink:to="action"/>
42:
43: </note>
```

Строки 7–9 содержат элемент `message` с атрибутом `role`, значение которого равно `message`. Каждый из элементов-локаторов является элементом

sender или action, на что указывает соответствующее значение их атрибутов role. Три элемента relate содержат атрибуты xlink:to и xlink:from, определяющие связи между ресурсами.

Вы уже знаете, что значительная часть рассмотренных нами конструкций не поддерживается ни одной программой. Однако по мере развития технологий вы получите новые возможности для связывания данных из нескольких ресурсов, независимо от их месторасположения в Internet.

Резюме

Язык XLink (XML Link Language) предоставляет XML гибкие возможности связывания элементов HTML. Такие средства, как простые и расширенные ссылки, предназначены для обнаружения локальных и удаленных ресурсов, а не только документов. XLink предоставляет широкие возможности по приведению ссылок в действие, а также отображению ресурсов. Для указания способа обработки ссылок предназначены элементы arc. XLink еще не обрел широкой поддержки со стороны броузеров и обработчиков, хотя такая поддержка должна появиться в самое ближайшее время. Новые устройства и агенты пользователей, полностью использующие все преимущества XLink, также должны появиться в обозримом будущем.

Вопросы и ответы

Каково основное различие между фундаментальными типами ссылок XLink?

Простая ссылка — это однонаправленная ссылка, аналогичная ссылкам HTML, но предоставляющая более широкие функциональные возможности. Расширенная ссылка позволяет связывать несколько ресурсов.

Необходима ли проверка атрибутов XLink?

Поскольку атрибуты XLink являются дополнительными по отношению к стандартным атрибутам в экземпляре документа XML, требуется их проверка с использованием DTD или других видов схем.

Какова роль элементов-локаторов в XLink

Элемент-локатор содержит адрес, обычно URL, удаленного ресурса.

Упражнение

Проверьте свои знания, полученные в результате изучения настоящего урока. Ответ вы найдете в Приложении А "Ответы на вопросы".

Несколько предыдущих дней вы работали с языком разметки музыкальных коллекций MCML. В настоящем уроке добавьте простую ссылку к одному из элементов CD, которая приводится в действие после щелчка мышью и приводит к замене текущего содержимого окна броузера содержанием документа shelf.xml. Вам необходимо создать документ shelf.xml, добавив в него содержимое каждого из компакт-дисков. Проверьте полученные результаты с помощью Netscape 6.0.

День 11-й

Язык XPointer

Язык XPointer помогает адресовать и обнаруживать информацию в рамках другого документа XML. XPointer расширяет язык XPath, с которым вы познакомились в уроке дня 9-го. XPointer объединяет выражения XPath с идентификаторами URI. Вы уже познакомились подробно с обеими этими технологиями, поэтому сегодня мы поговорим о том, как их совместить в выражении XPointer.

Сегодня рассматриваются следующие темы.

- Каким образом XPointer улучшает подход к идентификации фрагментов с помощью анкеров в HTML.
- Синтаксис выражений XPointer.
- Как использовать выражения отношений XPath в XPointer.
- Несколько специальных характеристик XPointer.

Именованные фрагменты и гипертекстовые ссылки указывают HTML

Спецификация XPointer, ее рабочий черновик консорциума *W3C* датирован 9 июля 1999 года, описывает строковое выражение, которое поддерживает прямую адресацию внутренних структур документов XML. Строки XPointer обеспечивают ссылки на элементы, текстовые строки, а также другие элементы документов XML, поскольку они базируются на выразительном и избирательном языке XPath. Это одна из отличительных черт XPointer, выгодно отделяющая его от анкерных ссылок HTML.



Я рекомендую вам изучить материал, посвященный XPath (день 9-й) и XLink (день 10-й), прежде чем приступать к настоящему уроку. Здесь представлен обзор стандартных средств XPath и XLink; его не стоит изучать изолированно от материала двух предыдущих уроков.

Давайте рассмотрим, каким образом "указывание" реализуется в HTML. Для нахождения определенного фрагмента HTML-страницы необходимо задать значение атрибута `name`, прежде чем будет переход по ссылке. Такой подход используется в HTML как для организации переходов между частями одного документа, так и для переходов к фрагментам других документов. Синтаксис соответствующего дескриптора следующий:

```
<a name="target_name">Текст или объект</a>
```

Так указывается месторасположение анкера на странице. Значение атрибута `name` связано с фрагментом HTML-страницы и предоставляет его адрес. Отдельный анкерный дескриптор, ссылающийся явным образом на именованное значение атрибута, может разрешить адрес. Синтаксис вызывающего анкерного дескриптора следующий:

```
<a href="URI#traget_name">Ссылка</a>
```

Гипертекстовая ссылка, созданная подобным образом, указывает на именованный фрагмент на странице в соответствии с идентификатором URI. После того как пользователь активизирует ссылку, щелкнув мышью на соответствующей ей надписи или объекте в окне броузера, броузер загрузит содержание именованного фрагмента и отобразит его в своем окне. Соответствующий механизм проиллюстрирован на рис. 11.1.

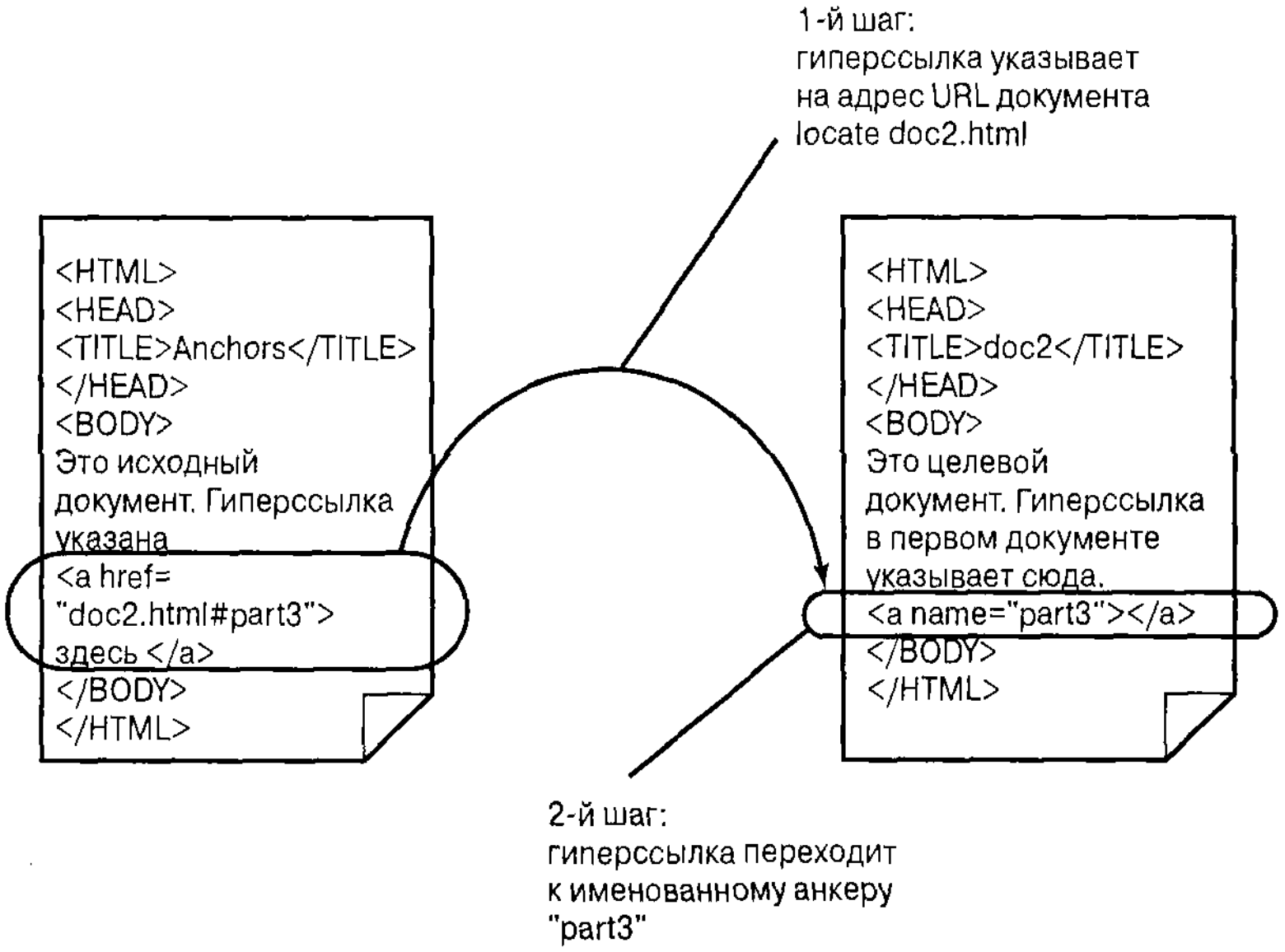


Рис. 11.1. Гипертекстовая ссылка HTML указывает из одного документа на именованный фрагмент в другом документе

Ограничения анкерных ссылок HTML

Для описанного выше подхода HTML характерен целый ряд ограничений. Прежде всего вам необходимо знать имена всех доступных целей. Если вы автор страницы, это не представляет особых проблем. Однако, если вы работаете со страницей на удаленном Web-узле, к созданию которой не имеете никакого отношения, необходимость знать точные имена целей, становится достаточно серьезной проблема. Конечно же, для того чтобы описанный выше подход сработал, все цели должны обязательно существовать. В противном случае вы не сможете использовать именованный анкер для обнаружения фрагмента страницы. Это означает, что HTML-разработчик должен заранее размещать на странице все необходимые именованные анкеры.

Другая проблема связана с безопасностью. Предположим, вам необходимо получить доступ к фрагментам HTML-страницы, которым еще не присвоены имена. Если вы хотите создать соответствующие именованные анкеры, вам необходимо обладать соответствующими правами для доступа и изменения страницы на сервере, на котором она размещена. Неизменные Web-страницы являются документами с атрибутом "только чтение"; не обладая соответствующими правами, вы не сможете внести в них

необходимые изменения. Обычно у вас есть возможность просматривать исходный код Web-страницы с атрибутом "только чтение", чтобы определить, какие именованные анкеры в ней уже существуют, но без должных прав вы не сможете добавить в нее новые анкеры. XPointer предоставляет авторам документов XML возможность доступа к фрагментам документа с атрибутом "только чтение".

Выражения XPointer

XPointer использует общий язык выражения XPath, с которым вы познакомились в уроке дня 9-го, значительно расширяя его возможности. Действительно, XPointer — это выражения XPath, дополненные идентификаторами URI, что позволяет вам решать следующие задачи.

- Обращение к диапазонам и узлам в целевом документе.
- Обнаружение информации в целевом документе при совпадении строк
- Реализация выражений адресации с помощью идентификаторов как идентификаторов фрагментов.

Богатый язык выражений XPointer позволяет вам обнаруживать информацию в ходе перемещения по структуре документа. Таким образом, вы можете выбрать фрагменты, базируясь на таких свойствах, как типы элементов, значения атрибутов, содержание элементов, относительное положение и порядок следования. Это предоставляет вам значительно более широкие возможности обнаружения, чем при использовании собственно XPath. XPath позволяет находить узлы в документе, в то время как XPointer позволяет находить узлы и определенные данные в содержании элементов, вплоть до отдельных символов.

Синтаксис XPointer

XPointer использует расширенную схему адресации URI, а не принятую в HTML адресацию с использованием анкеров. Синтаксис выражений XPointer следующий:

```
URI#scheme(expression)
```

При создании выражений XPointer вы указываете идентификатор URI, после него знак # (решетка), а затем *часть фрагмента* XPointer, которая состоит из схемы (`scheme`) и выражения (`expression`). Идентификатор URI отделяется от части фрагмента (а значит, схемы плюс выражение) с помощью знака #. Идентификатор URI отвечает за обнаружение ресурса, чаще всего это документ, а выражение XPointer представляет собой адрес определенного фрагмента в найденном документе.

В выражении XPointer можно указывать больше одного фрагмента; в данном случае синтаксис выражения будет таким:

```
URI#scheme(expression)scheme(expression)scheme(expression)...
```

В настоящее время единственной определенной схемой является схема `xpointer`, которая информирует обработчик о том, язык XPath используется в качестве языка адресации. Возможно, в будущем станут доступными и другие языки выражений, позволяющие расширить синтаксис XPointer, и тогда схемы можно указывать какими-то другими способами.

Обычно вам придется указывать подряд несколько фрагментов только в том случае, если структура целевого документа оказывается под вопросом. Это связано с тем, что при разрешении выражения XPointer считываются слева направо, однако обработчик прекратит считывание, как только полученный путь к фрагменту будет задан полностью.

Это значительно отличает XPointer от XPath, в котором в любом случае считывается полное выражение. Используя XPointer, вы можете включать часть фрагмента, а также указывать альтернативное выражение, если первое не сможет выполниться.

Поскольку в настоящее время единственной описываемой в спецификации схемой является схема XPointer, возможна и сокращенная форма представления выражения, как показано ниже:

```
URI#expression
```

Идентификационные ссылки XPointer

Один из способов использования XPointer заключается в добавлении атрибута `id` в элемент, на который вы планируете ссылаться. Это в чем-то аналогично подходу именованных фрагментов, принятому в HTML; при этом предполагается, что вам потребуется внести изменения в выражения XPointer. Простая строка после знака "решетки", не содержащая никаких других элементов, считается ссылкой на элемент с соответствующим атрибутом ID. Например, рассматривая систему управления сообщениями, с которой вы имели дело в нескольких предыдущих уроках, предположим, вам необходимо найти определенное сообщение, отмеченное некоторым идентификатором. Предположим, на данный момент в документе три сообщения, и вы хотите использовать выражение XLink для обнаружения первого из этих сообщений, для которого атрибут `ID` элемента `message` равен `m1`. В листинге 11.1 приведен соответствующий документ XML.

Листинг 11.1. Экземпляр XML, содержащий атрибуты ID в элементах `message` — `message01.xml`

```
1: <?xml version = "1.0"?>
2: <!-- листинг 11.1 - message01.xml -->
3:
4: <note>
5:     <message ID="m1" from="Kathy Shepherd">
6:        Remember to buy milk on the way home from work
7:     </message>
8:     <message ID="m2" from="Greg Shepherd">
9:        I need some help with my homework
10:    </message>
11:    <message ID="m3" from="Kristen Shepherd">
12:       Please play Scrabble with me tonight
13:    </message>
14:</note>
```

В стоках 5, 8 и 11 атрибуты `ID` добавлены к элементам `message`.

Для нахождения элемента `message` с атрибутом `ID="m1"` вы можете использовать сокращенную форму выражения XPointer:

```
message01.xml#m1
```

Это выражение найдет элемент `message` в строке 5 листинга 11.1. Его более сложная форма имеет следующий вид:

```
message01.xml#ID(m1)
```

Две формы выражения XPointer функционируют идентичным образом. Вы можете использовать любую из них. Последняя форма больше соответствует синтаксису других выражений XPointer, в которых обычно указывается ключевое слово, после которого в скобках указывается значение или аргумент.

Абсолютная адресация

В предыдущем примере выражение `id()` способно обнаружить определенный элемент без предоставления каких-либо дополнительных сведений. Это пример *элемента абсолютного обнаружения*. Четыре из этих элементов используются в выражениях XPointer. Они указываются после знака "решетки", но их нельзя объединять. Другими словами, в каждом выражении допускается использование только одного такого элемента. Все элементы абсолютной адресации XPointer перечислены в табл. 11.1.

Таблица 11.1. Элементы абсолютного обнаружения XPointer

Элемент	*Описание*
`id()`	Обнаруживает элемент с атрибутом `id`, равным значению, указанному в скобках
`root()`	Обнаруживает корневой элемент документа, указанного с помощью идентификатора URI
`html()`	Обнаруживает элемент `<A>` на HTML-странице, которая также является и правильным документом XML (а значит, документом DHTML), если значение этого элемента совпадает с указанным в скобках
`origin()`	Обнаруживает корневой элемент документа, который подобен элементу `root()`, но только находится в документе-источнике

Выражения отношений

Выражения отношений, или *элементы относительного обнаружения*, определяют расположение элемента по отношению к некоторому узлу. Другими словами, выражение может привести к другому результату, в зависимости от того, откуда начат "поиск". О отношениях и осях XPath вы узнали из урока дня 9-го. Использование их совместно с идентификатором URI, отделенным знаком "решетки", образует относительное выражение XPointer. Выражения отношений функционируют в XPointer практически так же, как в XPath, поэтому изучите еще раз примеры из урока дня 9-го, чтобы освежить сведения. Вы должны помнить, существует несколько различных осей отношений; однако в XPointer применимы далеко не все из них. К действительным в XPointer относятся следующие.

- Дочерний элемент.
- Предок.
- Потомок.
- Следующий.
- Предыдущий.
- Предыдущий брат
- Следующий брат

У вас может возникнуть вопрос, почему именно некоторые из осей отношений XPath не включены в этот список, такие как родитель или предок самого себя. Это связано с тем, что XPointer позволяет вам добавлять аргументы к выражениям для поиска элементов, находящихся в определенном отношении к некоторому элементу, и вы можете использовать любые виды отношений XPath. Если, например, вам необходимо выбрать первый предок по отношению к текущему элементу, вы выбираете родительский элемент.

Каждое из выражений для обнаружения элементов позволяет указывать вплоть до четырех аргументов. Эти аргументы позволяют повышать "чувствительность" выражений XPointer. Аргументы одинаково влияют на каждое из отношений и характеризуются своим абсолютным расположением в скобках. Все доступные аргументы кратко описаны в следующих разделах. Это обычные выражения XPath, используемые в XPointer; таким образом, вы увидите много похожего между этими примерами и выражениями, созданными вами в уроке дня 9-го. Несмотря ни на что, различные примеры позволяют вам понять общую логику XPath и XPointer.

Числовой выбор

Первый аргумент представляет собой номер необходимого варианта в наборе. Этот аргумент может быть положительным или отрицательным числом или ключевым словом `all`. Аргумент `all` позволяет выбрать все элементы из набора отношений. Заданное число позволяет выбрать определенный член из набора в соответствии с порядком документа, если число положительное, или выбрать определенный член из набора в соответствии с обратным порядком документа, если число отрицательное. Например, вам необходимо выбрать все элементы `message` из листинга 11.1, а в данный момент вы "находитесь" в элементе `note`. Вам необходимо выбрать все дочерние элементы по отношению к текущему узлу. В качестве примера рассмотрим следующее выражение:

```
child(all,message)
```

Это выражение выбирает все дочерние элементы `message`, которые содержатся в текущем элементе. В листинге 11.1 это приведет к выделению всех элементов `message`, если выражение было разрешено по отношению к элементу `note`. Если выбран другой элемент документа, то выражение может вернуть другой результат. На самом деле, в случае документа `message01.xml` (см. листинг 11.1) выражение приведет к неудачному результату, если будет выполнено к любому элементу, отличному от `note`.

Предполагая, что выбран именно элемент `note`, каким образом вы сможете выбрать именно третий дочерний элемент `message` с помощью выражения XPointer? Правильное выражение должно иметь следующий вид:

```
child(3,message)
```

Числовое значение (`3`) выбирает определенный элемент при вычислении выражения. В данном случае выбран родительский элемент, который является третьим дочерним элементом `message` по отношению к элементу `note`.

Тип узла

Второй аргумент определяет тип узла, обрабатываемого XPointer. Чаще всего речь идет об имени элемента. Вы увидите пример второго аргумента и в последнем разделе настоящего урока. Приведем еще раз примеры из предыдущего раздела:

```
child(all,message)
child(3,message)
```

В этих двух простых выражениях XPointer в качестве типа узла выступает тип элемента `message`. Первый пример приводит к выбору всех дочерних элементов `message`, а второй — только третьего из них.

Второй аргумент допускает использование подстановочных символов для расширения области поиска за рамки определенного типа. Это значение аргумента `#element`. Для того чтобы найти все дочерние элементы, относящиеся к определенному элементу, независимо от их типа, вы можете использовать следующее выражение:

```
child(all,#element)
```

Это выражение позволяет выбрать все дочерние элементы, независимо от их имен.

Фильтры атрибутов и их значений

Третий и четвертый аргументы — это пара значений атрибутов, которые можно использовать для отбора элементов в зависимости от значений их указанных атрибутов. Предположим, вам необходимо исследовать документ `message01.xml` (см. листинг 11.1) для нахождения элемента `message`, атрибут `from` которого равен `Kathy Shepherd`. Вы можете составить необходимое выражение несколькими способами, однако наиболее простым будет следующий:

```
child(all,#element, from, "Kathy Shepherd")
```

Это выражение выбирает все элементы, независимо от их типа, атрибут `from` которых равен `Kathy Shepherd`.

Вместо имени атрибута в качестве третьего аргумента вы можете использовать звездочку (*). Это означает, что любой атрибут с заданным значением будет выбран.

Вместо четвертого аргумента вы можете использовать звездочку или подстановочный символ для любого значения или ключевое слово `#IMPLIED`. Это ключевое слово, заимствованное из логики DTD, указывает, что для данного атрибута не нужно указывать никакого значения.

Резюме

XPointer — это технология, которая позволяет объединять идентификаторы URI с выражениями XPath для более точного выбора узлов в целевом документе. XPointer расширяет возможности XPath с помощью ряда средств. А значит, с XPointer вы получаете не только технологию с более широкими возможностями, но и более "избирательную".

Вопросы и ответы

Почему нельзя использовать анкерные дескрипторы HTML для доступа к фрагментам структуры документа с атрибутом "только чтение"?

В некоторых случаях это возможно, однако вы должны указать имена анкеров в документе HTML перед соответствующими фрагментами. Если вам необходимо получить доступ к фрагментам документа, для которого раньше не были указаны анкерные дескрипторы, значит, что HTML вам не подходит. XPointer позволяет вам выбирать элементы, пользуясь критериями, которые не ограничены именами анкерами, такими как шаблоны, типы элементов и атрибутов, а также другими результатами применения выражений XPath.

Какова связь между XPointer и XPath?

XPointer базируется на технологии XPath, к которой добавлены идентификаторы URI. XPointer используется для выбора фрагментов документа, найденного по идентификатору URI.

Почему синтаксис XPointer предоставляет несколько частей фрагмента?

Часть фрагмента — это комбинация схемы и выражения. На данный момент единственной схемой является XPointer. Несколько частей фрагментов можно объединить для более точного выбора фрагмента в экземпляре документа XML. При обработке выражения XPointer сначала разрешается первая часть фрагмента. В случае успеха дальнейшая обработка прекращается. Однако, если часть фрагмента нельзя разрешить, начинается обработка второй части, если таковая существует. Тогда программисты получают возможность использовать дополнительные шаблоны для нахождения фрагментов.

День 12-й

Объектная модель документа XML – DOM

Объектная модель документа DOM для XML предоставляет общий прикладной интерфейс API, спецификацию консорциума *W3C*. XML DOM дает стандартизированный способ доступа к информации, содержащейся в документах XML, и манипулирования ею, описывая стандартные свойства, события и методы для взаимодействия приложения и документов XML.

Сегодня рассмотрим следующие темы.

- Как можно исследовать узлы DOM и манипулировать ими.
- Объектные модели.
- Основные типы API.
- Объекты, свойства, методы и события DOM.
- Написание простых сценариев DOM с помощью JavaScript.

DOM для XML

XML DOM — это объектная модель, которая охватывает содержание всего документа XML. Это означает, что содержание документа не только доступно, но и им в некоторой степени можно манипулировать. Спецификация Document Object Model (DOM) консорциума *W3C* в настоящее время определяет, что DOM позволяет использовать свойства, методы и события. Полный текст спецификации Document Object Model (DOM) вы найдете по адресу `http://www.w3.org/DOM`.

В настоящем уроке я покажу, как создавать сценарии, использующие DOM, однако не буду обучать вас программированию на JavaScript. Если вы хотите получить подробные сведения о языке программирования JavaScript, обратитесь к книге *Освой самостоятельно JavaScript за 24 часа*.

Структура документа XML очень часто сравнивается с деревом, содержащим несколько узлов. Обе структуры могут иметь различную форму, для которой любая произвольно выбранная часть подобна данной большой или еще меньшей части. Другими словами, конструкции структурных компонентов и отношений повторяются. Если продолжить аналогию с ботаникой, то узлы дерева соответствуют ветвям, корням и стволу и т.д. Завершающие узлы не содержат дочерних. С другой стороны, ветви могут содержать дочерние ветви, которые, в свою очередь, также могут содержать дочерние ветви и т.д. и т.д. Из материала предыдущих уроков вы уже знаете, что подобная модель представляет собой параллель со структурой документа XML, которая характеризуется такими узлами, как элементы, атрибуты, текст, комментарии и инструкции об-

работки. Представьте себе листья и плоды на дереве, которые не могут содержать дочерние узлы; это аналоги атрибутов и текстовых данных в документе XML, которые также не могут содержать дочерние узлы. Определив ссылки на дочерние и родительские узлы, вы можете карабкаться по дереву или достигать любой части документа. Модель DOM расширяет эту структуру, представляя каждый узел в виде объекта со своими свойствами и методами. Модель DOM определяет иерархию объектов. Каждый узел в структуре документа может содержать любое количество дочерних узлов, а все узлы, за исключением корневого, также обладают и родительским узлом. Доступ к каждому узлу осуществляется по имени. Спецификация DOM предоставляет разработчику целый ряд способов использования подобного дерева документа.

Таким образом, узел представляет собой наименьшую единицу в иерархической структуре документа XML. Объект `Node` — это один узел в дереве документа, а объект `Document` — его корневой узел.

Список узлов, доступных в модели DOM для языка XML, приведен в табл. 12.1.

Таблица 12.1. Узлы в модели DOM для языка XML

Название узла	*Описание*
Node	Ссылка на компонент документа, такой как элемент, атрибут, комментарий или текстовая строка
Document	Объект, представляющий целый документ, содержащий все узлы
Element	Объект, представляющий элемент документа
Attribute	Объект, представляющий атрибут элемента в виде пары имя–значение
Processing Instruction	Инструкции, используемые обработчиком XML и указанные в документе
Text	Объект, представляющий текстовое содержание элемента
CDATA Section	Текстовое содержание, в котором отсутствуют знаки разметки
Document Fragment	Определенная часть документа XML
Entity	Жетон, используемый обработчиком XML при подстановке строки
Entity Reference	Ссылка для подстановочной строки, в начале которой указан знак амперсанда (&), а в конце — точка с запятой (;)
Document Type	Правило, определяющее элементы и атрибуты

Консорциум *W3C* определил стандарт DOM на нескольких уровнях. На первом уровне представлены основные модели описания документов с помощью HTML или XML. На этом уровне вам необходимы достаточно широкие функциональные возможности для перемещения по документу и манипулирования им. В настоящем уроке вы познакомитесь с уровнем 1 модели DOM достаточно подробно.

Уровень 2 модели DOM включает в себя объектную модель каскадных стилей. Добавляя к документу сведения о стиле, вы можете манипулировать представлением информации, связанной с документом. Уровень 2 обеспечивает возможность перемещения по документу с помощью событийной модели, а также обеспечивает поддержку пространств имен XML. В качестве событийной модели для перемещения по документу вы завтра, в следующем уроке, рассмотрите стандарт SAX (Simple API for XML).

Консорциум *W3C* определил уровень 3 модели DOM для загрузки адресов и сохранения документов вместе со связанными с ними моделями содержания, как это имеет место при использовании DTD и других схем. Уровень 3, еще не разработанный до конца, должен обеспечить проверку документа на предмет форматирования, ключевых событий и групп событий. Подробные сведения по данным вопросам вы найдете по адресу `http://www.w3.org/DOM/`.

За рамками уровня 3 предполагается, что DOM будет содержать общие вызовы окон для операционных систем Microsoft Windows, UNIX и Macintosh. Кроме того, к потенциальным функциональным возможностям DOM относят следующие.

- Запрос сведений у пользователя.
- Языки запросов с помощью интерфейса пользователя.
- Многопоточная адресация.
- Синхронизация реализации объектов.
- Повышенная безопасность.
- Функции хранилища.

Существует целый ряд разновидностей DOM, ориентированных на определенные языки разметки или их диалекты. Например, языку SVG (Scalable Vector Language), предназначенному для описания двумерных, полученных с помощью математических формул векторов, соответствует собственная разновидность модели DOM — SVG DOM, позволяющая программистам гораздо проще решать определенный круг задач. Консорциум *W3C* представил язык SVG как язык разметки, позволяющий работать с тремя типами графических объектов: фигурами векторной графики (состоят из прямых линий и кривых), изображениями и текстом. Графические объекты могут группироваться, к ним могут применяться стили, а также возможно их преобразование и компоновка с ранее обработанными объектами.

В настоящее время SVG — кандидат в рекомендации консорциума *W3C*.

Объектные модели

DOM предоставляет программистам возможность взаимодействия с информацией, сохраненной в виде текста. Это очень важно для Web-разработки, поскольку полностью статические документы теперь становятся приложениями. Таким образом, сам документ становится интерфейсом для информации пользователя, но, в отличие от газеты и журнала, позволяет изменять как содержание, так и форму представления этих данных. Наиболее распространенным форматом представления статических данных в окне броузера, конечно же, является HTML. Функциональные возможности и интерфейс для представления информации уже встроены в броузер, поэтому разработчик обладает минимальным контролем над ними. Динамический HTML (DHTML) дает разработчикам контроль над доставкой и представлением данных в Web, обычно с помощью языков сценариев, таких как JavaScript или VBScript, а также с помощью таблиц каскадных стилей CSS.

Вообще говоря, объектная модель — это структура для поддержки динамических поведений, использующая методы объектов, представленных моделью. Если вы знакомы с JavaScript или VBScript, то вам уже наверняка известно, что такое методы, события и объекты. До реализации поддержки DHTML в броузерах четвертого поколения JavaScript не мог манипулировать содержанием документа, по крайней мере, глобально. С выходом четвертых версий броузеров компании *Netscape* и *Microsoft* предложили свои варианты объектных моделей документа DOM. К сожалению, предложенные обеими компаниями реализации не идентичны. По этой причине одной из задач, поставленной консорциумом *W3C* при разработке уровня 1 модели DOM, было обеспечение независимости от платформ прикладного интерфейса DOM API. По мере разработки технологий все это стало более или менее естественным, но вам необходимо обязательно знать о существовании различий между реализациями модели DOM, выполненными компаниями *Netscape* и *Microsoft*. Например, компания *Microsoft* расширила стандарт DOM консорциума *W3C* методами, недоступными в броузерах компании *Netscape*.

Создание структур DOM с помощью сценариев

DOM обеспечивает и использует организованную структуру объектов и сценариев для доступа к ее узлам, а также манипулирования ими. Сценарий может указывать на узел по его относительному или абсолютному положению, например первый узел в структуре документа. Сценарии также способны добавлять или удалять узлы. Благодаря этому содержание каждого узла можно получить или обновить как часть приложения. Таким образом, объекты DOM представляют собой хорошо структурированные, уникальным образом идентифицированные контейнеры. Их поведение можно расширить с помощью сценариев, которые вы создаете для манипулирования как самими объектами-контейнерами, так и их содержанием.

Объект `Document` — это корневой узел-контейнер для всех остальных объектов, определенных в документе XML. Этот узел предоставляет доступ к определениям типа документа DTD, а также определяет дочерние элементы в структуре документа. На "корневом" уровне документа находится *узел корневого элемента*, однако это вовсе не одно и то же, что и корневой узел. Как вы уже узнали из уроков 9-го, 10-го и 11-го, дерево документа представляет собой адресуемую и упорядоченную коллекцию узлов. Поскольку все объекты в дереве документа можно идентифицировать, вы можете ссылаться как на них, так и на их содержание. Например, JavaScript позволяет вам сослаться на текстовое содержание первого дочернего узла в структуре документа следующим образом:

```
mystuff.documentType.childNodes.item(0).text
```

На первый взгляд эта инструкция покажется несколько запутанной, однако, если разобраться, она построена абсолютно логично. Документ `mystuff` содержит элемент `documentType` с целым рядом дочерних узлов (`childNodes`). Инструкция ссылается на первый из этих дочерних элементов (`item(0)`), а целью всей инструкции является обращение к текстовому содержанию элемента. Вы уже отметили, что нумерация элементов начинается с нуля; это справедливо по отношению ко всем языкам сценариев.

Как вы уже знаете, одно из основных преимуществ XML состоит в том, что вы можете создавать свои собственные наборы элементов. Таким образом, вы можете создавать объекты, которые базируются на семантике определенного приложения, после чего использовать их с помощью сценариев. Обычно, вам необходимо знать не только порядковый номер или имя объекта; скорее вам необходимо знать содержание объекта в документе. Это связано с тем, что для получения содержания объекта в документе сценарий может использовать отношения объектов.

Отношения DOM

При использовании DOM можно "перемещаться" по иерархии объектов. Другими словами, выбрав определенный объект в качестве отправной точки, вы сможете определить следующие характеристики.

- Размещение дерева документа.
- Дочерние узлы объекта.
- Родительские узлы объекта.
- Браться и предки объекта.

DOM как универсальный прикладной интерфейс API

Как вы уже видели, основное назначение спецификации DOM — предоставление общего интерфейса (API) разработчикам для использования при манипуляциях объектами документа. Прикладной интерфейс API предлагает идентификаторы, которые разработчик сможет использовать при обращении к объектам и методам. Основные компании, разрабатывающие прикладные инструменты, предоставили реализации модели DOM для определенных языков программирования. Основная концепция состоит в том, чтобы DOM API использовался совершенно одинаково, независимо от того, на каком конкретном языке программирования он реализован. Это означает, что реализации DOM API, выполненные на Java, C, C++, C#, JavaScript, Visual Basic, VBScript, Perl и т.д., должны вести себя практически идентично.

Такой платформно-независимый подход требует, чтобы модель DOM была определена на самом общем языке описания. В спецификации DOM консорциума *W3C* использован язык, известный как язык описания интерфейсов IDL (Interface Description Language), для описания природы интерфейсов, используемых каждой реализацией DOM. IDL — это стандарт (стандарт ISO № 14750), разработанный группой разработчиков, известной как OMG (Object Management Group — Группа управления объектами). Подробные сведения об языке IDL вы найдете по адресу `http://www.omg.org/`.

Основные типы API

Раньше вы уже читали о том, что все узлы представляются с помощью модели DOM; однако большинство программистов используют три основных типа API. Это узлы `Node`, `Document` и `Element`. Поскольку эти узлы очень часто встречаются в документах, вам необходимо решать основные задачи программирования DOM, используя только эти узлы. Вы должны помнить, что узел `Node` — это базовый тип для большинства объектов в DOM. Он может содержать любое количество дочерних узлов, а также родительский узел. На самом деле, он обычно содержит родительский узел, за исключением того случае, когда он является корневым узлом в иерархии документа, а значит, не имеет родителя.

Объект узла `Element` используется для представления элементов XML. Как вы знаете, элементы могут содержать текстовые данные (например, строку символов между открывающим и закрывающим дескрипторами), другие элементы, а также атрибуты. Объекты элементов содержат целый ряд объектов атрибутов, которые представляют комбинацию тех атрибутов, выраженных в экземпляре документа, а также атрибутов, определенных в соответствующей схеме, независимо от того, присутствуют они в экземпляре XML или нет.

Объект `Document` представляет корневой узел документа. А значит, он не содержит родителя. Таким образом, вызов метода, выполненный с помощью сценария (например, `getParentNode()`) для получения сведений о родительском узле для узла `Document`, вернет нулевое значение.

Вызов DOM

DOM может реализоваться различными способами с помощью синтаксических анализаторов, поддерживающих обработку DOM. При выполнении упражнений настоящего урока обработчик MSXML используется совместно с JavaScript. В этих упражнениях мы ограничимся использованием Microsoft Internet Explorer. Хотя и существуют версии броузера Internet Explorer для целого ряда платформ и операционных систем, не все посетители Web-узла могут иметь необходимое программное обеспечение. Таким образом, некоторые из рассмотренных в настоящем уроке приемов могут относиться только к доступу к DOM, а не использованию в Web. Web-броузер Microsoft Internet Explorer

для таких платформ, как UNIX, DOS, Macintosh и Windows, можно совершенно бесплатно загрузить с Web-узла `http://www.microsoft.com`. Принципы, с которыми вы познакомитесь в настоящем уроке, применимы и к другим реализациям DOM.

Для решения подобных проблем созданы реализации DOM, поддерживающие различные обработчики и сценарии. Если вы знакомы с другими языками для разработки приложений или создания сценариев, вы сможете найти описание соответствующих методов доступа к объектам DOM по адресу `http://www.w3.org/DOM/`. Там вы найдете ссылки на реализации DOM, выполненные на PERL, Java, Python и нескольких других популярных языках программирования, поддерживающие большое количество платформ.

У большей части реализаций DOM есть целый ряд общих характеристик. DOM предоставляет вам интерфейс для загрузки, доступа и манипулирования документами XML. В следующей главе вы познакомитесь с прикладным интерфейсом SAX (Simple API for XML), который представляет собой модель, управляемую событиями. Основное отличие между этими двумя подходами состоит в том, что DOM загружает в память весь экземпляр документа сразу, а SAX обрабатывает по одному узлу за раз, после чего переходит к следующему узлу согласно порядку документа.

После обработки документа DOM-совместимым обработчиком создается древовидная структура, которая используется для чтения и записи сценариями или языками программирования. После создания дерева объектов синтаксический анализатор управляет им, позволяя программистам использовать методы, встроенные в анализатор, а не создавать что-то новое. Благодаря загрузке в память сразу всего документа возможен произвольный доступ к его отдельным узлам. Взаимосвязь между документом XML, синтаксическим анализатором XML, созданным с его помощью деревом DOM, и сценарием или языком, выполняющим операции чтения и записи, показана на рис. 12.1.

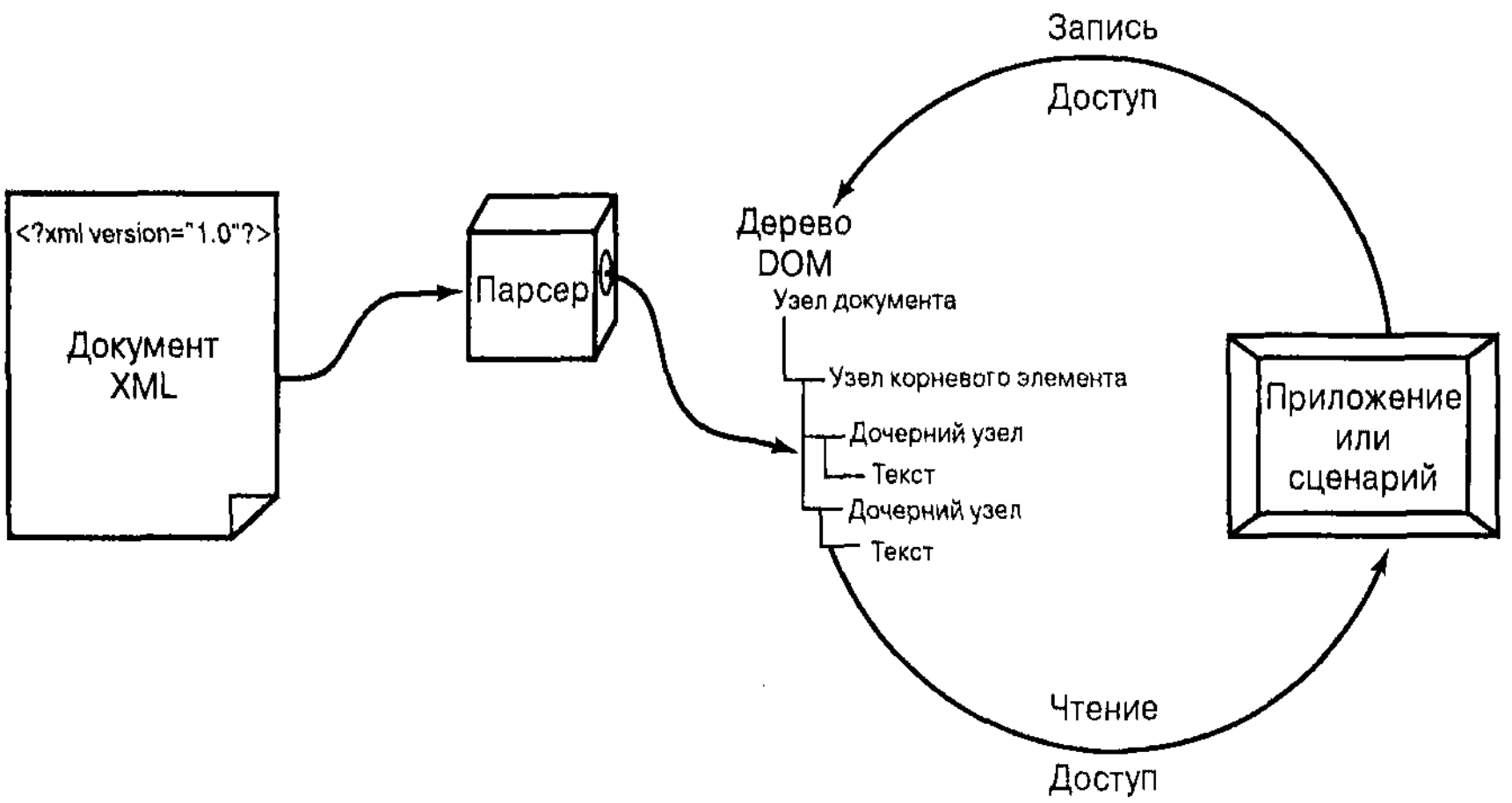


Рис. 12.1. Дерево DOM, реализованное с помощью синтаксического анализатора XML

Для начала использования сценария в дереве DOM вам необходимо создать объект документа. При использовании для этого языка JavaScript и поддерживающего модель DOM синтаксического анализатора MSXML необходимо создать новый объект `ActiveXObject`, который инициализирует компонент COM (Component Object Model — модель компонентных объектов Microsoft), входящий в состав библиотеки, устанавливаемой вместе с Microsoft Internet Explorer 5.0 или более поздней версии. Объекту в нашем случае `ActiveXObject` в библиотеке COM соответствует объект `XMLDOM`. Соответствующая строка кода на JavaScript выглядит следующим образом:

```
Var oMyStuff = New ActiveXObject("Microsoft.XMLDOM")
```

Этот сценарий создает новый объект документа ActiveXObject на базе COM-компонента XMLDOM. Объект документа присваивается переменной oMyStuff. Символ o указывает на то, что переменная представляет объект.

Как уже вам известно, если вы занимаетесь разработкой приложений на Visual Basic, Java или Visual C++, модель COM позволяет создавать языково-независимые интегрированные программные компоненты, которые можно использовать в различных приложениях. Это означает, что вы можете обращаться к синтаксическому анализатору MSXML с помощью любого языка программирования, поддерживающего модель COM. Несмотря на то, что в настоящем уроке при выполнении большинства упражнений вы будете использовать JavaScript, вы также можете выполнять их и на VBScript, создавая объект документа с помощью метода CreateObject, как показано ниже:

```
Set oMystuff = CreateObject("Microsoft.XMLDOM")
```

Этот код на VBScript полностью эквивалентен приведенному раньше коду на JavaScript. Он создает новый объект документа на базе COM-компонента XMLDOM. Объект документа присваивается переменной oMyStuff.

Оба приведенных выше примера создают объект для выполнения сценария на стороне клиента. В действительности, большая часть упражнений, которые вы выполните в настоящем уроке, касаются выполнения сценариев DOM на стороне клиента; однако вы можете создать объект ASP (Active Server Page) для выполнения сценария на стороне клиента. Строка VBScript для создания объекта для выполнения сценария на стороне сервера приведена ниже.

```
Set oMystuff = Server.CreateObject("Microsoft.XMLDOM")
```

Этот код VBScript работает точно так же, как в предыдущих примерах. Он создает объект документа ASP (Server.CreateObject), используя компонент COM. Объект документа присваивается переменной oMyStuff.

Свойства, методы и события объектов DOM

Модель XML DOM компании *Microsoft* содержит четыре основных объекта: XMLDOMDocument, XMLDOMNode, XMLDOMNodeList и XMLDOMNameNodeMap. Другие реализации XML DOM, выходящие за рамки рассмотрения настоящего урока, используют другие объекты, зависящие от языков реализации. Каждый из объектов обладает своими собственными свойствами, методами и событиями. Если вы не знакомы с основными положениями объектно-ориентированного программирования OOP (Object Oriented Programming) и не знаете, в чем состоят отличия между свойствами, методами и событиями, замечательное руководство по языку Java вы найдете на Web-узле компании *Sun Microsystems* по адресу http://java.sun.com/docs/books/tutorial/java/concepts/object.html.

Объект XMLDOMDocument будет активно использоваться в настоящем уроке. Некоторые из часто используемых свойств, методов и событий перечислены в табл. 12.2–12.4. Весь этот набор свойств, методов и событий доступен в реализации модели DOM компании *Microsoft*. Другие реализации могут содержать другие элементы, не перечисленные в таблицах.

Таблица 12.2. Основные свойства объекта XMLDOMDocument

Свойство	*Описание*
async	Указывает на возможность асинхронной загрузки. Если при загрузке документа значение свойства async равно false, синтаксический анализатор прекращает выполнение любых действий до полной загрузки документа в память
attributes	Содержит список атрибутов узла

Свойство	Описание
childNodes	Содержит список дочерних узлов
dataType	Определяет тип данных узла
docType	Содержит узел типа документа, который определяет схему DTD, используемую при проверке документа
document Element	Возвращает корневой узел, который содержит корневой элемент документа XML
firstChild	Содержит первый дочерний элемент узла
lastChild	Содержит последний дочерний элемент узла
namespace URI	Возвращает идентификатор URI для узла пространства имен
nodeName	Содержит имя элемента, атрибута или ссылки для узла
nodeType	Указывает тип узла XML DOM, который определяет допустимые значения, а также наличие дочерних узлов
nodeValue	Содержит текст, связанный с узлом
parseError	Возвращает объект IXMLDOMParseError, который содержит сведения о последней ошибке обработки
preserve WhiteSpace	Требует свободного пространства в документе XML
readyState	Указывает текущее состояние документа XML
resolve Externals	Разрешает пространства имен, схемы DTD и внешние ссылки во время обработки документа
validate OnParse	Указывает на необходимость проверки документа при его обработке
xml	Содержит представление XML узла и всех его потомков

Таблица 12.3. Основные методы объекта XMLDOMDocument

Метод	Описание
Clonenode()	Создает новый узел, точную копию текущего
Create Attribute()	Создает новый атрибут
CreateCDATASec tion()	Создает узел раздела CDATA
CreateComment()	Создает узел комментария
CreatrElement()	Создает узел элемента, используя указанное имя
CreateEntityRe ference()	Создает объект ссылки
CreateNode()	Создает узел
CreateTextNode()	Создает текстовый узел
GetElements ByTagName()	Возвращает коллекцию элементов с указанным именем
HasChildNodes()	Возвращает значение true, если узел содержит дочерние узлы.

Метод	Описание
`Load()`	Загружает существующий документ XML из указанного места расположения
`LoadXML()`	Загружает документ XML из указанной строки, а не с адреса URL
`NodeFromId()`	Возвращает узел, значение атрибута `ID` которого совпадает с указанным
`RemoveChild()`	Удаляет указанный дочерний элемент из списка и возвращает его
`ReplaceChild()`	Заменяет указанный старый дочерний узел указанным новым узлом в наборе дочерних узлов
`Save()`	Сохраняет документ XML

Таблица 12.4. События объекта `XMLDOMDocument`

Событие	Описание
`ondataavailable`	Указывает на то, что данные документа XML доступны
`onreadystatechange`	Указывает, что свойство `readyState` изменилось

Загрузка файла XML с указанного адреса URL

Из табл. 12.2 вы уже знаете, что метод `load()` может использоваться для загрузки документа XML в объект документа DOM, предоставляя узлы документа XML для работы сценариев. Соответствующий метод JavaScript выглядит следующим образом:

```
oMystuff.load('имя_файла.xml')
```

Объект `oMystuff`, созданный раньше, теперь содержит сведения из документа `имя_файла.xml`.

Обычно вам надо убедиться в том, что синтаксический анализатор прекратит выполнение операций при загрузке документа в объект документа DOM. Для этого вам необходимо присвоить значение `false` свойству `async` объекта. При использовании JavaScript это выглядит следующим образом:

```
oMystuff.async = "false"
```

Значение `false` свойству `async` присваивается для запрета асинхронной загрузки, что прекращает выполнение любых операций синтаксическим анализатором до загрузки в память целого экземпляра документа.

Теперь вы знаете обо всех методах и свойствах, необходимых для создания объектов и загрузки документа XML. Используя JavaScript, вам необходимо создать объект `ActiveXObject` для узла документа, а также назначить значение `false` его свойству `async`, после чего загрузить документ с указанного адреса URL. Соответствующий код приведен ниже:

```
1: var oMystuff = new ActiveXObject("Microsoft.XMLDOM")
2: oMystuff.async="false"
3: oMystuff.load("message01.xml")
```



Новый объект `oMystuff` создается в строке 1 как экземпляр синтаксического анализатора XML компании *Microsoft*. В строке 2 прекращается выполнение всех операций синтаксического анализатора до загрузки в память целого экземпляра документа. В строке 3 указан метод `load` вместе с адресом URL для загрузки документа XML.

Для проверки этого сценария начните с создания простого документа XML. Вы можете повторно использовать документ `message01.xml`, созданный при выполнении упражнений в предыдущем уроке. Документ `message01.xml` представлен в листинге 12.1. Единственное отличие состоит в комментарии в строке 2, который указывает на главу 12.

Листинг 12.1. Простой документ XML `message01.xml`

```
1:  <?xml version = "1.0"?>
2: <!-- листинг 12.1 - message01.xml -->
3:
4: <note>
5:     <message ID="m1" from="Kathy Shepherd">
6:        Remember to buy milk on the way home from work
7:     </message>
8:     <message ID="m2" from="Greg Shepherd">
9:        I need some help with my homework
10:    </message>
11:    <message ID="m3" from="Kristen Shepherd">
12:       Please play Scrabble with me tonight
13:    </message>
14:</note>
```

Элемент `note` содержит несколько дочерних элементов `message`, которые могут содержать только текстовые данные. У каждого элемента `message` есть атрибуты `ID` и `from`.

Затем добавьте описанные раньше сценарии в структуру документа HTML для загрузки файла `message01.xml` в объект документа DOM для дальнейшей обработки. Один из вариантов заключается в создании сценария в разделе `<head>` в документе HTML. В случае файла `message01.xml` это выглядит следующим образом:

```
1:  </head>
2: <script language="javascript">
3: <!-- hide script from older browsers
4: var oMystuff = new ActiveXObject("Microsoft.XMLDOM")
5: oMystuff.async="false"
6: oMystuff.load("message01.xml")
7: -->
8: </script>
9: </head>
```

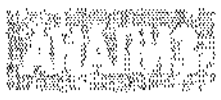
АНАЛИЗ

Этот фрагмент кода содержит сценарий, который начинается в строке 2 в рамках раздела `<head>` (строки 1–9) HTML-страницы. В строке 2 с помощью атрибута `language` элемента `script` указывается используемый язык сценариев (`language="javascript"`). В строке 3 указан комментарий, который "скрывает" сценарий от старых версий броузеров, которые не поддерживают язык JavaScript. В строках 4–6 создается объект документ, а также загружается документ XML, о чем подробно мы уже говорили. Строки 7, 8 и 9 завершают комментарий, сценарий и раздел `<head>` соответственно.

Приведенный фрагмент кода только загружает документ и использует его узлы при дальнейшей обработке. Предположим, вам необходимо отобразить текстовое содержание документа XML на экране, используя метод `alert`. Для этого просто добавьте к коду этот метод в раздел `<script>`. Полный вариант HTML-страницы приведен в листинге 12.2. Создайте эту страницу, сохраните под именем `DOM01.html`, после чего загрузите в Microsoft Internet Explorer 5.0 или более поздней версии.

Листинг 12.2. Пример сценария DOM DOM01.html

```
1:  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
2: <!-- листинг 12.2 - DOM01.html -->
3:
4: <html>
5: <head>
6: <title>DOM Scripting</title>
7: </head>
8: <script language="javascript">
9: <!-- hide script from older browsers
10:var oMystuff = new ActiveXObject("Microsoft.XMLDOM")
11:oMystuff.async="false"
12:oMystuff.load("message01.xml")
13:alert(oMystuff.text);
14:-->
15:</script>
16:</head>
17:<body>
18:Done!
19:</body>
20:</html>
```



С основной частью компонентов этого документа вы уже познакомились раньше. Строки 8–15 содержат сценарий, который создает объект DOM (строка 10) и загружает документ XML (строка 12). В строке 13 указан метод `alert` JavaScript. При выполнении этого метода обработка HTML-страницы прекращается, на экране отображается сообщение с текстовым содержанием объекта `oMystuff`. В данном случае объект содержит полный экземпляр документа, поэтому в сообщении будут представлены все текстовые данные из всех дочерних узлов. В строке 18 содержится текст `Done!` (`Готово!`), который отображается после щелчка мышью на кнопке OK в окне сообщения.

Если вы проверяете эту HTML-страницу с помощью Microsoft Internet Explorer, окно с сообщением должно выглядеть так, как показано на рис. 12.2. В этом окне перечислены все текстовые данные из документа XML, что соответствует строкам 6, 9 и 12 из листинга 12.1.

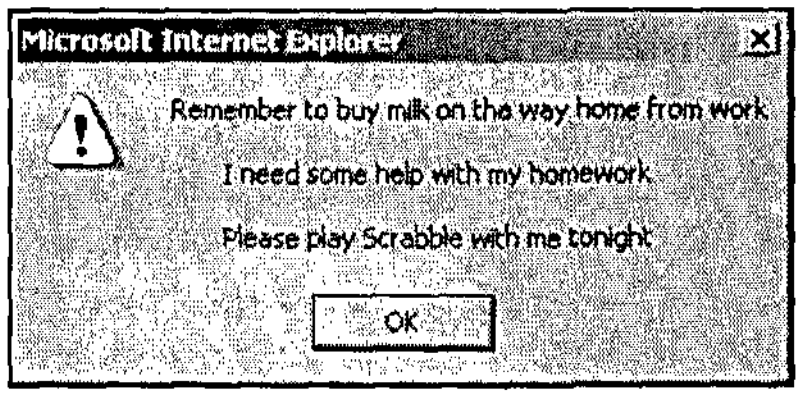


Рис. 12.2. Сообщение, отображенное с помощью метода `alert`, содержит все текстовые данные из экземпляра документа XML

Отображение результатов на экране

Предположим, вам необходимо возвратить результаты HTML-странице, а не представить их с помощью метода `alert`. Вы можете сделать это одним из нескольких способов. Например, вы можете удалить метод `alert` и указать объект в теле HTML-страницы, используя метод `document.write` JavaScript в отдельном сценарии. Используя этот метод, вы можете создать второй сценарий в теле HTML-страницы. Один из возможных вариантов решения этой задачи приведен в листинге 12.3. Как видите, этот

подход работает, но оказывается не слишком эффективным и практичным. Измените созданную раньше HTML-страницу и сохраните ее под именем DOM02.html.

Листинг 12.3. Передача текстовых данных из документа XML в документ HTML
DOM02.html

```
1: <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
2: <!-- листинг 12.3 - DOM02.html -->
3:
4: <html>
5: <head>
6:  <title>DOM Scripting</title>
7:  <script language="javascript">
8:  <!-- hide script from older browsers
9:  var oMystuff = new ActiveXObject("Microsoft.XMLDOM")
10:  oMystuff.async="false"
11:  oMystuff.load("message01.xml")
12:  -->
13:  </script>
14:  </head>
15:  <body>
16:  <script language="javascript">
17:  <!-- hide script from older browsers
18:     document.write
19:     ("<h2>The text content of the XML file contains:</h2>")
20:
21:     document.write
22:     (oMystuff.text)
23:
24:     document.write
25:     ("<hr />")
26:  -->
27:  </script>
28:  </body>
29:  </html>
```



Первый сценарий в этом листинге (строки 7–13) подобен сценарию из листинга 12.2, за исключением того, что метод `alert` опущен. Второй сценарий, созданный в теле HTML-страницы, содержится в строках 16–27. Строки 18–19 отображают на HTML-странице строку `<h2>The text content of the XML file contains:</h2>`. Поскольку эта строка содержит знак разметки HTML (`H2`), она будет отображаться с использованием стиля заголовка второго уровня. Строки 21 и 22 отображают результат использования свойства `text` объекта `oMystuff` (например, все текстовое содержание узла документа и его дочерних узлов). Строки 24 и 25 просто добавляют горизонтальную линию в окне броузера. Строка 27 завершает второй сценарий.

Выбор узла по его расположению

Предположим, что вам необходимо представить только текстовое содержание второго дочернего узла по отношению к корневому узлу документа. Быстрый анализ документа XML (см. листинг 12.1) показывает, что таковым является второй элемент `message`. Этот элемент содержит строку `I need some help with my homework`. Для того чтобы указать узел по его месторасположению, вам необходимо использовать свойства, которые содержат цифровые данные. В этом упражнении вам на самом деле необходимо возвратить тексто-

вое содержание второго дочернего узла по отношению к корневому узлу документу. При использовании JavaScript это реализуется с помощью следующего выражения:

```
oMystuff.document.documentElement.childNodes.item(1).text
```

Обратите внимание, что для выбора второго дочернего узла вам необходимо указать значение свойства `item`, равное `1`. Это связано с тем, что перечисление значений свойств в JavaScript начинается с нуля для указания первого экземпляра, после чего следует значение `1`, `2`, `3` и т.д.

В этом примере вы также можете объединить два предыдущих сценария в один. Для простоты реализации поместите объединенный сценарий в раздел `<body>` HTML-страницы. Полный вариант HTML-страницы, сохраненный под именем `DOM03.html`, приведен в листинге 12.4.

Листинг 12.4. Передача текстовых данных из узла `DOM03.html`

```
1: <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
2: <!-- листинг 12.4 - DOM03.html -->
3:
4: <html>
5: <head>
6: <title>DOM Scripting</title>
7: </head>
8:  <body>
9:  <script language="javascript">
10:  <!--
11:
12:     var oMystuff = new ActiveXObject("Microsoft.XMLDOM")
13:     oMystuff.async="false"
14:     oMystuff.load("message01.xml")
15:
16:     document.write
17:     ("<h2>The selected XML element in the file contains:</h2>")
18:
19:     document.write
20:     (oMystuff.documentElement.childNodes.item(1).text)
21:
22:     document.write
23:     ("<hr />")
24:
25: -->
26: </script>
27:
28: </body>
29: </html>
```

АНАЛИЗ Сценарий, используемый для загрузки и обработки документа XML содержится в строках 9–26 этого листинга. Строки 12–14 инициализируют объект документа, а также загружают документ XML. Строки 19–20 содержат выбор текстового содержания второго дочернего узла.

Выбор элементов XML по имени

Предположим, вы располагаете некоторыми сведения о структуре документа XML, к которому вы хотите обратиться с помощью модели DOM; кроме того, вам необходимо выбрать текстовое содержание определенного элемента, например второй элемент `message` в документе XML `message01.xml`. Для решения подобной задачи вы

можете воспользоваться методом `getElementsByTagName`. Для описанного сценария обратитесь к следующему коду:

```
oMystuff.getElementsByTagName("message").item(1).text
```

Это выражение возвращает текстовое содержание второго (`item(1)`) элемента `message` в объекте `oMystuff`.

Используя метод `document.write`, добавьте эту строку в код HTML-страницы, сохранив результат под именем `DOM04.html`. Работающий вариант HTML-страницы приведен в листинге 12.5.

Листинг 12.5. Выбор элемента XML по имени `DOM04.html`

```
1: <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
2: <!-- листинг 12.5 - DOM04.html -->
3:
4: <html>
5: <head>
6: <title>DOM Scripting</title>
7: </head>
8:  <body>
9:  <script language="javascript">
10:  <!--
11:
12:     var oMystuff = new ActiveXObject("Microsoft.XMLDOM")
13:     oMystuff.async="false"
14:     oMystuff.load("message01.xml")
15:
16:     document.write
17:     ("<h2>The selected XML element in the file contains:</h2>")
18:
19:     document.write
20:     (oMystuff.documentElement.childNodes.item(1).text)
21:
22:   document.write
23:   ("<br/><br/> Result by Tag Name:<br/>")
24:
25:   document.write
26:   (oMystuff.getElementsByTagName("message").item(1).text)
27:
28:     document.write
29:     ("<hr />")
30:
31:  -->
32:  </script>
33:
34:  </body>
35:  </html>
```



Строки 25–26 добавляют в сценарий новый метод. Код возвращает текст, который содержится во втором элементе `message`. Строки 22–23 добавлены только для разделения полученных результатов.

Сообщения об ошибках DOM

Специальное свойство в модели DOM (`parseError`) содержит и возвращает объект, где приведены сведения о последней ошибке обработки. Используя те или иные свойства этого объекта, вы создаете процедуру для проверки документов XML на наличие ошибок. Свойства объекта `XMLDOMParseError`, который содержится в свойстве `parseError` объекта `XMLDOMDocument`, перечислены в табл. 12.5.

Таблица 12.5. Свойства объекта `XMLDOMParseError`

Свойство	*Описание*
`errorCode`	Содержит код последней ошибки обработки
`filePos`	Содержит точное расположение ошибки в файле
`line`	Возвращает номер строки, при обработке которой возникла ошибка
`linePos`	Содержит точное расположение неправильного элемента строки, при обработке которой возникла ошибка
`reason`	Содержит описание ошибки
`scrText`	Содержит полный текст строки, содержащей ошибку
`url`	Содержит адрес URL документа XML, содержащего ошибку

Предположим, вам необходимо написать процедуру JavaScript для проверки документа XML. Обнаружив ошибку, необходимо вернуть значения свойств `errorCode`, `line`, `reason` и `url`. Используя JavaScript, вы можете использовать свойство `parseError` объекта документа, указав необходимые свойства `XMLDOMParseError`. Этот код выглядит следующим образом:

```
1: document.write
2:    ("<br>Error Reason Code: ")
3:    document.write
4:    (oMystuff.parseError.errorCode)
5:
6:    document.write
7:    ("<br>Error Line Number: ")
8:    document.write
9:    (oMystuff.parseError.line)
10:
11:    document.write
12:   ("<br>Error Reason Description: ")
13:    document.write
14:   (oMystuff.parseError.reason)
15:
16:    document.write
17:   ("<br>URL of File with Error: ")
18:    document.write
19:   (oMystuff.parseError.url)
```



Строки 3 и 4 возвращают свойство `errorCode`, если при загрузке документа XML в память возникнет ошибка обработки. Строки 8 и9 возвращают номер строки; строки 13 и 14 возвращают описание причины ошибки, а строки 18 и 19 возвращают адрес URL документа XML.

Теперь добавим эти свойства в сценарий для проверки документа на наличие ошибок обработки. Для того чтобы вызвать появление ошибки, измените метод загрузки документа в сценарии, чтобы он искал несуществующий документ XML. При выполнении сценария он сообщит об ошибке.

В листинге 12.6 приведена HTML-страница, содержащая полный вариант сценария. Создайте свой сценарии в соответствии с приведенным примером и сохраните его под именем `DOMerror01.html`.

Листинг 12.6. Сценарий, содержащий ошибки, `DOMerror01.html`

```
1: <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
2: <!-- листинг 12.6 - DOMerror01.html -->
3:
4: <html>
5: <head>
6: <title>DOM Scripting</title>
7: </head>
8: <body>
9: <script language="javascript">
10:<!--
11:
12:   var oMystuff = new ActiveXObject("Microsoft.XMLDOM")
13:   oMystuff.async="false"
14:   oMystuff.load("wrong_filename.xml")
15:
16:   document.write
17:  ("<br>Error Reason Code: ")
18:   document.write
19:  (oMystuff.parseError.errorCode)
20:
21:   document.write
22:  ("<br>Error Line Number: ")
23:   document.write
24:  (oMystuff.parseError.line)
25:
26:   document.write
27:  ("<br>Error Reason Description: ")
28:   document.write
29:  (oMystuff.parseError.reason)
30:
31:   document.write
32:  ("<br>URL of File with Error: ")
33:   document.write
34:  (oMystuff.parseError.url)
35:
36:-->
37:</script>
38:</body>
39:</html>
```



Процедура выявления ошибок, подробно описанная нами раньше, в приведенном листинге находится в строках 16–34. В строке 14 (`oMystuff.load("wrong_filename.xml")`) предпринята попытка загрузить несуществующий документ XML. Это приводит к возникновению ошибки выполнения, отображаемой в окне броузера в следующем виде:

```
Error Reason Code: -2146697210
Error Line Number: 0
Error Reason Description: The system cannot locate the object
specified.
URL of File with Error: wrong_filename.xml
```

Сообщения об ошибках, предоставляемые объектом XMLDOMParseError, обычно достаточно информативны. Для того чтобы увидеть еще одно сообщение об ошибке, измените документ message01.xml, использовав символы разного регистра в дескрипторах одного из элементов message, после чего сохраните результат под именем message_bad.xml. В листинге 12.7 приведен пример неправильно отформатированного экземпляра XML.

Листинг 12.7. Неправильно отформатированный экземпляр XML message_bad.xml

```
1: <?xml version = "1.0"?>
2:  <!-- листинг 12.1 - message01.xml -->
3:
4:  <note>
5:      <message ID="m1" from="Kathy Shepherd">
6:         Remember to buy milk on the way home from work
7:      </message>
8:      <message ID="m2" from="Greg Shepherd">
9:         I need some help with my homework
10:      </message>
11:      <Message ID="m3" from="Kristen Shepherd">
12:         Please play Scrabble with me tonight
13:      </message>
14: </note>
```

АНАЛИЗ

Третий элемент message (строки 11–13) начинается с дескриптора с первой прописной буквой M, а закачивается дескриптором с первой строчной буквой m. Это приведет к ошибке выполнения при загрузке документа в объект документа DOM.

Для того чтобы увидеть отчет об обработке документа, приведенного в листинге 12.7, измените строку 14 в документе DOMerror01.html таким образом, чтобы в ней предпринималась попытка загрузки документа message_bad.xml. После этого строка должна иметь следующий вид:

```
14: oMystuff.load("message_bad.xml")
```

После выполнения сценария вы увидите в окне броузера следующее сообщение об ошибке:

```
Error Reason Code: -1072896659
Error Line Number: 13
Error Reason Description: End tag 'message' does not match the start
tag 'Message'.
URL of File with Error: message_bad.xml
```

Теперь вы точно будете знать, какие действия необходимо предпринять для исправления ошибки. Подобными средствами выявления ошибок обладают практически все современные синтаксические анализаторы XML.

Резюме

В настоящем уроке вы познакомились с объектной моделью документа DOM для XML. Вы узнали, что эта модель предоставляет вам программный интерфейс API, который используется различными языками сценариев и разработки приложений. Вы узнали, ка-

ким образом модель DOM представляет содержание документа XML в виде иерархической структуры узлов. И наконец, вы написали несколько сценариев, использующих различные свойства и методы объекта документа для обнаружения узлов в документе XML и сообщения об ошибках обработки. Завтра вы познакомитесь с еще одним подходом к решению подобных задач, который базируется на использовании SAX (Simple API for XML) — управляемой событиями моделью обработки документов XML.

Вопросы и ответы

Что такое объектная модель?

В программировании объектная модель, например DOM, — это группа связанных объектов, которая позволяет выполнять определенный ряд задач. Благодаря связыванию с объектами методов, свойств и событий становится возможной разработка интерфейсов, которые называются прикладными интерфейсами API (Application Programming Interface). Объектная модель DOM предлагает интерфейс API, который не зависит от используемых при его реализации платформ и языков программирования.

Каким образом программист может использовать модель DOM?

Вы создаете экземпляр синтаксического анализатора XML, инициализируя объект DOM. Подход *Microsoft*, использованный нами при выполнении упражнений в настоящем уроке, представляет XML DOM в виде набора стандартных COM-компонентов синтаксического анализатора MSXML, поставляемого вместе с Microsoft Internet Explorer 5.0 или более поздней версии.

В каких случаях сценарии, использующие модель DOM, должны выполняться на стороне клиента, а в каких — на стороне сервера?

Приложения DOM, которые выполняются на стороне клиента, оказываются особенно полезными в тех случаях, когда вам необходима проверка документов XML. Об обработке DOM на стороне сервера стоит подумать при выполнении приложений без участия пользователя. В подобных ситуациях контроль над выполнением сценария обычно выполняется с помощью сценария. Таким образом, вы можете построить Web-узел, который для управления экземплярами документов XML будет использовать объектную модель DOM.

Упражнение

Проверьте свои знания, полученные в результате изучения настоящего урока. Ответ вы найдете в Приложении А "Ответы на вопросы".

Напишите сценарий, который будет инициализаровать объектную модель DOM. Загрузите документ `cd.xml` в объект документ и отобразите в окне броузера содержание элементов `title` и `artist` для первого компакт-диска в вашей коллекции.

Используйте метод `getElementsByTagName()`, а также соответствующие метод `item()` и свойство `test` для получения соответствующих значений.

Упражнение 2

Измените код таким образом, чтобы в окне броузера отображалось содержание элементов `title` и `artist` для второго компакт-диска в вашей коллекции.

День 13-й

Simple API для XML

SAX (Simple API for XML) — это открытый стандарт для обработки документов XML, базирующийся на событиях. В отличие от подхода с использованием объектной модели документа DOM, с которым вы познакомились в уроке дня 12-го, когда создается древовидное представление информации ваших документов XML, SAX не содержит собственной стандартной объектной модели. SAX определяет абстрактный программный интерфейс, позволяющий моделировать экземпляр документа XML линейной последовательностью вызовов методов.

Сегодня рассмотрим следующие темы.

- Различия между прикладными интерфейсами API стандартов DOM и SAX; когда и какой следует использовать.
- Каким образом основанный на событиях прикладной интерфейс API работает с иерархическими данными XML.
- Как написать простую Java-программу для вызова синтаксического анализатора JAXP компании *Sun Microsystems* на базе SAX.

Что такое SAX

Стандарт SAX 1.0 (Simple API for XML) представлен 11 мая 1998 года. SAX — это основанный на событиях прикладной интерфейс API для обработки документов XML. Этот стандарт разработан группой обсуждения XML-DEV. Новая версия SAX 2.0, обеспечивает поддержку пространств имен, а также более сложных строк запросов. В настоящем уроке вы познакомитесь с SAX 1.0 и будете использовать его для обработки документов XML. Подробные исторические сведения о работе над этим проектом вы найдете в Internet по адресу `http://www.megginson.com/SAX/`.

Как и DOM, SAX предоставляет стандартный набор методов для программного манипулирования данными и структурным содержанием экземпляра документа XML. SAX — это еще один способ обработки частей документа XML и доступа к ним. Во время обработки документа каждая его часть идентифицируется, после чего вызывается соответствующее событие. Такой подход значительно отличает SAX от прикладного интерфейса модели DOM. Как вы узнали из предыдущей главы, DOM считывает в память весь экземпляр документа за раз. SAX обрабатывает каждый элемент разметки как отдельное событие, не требуя загрузки всего документа в память. И это далеко не единственное отличие SAX от прикладного интерфейса модели DOM. Например прикладной интерфейс модели DOM

поддерживает возможность добавления и удаления узлов. На самом деле, с помощью прикладного интерфейса модели DOM вы вообще можете создать целый документ XML с нуля. Однако вы не сможете этого сделать с помощью SAX. В настоящей главе вы подробно познакомитесь с различиями между двумя этими подходами.

Синтаксические анализаторы SAX

Существует достаточно много синтаксических анализаторов SAX, написанных на различных языках программирования, таких как Java, Python, PERL или Visual Basic, каждый из которых позволяет реализовать все основные методы прикладного интерфейса SAX. Изначально разработка SAX велась с применением языка Java, поэтому значительная часть реализаций выполнена именно на этом языке программирования. Наиболее популярные синтаксические анализаторы SAX перечислены в табл. 13.1.

Таблица 13.1. Популярные синтаксические анализаторы SAX

Синтаксический анализатор	*Описание*
MSXML 3.0	Этот анализатор компании *Microsoft* поддерживает стандарты SAX и DOM, а также ряд других технологий XML. Доступен для загрузки с Web-узла `http://msdn.microsoft.com/xml`
MSXML 4.0	MSXML 4 от компании *Microsoft* обладает улучшенной поддержкой технологий XML, включая интеграциюSAX и Dom, что позволяет вызывать события SAX объекта `DOMobject`, а также создавать объекты SAX на основе узлов DOM. Синтаксический анализатор MSXML 3.0 доступен для загрузки с Web-узла `http://msdn.microsoft.com/xml`
Xerces	Это программный продукт, разработанный организацией *Apache Software Foundation*; очень эффективный синтаксический анализатор, поддерживающий стандарты SAX и DOM. Доступен для загрузки с Web-узла `http://www.apache.org`. Xerces платформно-независим, поскольку написан на Java и распространяется бесплатно
JAXP	Это синтаксический анализатор, разработанный компанией *Sun Microsystems*; поддерживает стандарты SAX и DOM. Доступен для загрузки с Web-узла `http://java.sun.com/xml`. JAXP написан на Java, что делает его платформно-независимым

Сравнение SAX и DOM

Вы должны помнить, что прикладной интерфейс DOM API создает древовидную модель, которая сохраняет все данные документа XML в виде иерархии узлов. Экземпляр документа целиком загружается в память, что дает произвольный доступ к любому узлу дерева документа. DOM позволяет добавлять и удалять узлы, т.е. вносить изменения в документ программно.

Обычно при создании базирующегося на событиях приложения вы создаете функции, которые вызываются пользовательскими событиями или транзакциями. Например, сценарий может ожидать события `onClick` для вызова определенной процедуры. Написание приложения, вызывающего синтаксический анализатор SAX, который зависит от событий, требует применения такого же подхода; однако именно синтакси-

ческий анализатор, а не пользователь, приводит к возникновению событий. Синтаксические анализаторы SAX вызывают методы после возникновения событий разметки при обработке документа. Части вашего документа XML (начало и конец, открывающие и закрывающие дескрипторы, атрибуты и текстовые данные) вызывают события, которые распознаются синтаксическим анализатором SAX при обработке документа.

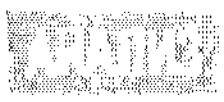
Синтаксический анализатор проектируется таким образом, чтобы считывать и распознавать уникальные знаки разметки в потоке данных, образующих документ XML. Рассмотрите простой документ XML, приведенный в листинге 13.1, который будет передан синтаксическому анализатору SAX.

Листинг 13.1. Простой документ XML `message01.xml`

```
1: <?xml version = "1.0"?>
2: <!-- листинг 13.1 - message01.xml -->
3: <note>
4:     <message from="Kathy Shepherd">
5:        Remember to buy milk on the way home from work
6:     </message>
7: </note>
```

К событиям, которые можно получить синтаксическим анализатором SAX, относятся следующие:

```
1: Начало документа
2: Начало элемента (note)
3: Начало элемента (message)
4: Пара имя-значение атрибута (from="Kathy Shepherd")
5: Текстовые данные (Remember to buy milk on the way home from work)
6: Конец элемента (message)
7: Конец элемента (note)
8: Конец документа
```



В строке 1 указано событие, соответствующее началу экземпляра документа, генерируемое синтаксическим анализатором SAX при обработке документа. В строке 2 указано событие, соответствующее началу элемента `note`. Закрывающий дескриптор элемента `note` приводит к возникновению события завершения элемента, указанного в строке 7. Строки 3 и 6 соответствуют открывающему и закрывающему дескрипторам элемента `message`. Строка 4 содержит событие, соответствующее событию `from` элемента `message`. Событие, связанное с обработкой текстового содержания элемента `message`, показано в строке 5.

На самом деле, с простым экземпляром документа XML, приведенным в листинге 13.1, связан еще целый ряд потенциальных событий, не перечисленных выше. Например, SAX обходится со знаками пробела совсем не так, как это делает DOM. Подробнее об этом вы узнаете дальше в настоящем уроке.

Подходы с использованием SAX и DOM очевидно отличаются между собой и по тому, как они представляют экземпляр документа XML. При использовании базирующегося на событиях SAX в памяти не создается никакой древовидной структуры, соответствующей экземпляру документа. Вместо этого документ обрабатывается последовательно, символ за символом. Поскольку SAX не требует представления в памяти целого документа, в некоторых ситуациях он оказывается намного предпочтительнее, чем DOM. Например, при использовании SAX можно проводить поиск самых малых фрагментов информации в очень больших экземплярах документов. Кроме того, SAX позволяет вам прерывать обработку после нахождения необходимых сведений, не требуя при этом загрузки в память документа для его обработки. SAX можно

использовать для построения деревьев DOM. Именно на этом принципе построена бо́льшая часть синтаксических анализаторов DOM. В результате вы получаете возможность использовать как деревья DOM, так и потоки SAX; однако подобные сложные операции выходят за рамки рассмотрения настоящего урока.

Поскольку данные передаются приложению SAX из документа XML, вы можете ожидать более высокой производительности, а также меньшего обращения к памяти. Однако почти все преимущества высокой производительности сводятся практически на нет более сложными структурами обработки. Сложные запросы, выполняемые в документах, с большим количеством уровней вложения дочерних элементов, с очень сложными межэлементными отношениями, могут стать слишком обременительными из-за сложностей, связанных с обработкой событий. А следовательно, многие разработчики предпочитают использовать DOM для документов, требующих частых модификаций, но используют синтаксический анализатор SAX для считывания очень длинных элементов, которые не претерпевают никаких изменений. Растущая популярность XML приводит к возникновению различных комбинаций SAX и DOM, упрощающих выполнение различных задач.

Выбор SAX вместо DOM

Основанный на событиях синтаксический анализатор обрабатывает документы последовательно, поэтому в ряде случаев стандарт Simple API for XML оказывается намного предпочтительнее, чем стандарт Document Object Model.

Обработка больших документов

Одно из ключевых преимуществ SAX заключается в том, что для его работы требуется намного меньше памяти для обработки документа XML, чем при использовании синтаксического анализатора XML. При работе SAX требуемый объем памяти не возрастает вместе с увеличением размера обрабатываемого файла. Например, при обработке документа объемом в 100 Кбайт с помощью синтаксического анализатора DOM необходимо около 1Мбайт памяти; при использовании для обработки этого же документа с помощью синтаксического анализатора SAX необходимо намного меньше памяти. Если вы будете работать с большими документами, использование синтаксического анализатора SAX кажется наиболее разумным, особенно если вам не нужно изменять содержание документа.

Программное прерывание обработки документа

Поскольку SAX позволяет вам прервать обработку документа в любое время, вы можете использовать SAX при создании приложений, предназначенных для нахождения и извлечения определенных данных. Например, вы можете создать приложение для поиска определенного элемента в каталоге. При нахождении этого элемента приложение вернет связанные с ним данные, такие как код и состояние доступности, после чего прекратит дальнейшую обработку.

Извлечение небольших фрагментов информации

Для многих решений на базе XML совсем необязательно считывать весь документ целиком для получения необходимых результатов. Например, вы можете представить себе приложение системы новостей, которое просматривает базу данных для поиска статей, касающихся определенной темы; при этом вряд ли следует загружать в память все ненужные данные. При использовании SAX ваше приложение может найти данные на определенную тему, после чего создать соответствующую структуру документа и передать ее вашему приложению для чтения новостей, Web-броузеру или, возможно, беспроводному устройству. Поиск только небольшого фрагмента информации позволяет значительно сэкономить системные ресурсы.

Выбор DOM вместо SAX

В определенных ситуациях использование DOM оказывается предпочтительнее.

Произвольный доступ к документу

SAX не позволяет загрузить весь документ в память. Вы должны обрабатывать данные последовательно. Если вам необходим произвольный доступ к узлам, между которыми существуют достаточно сложные отношения, DOM предоставляет гораздо более подходящий выбор. При наличии в документе большого количества внутренних перекрестных ссылок SAX оказывается очень неудобным.

Сложные фильтры XPath

Используйте DOM в том случае, если вам необходима сложная фильтрация с использованием XPath для извлечения сложных структур данных, содержащих важные сведения. Древовидная структура DOM возвращает сведения со сложной структурой автоматически. При использовании SAX вам придется извлекать подобные сведения самостоятельно.

Создание и изменение XML

Как уже отмечалось, SAX не может создать новый экземпляр документа XML, чего нельзя сказать о DOM API. DOM позволяет достаточно легко создавать новые документы XML. SAX позволяет отслеживать события, но его нельзя использовать для построения экземпляра документа XML для дальнейших обработки, хранения или передачи. DOM позволяет вам изменять находящийся в памяти документ, а также считывать его из исходного файла XML. SAX предназначен для считывания, но не для записи документов XML. DOM намного лучше подходит для изменения документа XML в памяти и его сохранения.

Методы, вызываемые при обработке событий SAX

Несмотря на то, что некоторые распространенные методы относятся к подходу с использованием SAX, различные реализации вызывают дополнительные методы при возникновении определенных событий. Наиболее популярные методы представлены в табл. 13.2.

Таблица 13.2. Избранные методы синтаксического анализатора SAX

Метод	*Описание*
`documentLocator`	Метод, возвращающий имя файла, путь или URL обрабатываемого документа XML
`startDocument`	Метод, вызываемый при достижении обработчиком начала объекта `XMLdocument`
`endDocument`	Метод, вызываемый при достижении обработчиком конца объекта `XMLdocument`
`startElement`	Метод, вызываемый при достижении обработчиком открывающего дескриптора элемента
`endElement`	Метод, вызываемый при достижении обработчиком закрывающего дескриптора элемента
`characters`	Метод, вызываемый при достижении обработчиком текстовых данных или знаков пробела
`processing Instruction`	Метод, вызываемый при обнаружении инструкции обработки

SAX и Java

В декабре 1997 года Питер Мюррей-Раст (Peter Murray-Rust) впервые реализовал на Java то, что впоследствии превратилось в синтаксический анализатор SAX. Питер Мюррей-Раст — автор бесплатного, базирующегося на Java XML-броузера JUMBO. Мюррей-Раст начал с попытки поддержки трех различных синтаксических анализаторов с их собственными прикладными интерфейсами в одном броузере JUMBO. Он убедился, что это все ненужные дополнительные сложности, поэтому он решил, что разработчики синтаксических анализаторов должны обеспечить поддержку общего прикладного интерфейса API с кодовым названием YAXAPI (Yet Another XML Parser API). Членами группы рассылки XML-DEV являлись Тим Брей (Tim Bray), автор синтаксического анализатора XML Lark и один из авторов спецификации XML, Дэвид Меггинсон (David Megginson), автор синтаксического анализатора XML Microstar's Elfred, Джон Бозак (John Bosak), создатель XML, а также многие другие, приложившие немало усилий к разработке стандарта Simple API for XML, или просто SAX.

Значительную часть людей, задействованных в разработке стандарта SAX, составляли Java-программисты, поэтому то, что для реализации большинства методов использовался именно язык Java, не должно вызывать никакого удивления. В дальнейшем методы SAX были реализованы на PERL, Python, Visual Basic и других языках программирования. В следующем разделе вы узнаете, как можно вызывать методы событий при обработке документа XML, используя Java. Вы создадите и скомпилируете простое приложение Java для идентификации некоторых событий, возникающих при обработке документа XML. Обратите внимание, что в настоящей книге я не ставил перед собой задачу обучить вас программированию на языке Java или основным приемам объектно-ориентированного программирования. Если вы хотите получить подробные сведения о языке Java, обратите внимание на книгу *Sams Teach Yourself Java in 21 Days.* Вы также можете загрузить подробную документацию по Java с Web-узла компании *Sun Microsystems* по адресу `http://java.sun.com/j2se/1.3`.

Java предлагает процедурный, объектно-ориентированный (характеризуемый такими понятиями, как классы, инкапсуляция и объекты) подход к программированию. В настоящем разделе вы напишете Java-программу, имеющую дело с событиями SAX как с объектами и позволяющую повторно использовать программные компоненты из стандартной библиотеки классов SAX.

Необходимое программное обеспечение для создания Java-программы

Вам потребуется загрузить и установить несколько программных пакетов для выполнения упражнений настоящего урока. Если вы еще не установили на своем компьютере стандартную версию Java версии 2, вы можете совершенно бесплатно загрузить ее с Web-узла компании *Sun Microsystems* `http://java.sun.com/j2se`. Установите загруженное программное обеспечение в соответствии с указаниями, которые вы найдете на том же Web-узле.

Вам также потребуется загрузить и извлечь из архива файлы SAX API. Здесь существует много различных вариантов, но вы будете использовать синтаксический анализатор JAXP от той же компании *Sun Microsystems*. Вы можете загрузить бесплатную копию JAXP по адресу `http://java.sun.com/xml/download.html`. Распакуйте загруженный дистрибутив на жесткий диск своего компьютера, после чего изучите содержимое папки. Вам необходимо определить переменную окружения `classpath` или использовать параметр компилятора `classpath` для указания на эту папку во время компиляции Java-программ.

Java-приложение для получения списка событий SAX

Написание Java-приложений во многом похоже на создание сложных структур из простых блоков. Блоки могут быть различной формы и цвета. Вы получите строго определенные результаты, комбинируя качества отдельных компонентов. Java обеспечивает такие же возможности, поскольку вы можете использовать методы и интерфейсы каждого из созданных или импортированных классов для создания структуры объектов, поведение которых полностью предсказуемо. Для приложения, которое вы напишете, выполняя упражнение, импортируйте классы и интерфейсы из нескольких загруженных пакетов. Вы создадите вызовы методов синтаксического анализатора SAX, подобных тем, что перечислены в табл. 13.2. Все методы, которые вам предстоит создать, показаны на рис. 13.1.

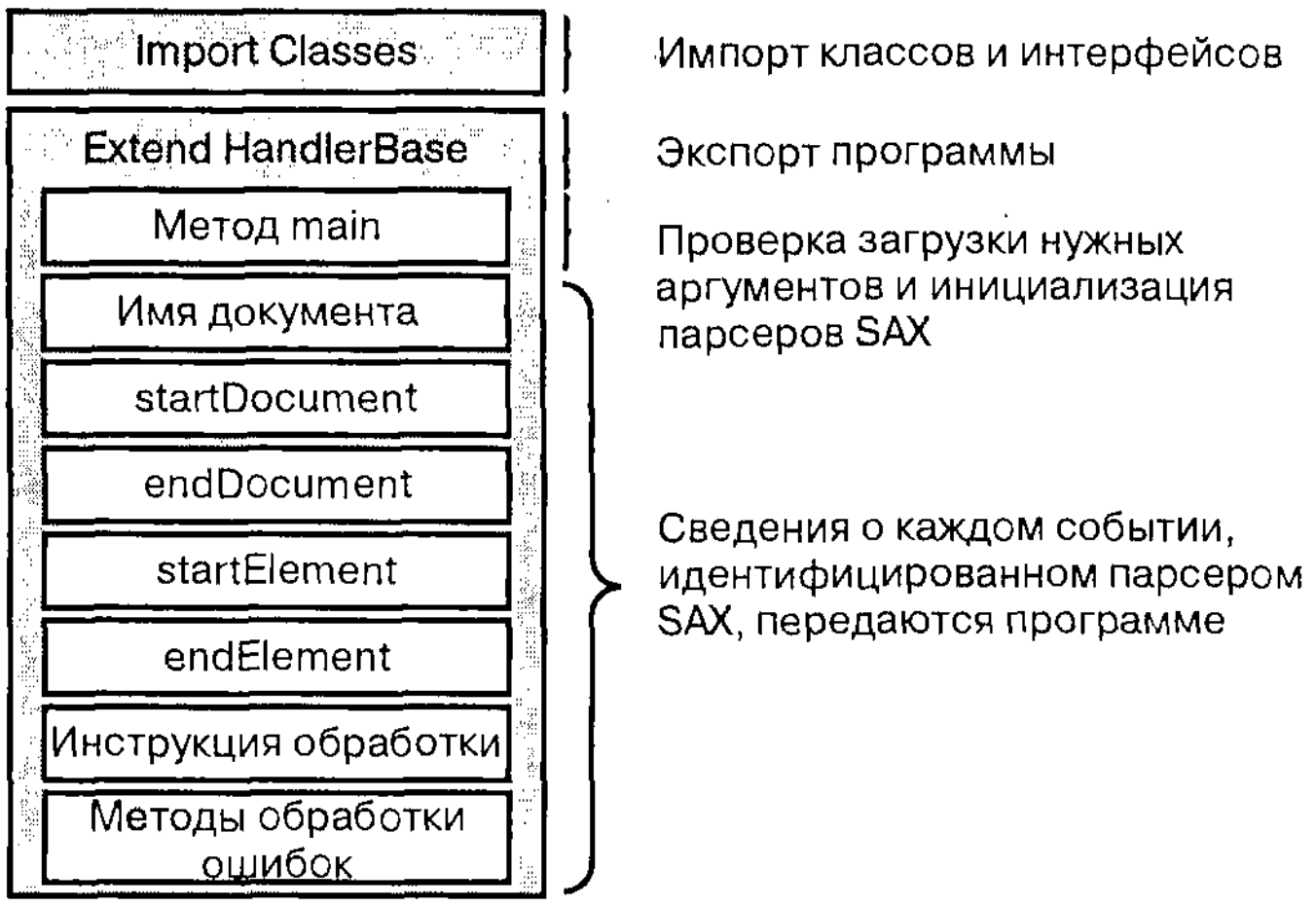


Рис. 13.1. Структура Java-приложения EList

Вы можете использовать текстовый редактор и создавать приложение по мере изучения материала настоящего урока. В конце раздела приведен полный вариант класса `Elist`, а также комментарии к каждому из вызываемых методов.

Импортирование классов

Вам необходимо импортировать пакет Java Input/Output (I/O), который позволит вашей программе управлять данными через потоки ввода-вывода. Для этого импортируйте пакет `java.io` из дистрибутива Java, загруженного вами с Web-узла компании *Sun Microsystems*. Классы из этого пакета могут использоваться для обработки файлов и работы в сети, а также предоставляют стандартный набор интерфейсов для других типов ввода-вывода. Для импортирования пакета используется ключевое слов `import`, как показано ниже:

```
import [пакет]
```

Как и все зарезервированные ключевые слова Java, ключевое слово `import` должно указываться только буквами нижнего регистра. В дополнение к пакету `java.io` вам также потребуется пакет, содержащий классы и программные интерфейсы для синтаксических анализаторов SAX, `org.xml.sax`. Сюда входит класс `HandlerBase`, который вы расширите при создании приложения.

Для инициализации классов для синтаксического анализатора SAX вы также должны импортировать пакет `javax.xml.parsers`. Этот пакет содержит классы для синтаксических анализаторов SAX и DOM, которые будут использоваться вашим приложением.

Кроме того, вам потребуется пакет `javax.xml.parsers.SAXParserFactory` — для инициализации синтаксического анализатора для проверки документов с использованием схем DTD; `javax.xml.parsers.ParserConfigurationException` — для обработки ошибки в том случае, если синтаксический анализатор не удалось должным образом инициализировать; `javax.xml.parsers.SAXParser` — для инициализации объекта синтаксического анализатора SAX. Полный блок операций импорта в вашем приложении будет выглядеть следующим образом:

```
import java.io.*;
import org.xml.sax.*;
import javax.xml.parsers.SAXParserFactory
import javax.xml.parsers.ParserConfigurationException
import javax.xml.parsers.SAXParser
```

Расширение класса HandlerBase

Java-программа, которую вы напишете, расширит класс `HandlerBase` из пакета `org.xml.sax` реализации SAX 1. В реализации SAX 2 этот класс заменен на класс `DefaultHandler`. Соответственно при использовании класса `DefaultHandler` вам потребуется импортировать пакет `org.xml.sax.helpers. DefaultHandler`; в остальном обе реализации совершенно идентичны. Для обучения мы импортируем классы и добавим расширения к классу `HandlerBase`. В этом упражнении вы завершите класс и назовете его `EList` (Event List — список событий); он предназначен для получения списка событий SAX. Вызывать собственный класс необязательно, однако вам нужно познакомиться со структурой Java-программы, сохранить ее с расширением `.java`. По общему соглашению все имена классов, а значит, и идентификаторы классов Java начинаются с букв верхнего регистра, причем прописные буквы ставят в начале каждого слова в имени класса (например, `MyOwnPersonalClassName`). Инструкция для пользовательского расширения основного класса выглядит следующим образом:

```
public class EList extends HandlerBase
{
    [...все методы определяемого класса]
}
```

Все методы определенного пользователем класса должны быть в фигурных скобках и следовать после объявления класса `EList`. Класс `HandlerBase` реализует четыре интерфейса из пакета `org.xml.sax`. Если вы используете класс `DefaultHandler` из реализации SAX 2, будут импортированы несколько других интерфейсов. Все эти интерфейсы перечислены в табл. 13.3.

Таблица 13.3. Несколько интерфейсов, реализуемых классом `HandlerBase` из пакета `org.xml.sax`

Интерфейс	*Описание*
DocumentHandler	Обрабатывает события по мере их появления во время обработки документа
DTDHandler	Обрабатывает ссылки, определенные в соответствующей схеме DTD (Document Type Definition)
EntityReference	Обрабатывает любые внешние ссылки
ErrorHandler	Обрабатывает любые ошибки, возникшие при обработке ошибок

Метод main

Первый метод, который должен выполняться в Java-приложении, — это метод `main`, один из обязательных методов Java. Метод `main` вашего приложения будет выполнять несколько необходимых функций. Сначала он потребует от пользователя указать имя файла для экземпляра документа XML, а также прервет выполнение приложения и отобразит сообщение об ошибке, если файл не указан. Затем вы создадите метод, требующий указания в командной строке одного аргумента. Соответствующий фрагмент кода для метода `main` приведен ниже:

```
1: public static void main( String fname[] )
2:     {
3:         if ( fname.length != 1 ) {
4:            System.err.println( "Expected Syntax: java Elist
              [filename]\n" );
5:            System.exit( 1 );
6:         }
```



В строке 1 объявляется метод `main`, который требует указания строкового атрибута `frame`. Ключевое слово `static` указывает на то, что метод `main` является методом класса. Ключевое слово `void` указывает на то, что метод выполняет определенные задачи, но не возвращает никакой информации после их завершения. Все требования метода `main` базируются на синтаксисе Java. В строках 3–4 осуществляется проверка того, что в командной строке введен только один аргумент при вызове класса `EList`. Если синтаксис Java не выполняется, на экране отображается сообщение об ошибке (например, `"Expected Syntax: java EList [filename]\n"`), после чего дальнейшая обработка инструкций прекращается.

Для начала обработки экземпляра документа XML вам необходимо инициализаровать объект `SAXParseFactory` из пакета `javax.xml.parsers` для получения SAX-совместимого синтаксического анализатора. Вы передадите экземпляр документа XML, полученный по указанному пользователем адресу URL, а также экземпляр класса `EList` SAX-совместимому синтаксическому анализатору благодаря вызову метода `newSAXParser`. В свою очередь, синтаксический анализатор SAX считает экземпляр документа XML символ за символом, вызывая методы SAX, указанные вами в классе `EList`. Вы узнаете, как указывать отдельные методы SAX очень быстро. Любые ошибки в классе `EList` будут приводить к немедленной остановке обработчика, а также отображению соответствующих сведений на экране.

Код, завершающий метод `main`, приведен ниже:

```
1: SAXParserFactory saxFactory = SAXParserFactory.newInstance.();
2:
3:  saxFactory.setValidating( false );
4:
5:         try {
6:            SAXParser saxParser = saxFactory.newSAXParser();
7:            saxParser.parse( new File( fname[ 0 ] ), new EList() );
8:         }
9:         catch ( SAXParseException spe ) {
10:           System.err.println( "Parse Error: " + spe.getMessage() );
11:        }
12:        catch ( SAXException spe ) {
13:           se.printStackTrace();
14:        }
15:        catch ( ParserConfigurationException pce ) {
```

```
16:            pce.printStackTrace();
17:         }
18:         catch ( IOException ioe ) {
19:            ioe.printStackTrace();
20:         }
21:
22:         System.exit( 0 );
23:      }
```



В строке 1 инициализируется объект `SAXParseFactory` из импортированного раньше пакета `javax.xml.parsers`. В результате программа получает синтаксический анализатор SAX, поддерживающий обработку отдельных событий. В строке 3 указана специальная инструкция, которая используется для настройки объекта `SAXParseFactory` как синтаксического анализатора без поддержки проверки документов. Измените эту инструкцию, Если вы хотите, чтобы синтаксический анализатор в состоянии проводил проверку документа XML с помощью связанной с ним схемы DTD. Строки 6 и 7 инициализируют объект синтаксического анализатора SAX, вызывая метод `newSAXParser`, после чего передают указанный пользователем экземпляр документа XML, а также экземпляр класса `EList` SAX-совместимому синтаксическому анализатору. Строки 9–19 отслеживают возникающие ошибки (например, серьезные синтаксические ошибки XML), ошибки SAX, ошибки синтаксического анализатора, а также ошибки ввода-вывода (например, связанные с недоступностью экземпляра документа XML по указанному адресу URL). Строка 22 вызывает прекращение выполнения приложения и выход в операционную систему при возникновении любой из этих ошибок.

Методы SAX

Первый метод в следующем разделе вашего приложения должен использовать метод `setDocumentLocator` для указания адреса URL обрабатываемого документа синтаксическому анализатору SAX. Это документ, адрес которого пользователь указал в командной строке. Создайте ссылку на адрес URL документа, назвав ее `fname`, после чего передайте ее методу `getSystemId`. Вы будете использовать стандартный объект вывода, `output.println` для того, чтобы проинструктировать обработчик о необходимости напечатать строку `Name of Document:`, после которой указывается имя документа, полученное с помощью метода `fname.getSystemId()`. Для того чтобы это сработало, вам необходимо создать следующий метод:

```
public void setDocumentLocator( Locator fname )
   {
      System.out.println( "Name of Document: " + fname.getSystemId()
);
   }
```

Затем вы проинструктируете синтаксический анализатор SAX сообщить о том, что он обнаружил начало документа. Для этого используйте метод `startDocument` из класса `HandlerBase`. Это событие происходит всего один раз для каждого экземпляра документа. Вы вызываете стандартный метод вывода для отчета о том, что синтаксический анализатор SAX столкнулся с корневым узлом документа. Отчет должен быть представлен в виде строки — на экране пользователя отобразится сообщение `SAX Event - Start of Document`. В каждом из созданных вами методов SAX вы будете использовать инструкцию `throws SAXException`, чтобы указать синтаксическому анализатору на необходи-

мость обработки ошибок. Вы должны помнить, что метод `main` включает инструкции для обнаружения ошибок, сообщения о них пользователю, а также прекращения дальнейшей обработки документа. Полный метод выглядит следующим образом:

```
public void startDocument()
     throws SAXException
   {
      System.out.println( "SAX Event - Start of Document" );
   }
```

Конец документа XML с точки зрения синтаксического анализатора SAC считается событием, поэтому метод можно включить в отчет при возникновении события. Это очень похоже на метод, использованный в случае начала документа, за исключением того, что используется метод `endDocument` из класса `HandlerBase`. Метод выглядит следующим образом:

```
public void endDocument()
     throws SAXException
   {
      System.out.println( "SAX Event - End of Document" );
   }
```

Затем вы будете использовать метод `startElement` из класса `HandlerBase` для определения того, что синтаксический анализатор сталкивается с открывающим дескриптором элемента в экземпляре документа XML. Метод `startElement` получает два аргумента, каждый из которых представляет собой определенную пользователем ссылку. Первый аргумент содержит имя элемента, а второй — список его атрибутов. Поскольку элемент может содержать нуль или несколько атрибутов, вам потребуется обеспечить механизм, который сообщит о необходимом количестве аргументов элемента. Затем вам необходимо написать цикл, который будет перебирать каждый атрибут и пересылать его вместе с его значением в качестве обычного объекта вывода.

В случае нескольких атрибутов элемента вам необходимо перебирать их до тех пор, пока не останется ни одного. Кроме того, вам следует определить формат вывода пар "имя–значение атрибута", используя в них знак равенства и двойные кавычки в каждой из управляющих последовательностей. Например, для того чтобы напечатать двойную кавычку, вам необходимо использовать управляющую инструкцию (\"). Обратная косая в строке Java представляет управляющий символ, запрещающий обработку интерпретатором Java символа, который следует сразу после него, благодаря чему он передается как часть строки вывода. Полный вариант метода `startElement` приведен ниже:

```
1: public void startElement( String el_name, AttributeList attr )
2:    throws SAXException
3:     {
4:        System.out.println( "SAX Event - Element Start: " + el_name );
5:        if ( attr != null )
6:             for ( int i = 0; i < attr.getLength(); i++ )
7:                System.out.println( "SAX Event - Attribute: "
8:                + attr.getName( i )
9:                + "=\"" + attr.getValue( i )
10:                + "\"" );
11:   }
```

Строка 1 вызывает метод `startElement` при обнаружении элемента синтаксическим анализатором SAX. Имя элемента, представляющее собой первый аргумент этого метода, указано как `el_name`, а список атрибутов указан как `attr`. Строка 2 касается ошибок обработки, если такие возникнут. Строки 3 и 11 указывают на начало и конец метода соответственно с помо-

щью фигурных скобок. Стандартное устройство вывода (строка 4) вызывается для того, чтобы сообщить о возникновении события SAX при вызове метода. Строки 5—10 определяют наличие атрибутов у элемента. Если какие-либо атрибуты отсутствуют, значение аргумента `attr` будет равно нулю, в связи с чем цикл `for` выполняться не будет. Во время выполнения сообщаются имя (`getName`) и значение (`getValue`) каждого из атрибутов.

Закрывающий дескриптор каждого из элементов также воспринимается синтаксическим анализатором SAX как событие, поэтому вы можете написать соответствующий метод. Этот метод подобен методам обнаружения начала и конца документа, за исключением того, что вы используете метод `endElement` класса `HandlerBase`:

```
public void endElement( String el_name )
     throws SAXException
   {
                System.out.println("SAX Event - Element End: " +
el_name );
     }
```

Обработка текстовых данных в экземпляре документа XML также считается событием SAX. Метод `characters` содержит переменные для буферов, значения смещения символов и длины строки. Вы объявите переменные `buf`, `offset` и `len` как переменные типов данных `char []`, `int` и `int` соответственно. Полностью определенная строка помещается в переменную `stuff` и представляется, как обычно, с помощью метода `System.out.println`. Описанный нами метод выглядит следующим образом:

```
public void characters( char buf [], int offset, int len )
     throws SAXException
   {
      if ( len > 0 ) {
         String stuff = new String( buf, offset, len );

         System.out.println( "SAX Event - Characters: " + stuff );
      }
   }
```

SAX также представляет как события и инструкции обработки. Метод `processingInstruction` принимает два аргумента, которые соответствуют названию системы и значению, передаваемому приложению инструкцией обработки. Например, инструкция обработки в XML может выглядеть следующим образом:

```
<?mesage_reader incoming transactions?>
```

Здесь название системы — `mesage_reader`, а значение — `incoming transactions`. Полный метод выглядит следующим образом:

```
public void processingInstruction( String part1,String part2 )
     throws SAXException
   {.
      System.out.println( "SAX Event - Processing Instruction: "
               + part1 + " " + part2 );
   }
}
```

Это последний метод, который вы включите в свою простую программу. Вам необходимо закрыть метод `main`, добавив закрывающую фигурную скобку в конце листинга. Полный текст программы приведен в листинге 13.2. Сохраните ее под именем `EList`.

Листинг 13.2. Пример класса Java `EList.java`

```
1: // Листинг 3.2 - eList.java
2:
3:  /*
4:    compile with javac, ensure the CLASSPATH includes the appropriate
5:    JAXP packages
6:  */
7:
8:  import java.io.*;
9:  import org.xml.sax.*;
10:  import javax.xml.parsers.SAXParserFactory;
11:  import javax.xml.parsers.ParserConfigurationException;
12:  import javax.xml.parsers.SAXParser;
13:
14:  public class EList extends HandlerBase
15:  {
16:     public static void main( String fname[] )
17:     {
18:        if ( fname.length != 1 ) {
19:          System.err.println( "Expected Syntax: java EList [filename]\n" );
20:          System.exit( 1 );
21:        }
22:
23:        SAXParserFactory saxFactory = SAXParserFactory.newInstance();
24:
25:        /*If you wish to have SAX validate an XML document against
26:            a DTD, you may change this value to true*/
27:        saxFactory.setValidating( false );
28:
29:        try {
30:           SAXParser saxParser = saxFactory.newSAXParser();
31:           saxParser.parse( new File( fname[ 0 ] ), new EList() );
32:        }
33:        catch ( SAXParseException spe ) {
34:           System.err.println( "Parse Error: " + spe.getMessage() );
35:        }
36:        catch ( SAXException se ) {
37:           se.printStackTrace();
38:        }
39:        catch ( ParserConfigurationException pce ) {
40:           pce.printStackTrace();
41:        }
42:        catch ( IOException ioe ) {
43:           ioe.printStackTrace();
44:        }
45:
46:        System.exit( 0 );
47:     }
48:
49:
50:     // print the document URL path and name
51:     public void setDocumentLocator( Locator fname )
52:     {
53:        System.out.println( "Name of Document: " + fname.getSystemId() );
54:     }
55:
56:     // SAX Event - Start of Document
```

```
57:    public void startDocument()
58:  throws SAXException
59:    {
60:       System.out.println( "SAX Event - Start of Document" );
61:    }
62:
63:    // SAX Event - End of Document
64:    public void endDocument()
65:  throws SAXException
66:    {
67:       System.out.println( "SAX Event - End of Document" );
68:    }
69:
70:    // SAX Events for Elements and Attributes
71:    public void startElement( String el_name, AttributeList attr )
72:  throws SAXException
73:    {
74:       System.out.println( "SAX Event - Element Start: " + el_name
);
75:       if ( attr != null )
76:            for ( int i = 0; i < attr.getLength(); i++ )
77:               System.out.println( "SAX Event - Attribute: "
78:               + attr.getName( i )
79:               + "=\"" + attr.getValue( i )
80:               + "\"" );
81:
82:  }
83:
84:    // SAX Event - Element End
85:    public void endElement( String el_name )
86:  throws SAXException
87:    {
88:            System.out.println("SAX Event - Element End: " +
               el_name );
89:  }
90:
91:    // SAX Event - Character Data and Whitespace characters
92:    public void characters( char buf [], int offset, int len )
93:  throws SAXException
94:    {
95:       if ( len > 0 ) {
96:          String stuff = new String( buf, offset, len );
97:
98:          System.out.println( "SAX Event - Characters: " + stuff );
99:       }
100:   }
101:
102:   // SAX Event - Processing Instructions
103:   public void processingInstruction( String part1,String part2 )
104:    throws SAXException
105:   {
106:      System.out.println( "SAX Event - Processing Instruction: "
107:               + part1 + " " + part2 );
108:   }
109: }
```

Компиляция и выполнение EList

Для компиляции программы `EList` вам необходимо определить переменную окружения `CLASSPATH` или добавить параметр `CLASSPATH` к инструкции запуска компилятора из командной строки, чтобы удостовериться в том, что вы включили все необходимые пакеты JASP. В документации по JASP вы найдете подробные сведения, как это сделать. Например, определить переменную окружения `CLASSPATH` в Windows можно следующим образом:

```
Set CLASSPATH=[dir]\jaxp.jar; [dir]\parser.jar;
[dir]\crimson.jar; [dir]\xalan.jar;
```

В данном случае вам необходимо заменить `[dir]` на путь к дистрибутиву JAXP, который вы установили раньше в настоящем уроке. После определения переменной окружения вы можете скомпилировать программу, воспользовавшись следующей командой:

```
javac EList.java
```

Для того чтобы это сработало, переменная окружена должна быть задана должным образом, пакет Java 2 Standard Edition (Java 2SE) — правильно установлен, а папка `bin` должна указываться в качестве пути к выполняемым файлам. Выполните все необходимые действия по настройке переменных окружения, которые подробно описаны в документации по JASP и Java 2SE.

Поскольку Java выполняется на любых платформах, вы можете выполнить этот код, например, на компьютере, работающем под управлением UNIX. В этом случае инструкция компиляции будет выглядеть следующим образом.

```
javac -classpath ${XML_HOME}\jaxp.jar; ${XML_HOME}\parser.jar;
${XML_HOME}\crimson.jar; ${XML_HOME}\xalan.jar; EList.java
```

В данном случае `{XML_HOME}` — это каталог UNIX, в котором содержатся необходимые файлы из дистрибутива JAXP.

После компиляции вы можете выполнить программу с помощью следующей команды:

```
Java EList [file_name.xml]
```

Если вы успешно откомпилировали приложение, выполните его вместе с файлом `message01.xml` из листинга 13.1. Результат должен выглядеть следующим образом:

```
Name of Document: file:C:/SAX/message01.xml
SAX Event - Start of Document
SAX Event - Element Start: note
SAX Event - Characters:
SAX Event - Characters:
SAX Event - Element Start: message
SAX Event - Attribute: from="Kathy Shepherd"
SAX Event - Characters:

SAX Event - Characters:   Remember to buy milk on the way home from work
SAX Event - Characters:

SAX Event - Characters:
SAX Event - Element End: message
SAX Event - Characters:

SAX Event - Element End: note
SAX Event - End of Document
```

SAX и пробелы

В предыдущем примере вы обработали документ XML, который выглядит очень простым. Этот документ содержал всего два элемента — с атрибутом, некоторыми текстовыми данными, комментарием и объявлением XML. Экземпляр документа XML выглядел следующим образом:

```
<?xml version = "1.0"?>
<!-- листинг 13.1 - message01.xml -->
<note>
    <message from="Kathy Shepherd">
       Remember to buy milk on the way home from work
    </message>
</note>
```

Помимо других событий, синтаксический анализатор SAX сообщает о любых событиях, атрибутах и текстовых данных. Однако полученный результат обработки документа содержит больше отчетов об символах, чем есть строк в экземпляре документа. Экземпляр документа содержит только одну "очевидную" строку: `Remember to buy milk on the way home from work`. У вас может возникнуть вопрос, а почему в результатах, представленных вашей Java-программой `Elist`, содержится столько сведений о текстовых данных? Это связано с тем, что SAX учитывает знаки пробелов, которые считает важными, вместо чтобы просто опускать их. Без схемы DTD синтаксический анализатор просто не в состоянии определить, что элемент, например, содержит смешанные данные. Поскольку наш документ не ограничен с помощью схемы DTD, SAX учитывает все знаки пробелов; а значит, каждый знак пробела вызывает соответствующее событие, о чем синтаксический анализатор сообщает пользователю.

Проверьте это, удалив пробелы между элементами в документе `message01.xml`, после чего проверьте полученный документ с помощью программы `EList`. Сохраните измененный документ XML под именем `message01.xml`, после чего он должен выглядеть так, как приведено в листинге 13.3.

Листинг 13.3. Пример удаления пробелов из простого документа XML `message02.xml`

```
<?xml version = "1.0"?><!-- листинг 13.3 - message02.xml --
><note><message from="Kathy Shepherd">Remember to buy milk on the way
home from work</message></note>
```

После проверки документа вы получите следующий результат:

```
Name of Document: file:C:/SAX/message02.xml
SAX Event - Start of Document
SAX Event - Element Start: note
SAX Event - Element Start: message
SAX Event - Attribute: from="Kathy Shepherd"
SAX Event - Characters:   Remember to buy milk on the way home from work
SAX Event - Element End: message
SAX Event - Element End: note
SAX Event - End of Document
```

Обратите внимание, что количество событий, соответствующих текстовым данным, значительно уменьшилось.

Обработка ошибок, возникших при работе обработчика SAX

В следующем упражнении вы преднамеренно внесете ошибку в документ `message01.xml`. В частности, измените регистр одного из знаков в закрывающем дескрипторе элемента `note`. Результат будет еще интереснее, если вы внесете изменения в конце экземпляра документа. Вы должны помнить, что синтаксический анализатор SAX обрабатывает отдельные события. Таким образом, можно ожидать, что документ будет обрабатываться должным образом вплоть до обнаружения ошибки. Если вы внесете ошибку в закрывающем дескрипторе последнего элемента, сообщение об ошибке появится в самом конце. В листинге 13.4 приведен документ, в котором в закрывающем дескрипторе элемента `note` содержится ошибка.

Листинг 13.4. Плохо отформатированный экземпляр XML для поверки с помощью синтаксического анализатора SAX `message03.xml`

```
1: <?xml version = "1.0"?>
2:  <!-- листинг 13.4 - message03.xml -->
3:  <note>
4:       <message from="Kathy Shepherd">
5:          Remember to buy milk on the way home from work
6:       </message>
7:  </Note>
```

После проверки этого документа с помощью программы `EList` вы должны получить приблизительно следующий результат:

```
Name of Document: file:C:/SAX/message01.xml
SAX Event - Start of Document
SAX Event - Element Start: note
SAX Event - Characters:
SAX Event - Characters:
SAX Event - Element Start: message
SAX Event - Attribute: from="Kathy Shepherd"
SAX Event - Characters:

SAX Event - Characters:   Remember to buy milk on the way home from work
SAX Event - Characters:

SAX Event - Characters:
SAX Event - Element End: message
SAX Event - Characters:

Parser Error: Expected "</note>" to terminate element starting on line 3.
```

Поскольку обработчик работает с отдельными событиями, документ будет обрабатываться удачно до тех пор, пока не возникнет ошибка; в этом случае, в соответствии с вашими инструкциями, выполнение программы прерывается, после чего генерируется отчет об ошибке. В нашем случае ожидался закрывающий дескриптор `</note>`; вместо него в документе содержится закрывающий дескриптор `</Note>` (строка 7 в листинге 13.4).

Проверка действительности документов с помощью SAX

Синтаксический анализатор JAXP можно использовать для проверки экземпляра документа XML на соответствие всем ограничениям, указанным в схеме. Присвоив свойству `setValidating` объекта `saxFactory` значение `true`, вы отдадите соответствующие указания синтаксическому анализатору JAXP:

```
saxFactory. setValidating(true);
```

Указав синтаксическому анализатору JAXP на необходимость проверки документа, вы можете выявлять ошибки, характерные именно для подобных ситуаций. Например, вы можете отлавливать предупреждения и небольшие проблемы с помощью методов `minor` и `warn` объекта `SAXParseException`. Код для создания методов `minor` и `warn` приведен ниже:

```
public void error(SAXParseException minor)
throws SAXParseException
{
    throw minor;
}

public void warning(SAXParseException warn)
throws SAXParseException
{
    System.err.println( "Warning: " + warn.getMessage() );
}
```

Более подробные сведения о типах ошибок, о которых сообщают синтаксические анализаторы SAX, вы найдете по адресу `http://java.sun.com/xml/download.html`. В конце настоящего урока упражнение использует приведенные выше методы для проверки экземпляра документа XML на соответствие всем ограничениям, указанным в схеме.

Резюме

В настоящем уроке вы познакомились с SAX (Simple API for XML), прикладным интерфейсом, который позволяет разработчикам воспользоваться всеми преимуществами событийной обработки документов. В отличие от спецификации DOM, SAX не требует загрузки всего экземпляра документа XML в память. SAX сообщает об отдельных событиях, возникших при обработке документа. Если ваш документ достаточно велик, использование SAX позволяет значительно сократить использование памяти по сравнению с использованием DOM. Это особенно полезно в том случае, когда в большом документе вас интересуют всего несколько элементов.

Вопросы и ответы

В каких случаях программист должен выбирать SAX, а в каких — DOM?

DOM замечательно подходит для выполнения различных задач, связанных с обработкой документов. Вы можете использовать DOM в случае произвольного доступа к документу для реализации сложных схем поиска. Если вам необходимо выполнить сложную фильтрацию XPath, а также внести изменения в документы XML, DOM обеспечит вас очень широкими возможностями.

SAX обеспечивает более высокую производительность в случае обработки больших документов. Когда вам необходимо прервать работу обработчика при обнаружении первой же ошибки, лучше всего использовать SAX. Если вам необходимо получить небольшие фрагменты информации из больших экземпляров документов XML, но вы не можете позволить такую нагрузку на систему, к которой приводит использование DOM, то SAX — ваш лучший выбор.

Каким образом программист может использовать SAX?

Вы создаете экземпляр синтаксического анализатора XML, инициализируя объект SAX. Вам потребуется написать программу, которая будет в состоянии отлавливать события, возникающие при обработке документа, и сообщать о них должным образом.

Упражнение

Проверьте свои знания, полученные в результате изучения настоящего урока. Ответы вы найдете в Приложении А "Ответы на вопросы".

Измените свое приложение `EList` таким образом, чтобы оно позволяло проверять действительность документов XML в соответствии с ограничениями, определенными в схемах DTD. Сохраните измененное приложение как `EList2.java`, после чего выполните его компиляцию. Используйте документ `CD.xml` и связанную с ним схему `CD.dtd` из урока дня 4-го.

Убедившись в том, что все работает должным образом, преднамеренно внесите ошибку проверки действительности, а не синтаксическую ошибку XML, добавив пустой элемент, не объявленный в схеме DTD. Посмотрите на результаты обработки документа с использованием SAX.

День 14-й

Использование каскадных таблиц стилей CSS

Разделение стилей и содержания — это фундаментальная концепция XML. Инструкции назначения стилей связываются с документами XML несколькими способами. Один из них предполагает связывание с документами XML таблиц каскадных стилей CSS, которые изначально разрабатывались для применения на HTML-страницах.

Сегодня рассмотрены следующие темы.

- Компоненты каскадных таблиц стилей и основные действия, необходимые для создания документа CSS.
- Несколько полезных свойств CSS и назначение значений этим свойствам.
- Ограничения, связанные с использованием каскадных таблиц стилей.

Стиль и содержание

Одна из принципиальных задач, которые ставились при разработке XML, — отделение содержания от внешнего вида документа. В уроках первых двух дней вы уже узнали, что подобное разделение стиля и содержания — одно из ключевых различий между XML и HTML. HTML смешивает содержание и стиль в одной разметке. Например, дескриптор HTML `<H1>` не только определяет данные как заголовок первого уровня, но и указывает броузеру на необходимость отображения их шрифтом большего размера. Точно также дескриптор HTML `<I>` указывает броузеру на необходимость отображения данных с использованием курсивного начертания. Как вы уже знаете, XML — это язык разметки, который описывает содержание, независимо от того, как оно будет представлено в окне броузера, на напечатанной странице или после использования любым другим агентом пользователя.

ПРИМЕЧАНИЕ

Агент пользователя — это любое программное обеспечение, которое преобразует символы документа в данные и размечает их в соответствии с определенными правилами. Различные агенты пользователя реализуются на разных устройствах, таких как компьютеры, сотовые телефоны с поддержкой Web, Web-броузеры и т.д.

Вы можете создавать собственные типы элементов в XML, поэтому агент пользователя не содержит встроенного набора инструкций, позволяющего определять, каким образом отображать данные этих элементов. Если вы решили отобразить данные XML в окне броузера, вам необходимо предоставить броузеру соответствующие инструкции отображения, чтобы он смог представить данные в своем окне. Вы можете сделать это, используя каскадные таблицы стилей CSS или расширяемый язык стилей XSL. В настоящем уроке вы познакомитесь с использованием таблиц каскадных стилей CSS, а завтра вы сможете сравнить этот подход с языком преобразований XSLT.

Каскадные таблицы стилей и HTML

HTML изначально не задумывался как язык для отображения данных в окнах броузеров; он предназначался в первую очередь для разметки данных для совместного использования по сети, причем на различных аппаратных платформах. Однако, по сравнению с первыми версиями HTML, в этот язык добавлены дескрипторы, имеющие определенное "типографское" значение: курсивное начертание (<I>), полужирное начертание (<B>), разрыв строки (<BR>) и т.д. Но даже несмотря на добавление подобных специальных дескрипторов, HTML никогда не обладал возможностями полного контроля за отображением данных. До появления каскадных таблиц стилей CSS Web-разработчикам приходилось прибегать к различным ухищрениям, чтобы решить подобные задачи. Например, для того чтобы добавить отступ в начале абзаца на Web-странице, многие разработчики просто вставляли "невидимое" изображение в формате GIF необходимой ширины и "нулевой" высоты. Изображение было совершенно незаметно для пользователей, просматривающих страницу, однако заставляло текст сместиться вправо, таким образом имитируя наличие отступа в первой строке абзаца.



Если вас интересуют подобные приемы, посетите Web-узел Stupid HTML Indent Tricks по адресу `http://www.jbarchuk.com/indent`.

Каскадные таблицы стилей созданы как метод представления HTML-страниц с определенным форматированием без добавления в дескрипторы какой-либо дополнительной информации. Каскадные таблицы стилей можно создать с помощью обычного текстового редактора, например Блокнот из состава Windows, после чего их следует сохранить в файле с расширением `.css`. Сейчас существует несколько версий таблиц CSS. Консорциум *W3C* определил набор стилей, известный как Cascading Stylesheets Level 1 (CSS1). Стандарт CSS2, также определенный консорциумом *W3C*, представляет собой новую версию, обладающую рядом дополнительных возможностей, надмножество CSS1. В данный момент в состоянии разработки стандарт CSS3, который может "похвастаться" минимальной поддержкой со стороны агентов пользователей.

Одно из фундаментальных средств CSS — каскад стилей. Стили могут быть указаны в рамках документа, элемента и внешнего документа. В действительности, для любого заданного экземпляра документа стили могут существовать во всех этих трех указанных местах. Подобная гибкость по отношению к расположению стиля обеспечивает модульность и гибкость проектирования документов. Существует целый ряд правил, определяющих взаимодействие стилей и то, как будут разрешаться конфликты стилей. Подробное описание правил, касающихся второго вопроса, вы найдете по адресу `http://www.w3.org/TR/REC-CSS1`. Например, файл `external.css` определяет применение к содержанию элемента курсивного начертания. В рамках документа глобальный стиль может определять применение к содержанию элемента полужирного начертания. В данном каскаде полужирное начертание "подавит" курсивное. Короче говоря, стиль, определенный на уровне элемен-

та, подавляет стиль, определенный на уровне всего документа, который, в свою очередь, подавляет любые стили, определенные во внешних файлах таблиц каскадных стилей. В настоящем уроке вы будет создавать внешние файлы таблиц каскадных стилей для применения стилей к экземплярам документов XML.

Если вы неоднократно создавали HTML-документы для использования в Web, вы уже знакомы с каскадными таблицами стилей CSS. Каскадные таблицы стилей CSS для XML ничем от них не отличаются; они позволяют задать стили, определяющие, как агент пользователя должен обрабатывать тот или иной элемент. Сейчас вы создадите и свяжете каскадные таблицы стилей с документами XML, а это — один из способов указать броузеру, как отображать тот или иной элемент документа XML.

Поддержка каскадных таблиц стилей броузерами

Таблицы стилей для отображения элементов используются на стороне клиента. Не удивительно, что в различных броузерах для разных платформ реализация поддержки каскадных таблиц стилей выполнена по-разному. В табл. 14.1 подытожены сведения о поддержке CSS1 определенными реализациями броузеров. Поддержка стандартов CSS2 и CSS3 выходит за пределы рассмотрения настоящей книги. Подробные сведения о поддержке стандартов CSS2 и CSS3 вы найдете по адресу `http://www.w3.org/Style/CSS/`.

Таблица 14.1. Поддержка броузерами стандарта CSS1

Платформа	*Броузер*	*Версия*
Linux	Amaya	2.0, 2.1, 2.2, 2.3, 2.4, 3.0, 4.0, 4.1, 4.2, 4.3.2
Linux	Netscape Navigator	4.7, Mozilla, 6
Linux	Opera	4a, 5.0
Macintosh	ICab Internet Browser	2,5 (частично)
Macintosh	Microsoft Internet Explorer	3.0, 4.0, 5.0
Macintosh	Netscape Navigator	3.0 (частично), 4.06 (частично), 4.5, 4.74, 6
Unix	Amaya	2.0, 2.1, 2.2, 2.3, 2.4, 3.0, 4.0, 4.1, 4.2, 4.3.2
Unix	Arena	3
Unix	Microsoft Internet Explorer	4.01, 5.0
Unix	Netscape Navigator	4.6
Windows	Amaya	2.0, 2.1, 2.2, 2.3, 2.4, 3.0, 4.0, 4.1, 4.2, 4.3.2
Windows	Microsoft Internet Explorer	3.0, 4.0, 5.0, 5.5, 6.0
Windows	Netscape Navigator	4.5, 4.7, 6
Windows	Opera	3.60, 4.02

Поскольку поддержка CSS достаточно распространена, целый ряд броузеров способен сразу открывать документы XML с соответствующей таблицей стилей. Такому прямому методу открытия не нужна HTML-страница для обработки данных. Однако за все приходится платить. Раз этот подход реализуется без поддержки со стороны броузера, вы не можете гарантировать, что пользователь увидит данные именно в том виде, в котором вы предполагали. Вы изучите более надежные способы представления данных XML в день 16-й, когда будете использовать язык преобразований XSLT (Extensible Stylesheet Language Transformations), а также в день 17-й, когда будете создавать области данных XML в HTML-странице.

CSS и XML

При использовании XML, как и в случае HTML, отделение стиля от содержания обеспечивает гибкость представления данных, упрощает их поддержание. Вам не потребуется вносить изменения в источник данных XML, если необходимо изменить их представление; вы можете просто откорректировать соответствующую каскадную таблицу стилей. Точно так же, после создания каскадной таблицы стилей CSS для представления данных должным способом, вы можете связать с ней другой документ, обеспечив единый подход к представлению различных данных на своем Web-узле.

Создав несколько таблиц стилей, вы можете разработать систему, которая будет доставлять одни и те же данные различным агентам пользователя. Например, представьте себе, что вам необходимо предоставить одни и те же данные XML всем посетителям Web-узла, независимо от того, какими броузерами они пользуются. Вы можете написать сценарий, который будет проверять броузер пользователя для определения каскадной таблицы стилей CSS, которая обеспечит наилучшее представление информации. Такой подход, называемый *browser sniffing*, используется достаточно широко. Вы можете определить агент пользователя несколькими способами, однако, наверное, проще всего — использовать свойство `navigator.appName` языка JavaScript. Простой сценарий, который использует этот подход и может размещаться на любой HTML-странице, выглядит следующим образом:

```
1: <script language="JavaScript">
2: <!--
3:   document.write("Определен броузер " + navigator.appName);
4: //-->
5: </script>
```

Этот простой сценарий JavaScript отображает на экране (метод `document.write`) строку, которая состоит из названия броузера (определенного с помощью свойства `navigator.appName`), а также поясняющей части. При этом, конечно же, предполагается, что броузер поддерживает JavaScript, что чаще всего полностью соответствует действительности. Если броузер не поддерживает JavaScript, он полностью игнорирует строку 3 благодаря использованию комментариев HTML (<!--...-->) в строках 2 и 4.

Многие другие методы, выполняемые на стороне клиента и на стороне сервера, позволяют программно определить природу агента пользователя, используемого для доступа к данным в Web. После определения подобных сведений используйте соответствующую таблицу стилей, что обеспечит представление данных именно в таком виде, как вы и хотели.

Концепция нескольких таблиц стилей, используемых для представления подобных данных в разных Web-броузерах, очень важна при разработке Web-страниц. Используя эти подходы, вы сможете выполнять следующее.

- Избирательное управление различными броузерами.
- Получение данных различными способами в зависимости от "контекстно-зависимых" средств управления. Например, из одного исходного документа XML вы можете выбрать одни данные для мультимедийной презентации, другие данные — для представления в виде таблиц и т.д.
- Контроль за потоком данных из документа XML в различные приложения.
- Форматирование данных для использования определенным устройством, например настройка представлений данных для беспроводных устройств с поддержкой Web.

Завтра в следующем уроке вы создадите простой сценарий, который разрешит пользователю выбирать таблицу стилей XSL для просмотра данных с использованием различных представлений. Обязательно обратите внимание на много общих черт CS7S и XSL.

Создание простой таблицы стилей

Каскадная таблица стилей — это текстовый файл, обычно сохраняемый с расширением `.css`. Он создается с помощью обычного текстового редактора или с помощью редактора со встроенной поддержкой каскадных таблиц стилей CSS. Файл содержит правила для синтаксического анализатора агента пользователя по отображению данных. Эти правила состоят из селекторов и объявлений, которые определяют, каким образом данные должны быть представлены пользователю. Селекторы обычно ссылаются на отдельные элементы XML, а объявления определяют, каким образом тот или иной выбранный элемент отображается. Объявления содержат пары свойство—значение. Каждое свойство назначает определенное значение, например `block`, `10pt`, `bold`. Синтаксис правила CSS приведен ниже:

Селектор {свойство:значение; свойство:значение;...}

Используя все тот же документ `message01.xml`, с которым вы работаете уже несколько дней, начните с создания экземпляра документа XML, к которому в дальнейшем примените стиль, используя файл CSS. Каскадные таблицы стилей позволяют применять стили к элементам, но не к атрибутам, поэтому создайте экземпляр XML, содержащий несколько различных элементов, каждый из которых может быть представлен по-разному. Создайте документ, который выглядит приблизительно так, как показано в листинге 14.1. Сохраните его под именем `message01.xml`. Вы будете применять к эту документу несколько таблиц стилей, изучая материал настоящего урока.

Листинг *14.1*. Документ XML, к которому будут применены стили, `message01.xml`

```
1: <?xml version="1.0"?>
2: <!-- листинг 14.1 - message01.xml -->
3:
4: <note>
5:     <msg>
6:       <id>m1</id>
7:       <from>Kathy Shepherd</from>
8:       <message>Remember to buy milk on the way home from
        work</message>
9:     </msg>
10:
11:      <msg>
12:       <id>m2</id>
13:       <from>Greg Shepherd</from>
14:       <message> I need some help with my homework</message>
15:      </msg>
16:
17:    <msg>
18:        <id>m3</id>
19:      <from>Kristen Shepherd</from>
20:      <message> Please play Scrabble with me tonight</message>
21:     </msg>
22:</note>
```

Этот документ XML содержит корневой элемент `note`, который, в свою очередь, содержит дочерние элементы `msg`. Каждый из этих элементов также содержит дочерние элементы `id`, `from` и `message`.

Определение стилей

Предположим, вам необходимо отобразить экземпляр документа XML следующим образом: содержание элемента `id` — курсивом, содержание элемента `from` — полужирным, содержание элемента `message` — синим цветом. Вы можете создать правила CSS для каждого из этих элементов с помощью объявлений, содержащих соответствующие свойство–значение, чтобы получить необходимый результат.

Вам необходимо знать имена свойств, а также их соответствующие значения для получения должного эффекта.

- `font-style:italic`. Отображение содержания элемента курсивом.
- `font-style:bold`. Отображение содержания элемента полужирным.
- `color:blue`. Отображение содержания синим цветом.

Более полный список свойств и их возможных значений приводится дальше в настоящем уроке. Вы должны запомнить синтаксические правила CSS. Все правила, необходимые для получения перечисленных выше эффектов, приведены в листинге 14.2. Сохраните соответствующий файл CSS под именем `style01.css`.

Листинг 14.2. Простая каскадная таблица стилей `style01.css`

```
1: /* листинг 14.2 style01.css */
2:
3:  id
4:       {font-style:italic}
5:
6:  from
7:       {font-weight:bold}
8:
9:  message
10:    {color:blue}
```



В строке 1 показан синтаксис комментария в файле CSS. Комментарии в файлах CSS указываются с помощью знаков `/*` и `*/`. Набор правил для элемента `id` (строки 3 и 4) объявляет, что содержание элемента будет отображаться курсивом (`{font-style:italic}`). Содержание элемента `from` (строка 6) будет отображаться полужирным (`{font-style:bold}`). Содержание элемента `message` будет отображаться синим цветом (`{color:blue}`).

Связывание с файлом CSS

Теперь, когда вы уже создали файл CSS, вам необходимо связать его с документом XML. Вы можете сделать это, просто добавив соответствующую инструкцию обработки, которая включает в себя зарезервированное ключевое слово `xml-stylesheet`, а также два обязательных атрибута. Синтаксис инструкции обработки приведен ниже:

```
<?xml-stylesheet type="text/css" href="myfile.css">
```

Атрибут `type` обязателен, поскольку идентифицирует тип таблицы стилей, которую вы решили связать с документом XML. В случае каскадной таблицы стилей значение этого атрибута равно `"text/css"`. Такая часть этого значения, как `type`, называется MIME-типом. Хотя рассмотрение природы MIME-типов и выходит за рамки настоящей книги, вам следует знать, что все созданные вами таблицы стилей будут текстовыми; в них не используются такие MIME-типы, как `application`,

multipart, message и т.д. Такая часть, как css, называется подтипом и в данном случае объявляет, что вы создаете каскадную таблицу стилей CSS, а не таблицу стилей XSL, значения в формате Rich Text Format, формате с разделителями-запятыми и т.д.

Вам также необходимо включить атрибут href с действительным URL, указывающим на соответствующий файл CSS, имя которого указывается в качестве значения. Значение атрибута href может быть полностью определенным адресом URL, указывающим на определенный объект в Internet, или же частью адреса URL, определяющей относительное расположение локального файла CSS. Синтаксис инструкции обработки для связывания файла CSS с документом XML приведен ниже:

```
<?xml-stylesheet type="text/css" href="style01.css"?>
```

Добавьте эту инструкцию в документ XML перед корневым элементом, после чего сохраните результат под именем message02.xml. Полученный результат должен выглядеть приблизительно так, как показано в листинге 14.3.

Листинг 14.3. Документ XML после обработки с использованием каскадной таблицы стилей message02.xml

```
1: <?xml version="1.0"?>
2:  <!-- листинг 14.3 - message02.xml -->
3:
4:  <?xml-stylesheet type="text/css" href="style01.css"?>
5:
6:  <note>
7:      <msg>
8:        <id>m1</id>
9:        <from>Kathy Shepherd</from>
10:        <message>Remember to buy milk on the way home from
           work</message>
11:      </msg>
12:
13:     <msg>
14:       <id>m2</id>
15:       <from>Greg Shepherd</from>
16:       <message> I need some help with my homework</message>
17:     </msg>
18:
19:     <msg>
20:       <id>m3</id>
21:       <from>Kristen Shepherd</from>
22:       <message> Please play Scrabble with me tonight</message>
23:     </msg>
24: </note>
```



Инструкция обработки (<?xml-stylesheet type="text/css" href= "style01.css"?>) добавлена (строка 4) для связывания таблицы стилей типа ="text/css", сохраненной в файле style01.css в той же папке, что и файл message02.xml, на жестком диске вашего компьютера.

Если вы выполнили все мои указания, вы сможете просмотреть файл message02.xml с помощью броузера, поддерживающего стандарт CSS Level 1 (CSS1), как показано на рис. 14.1.

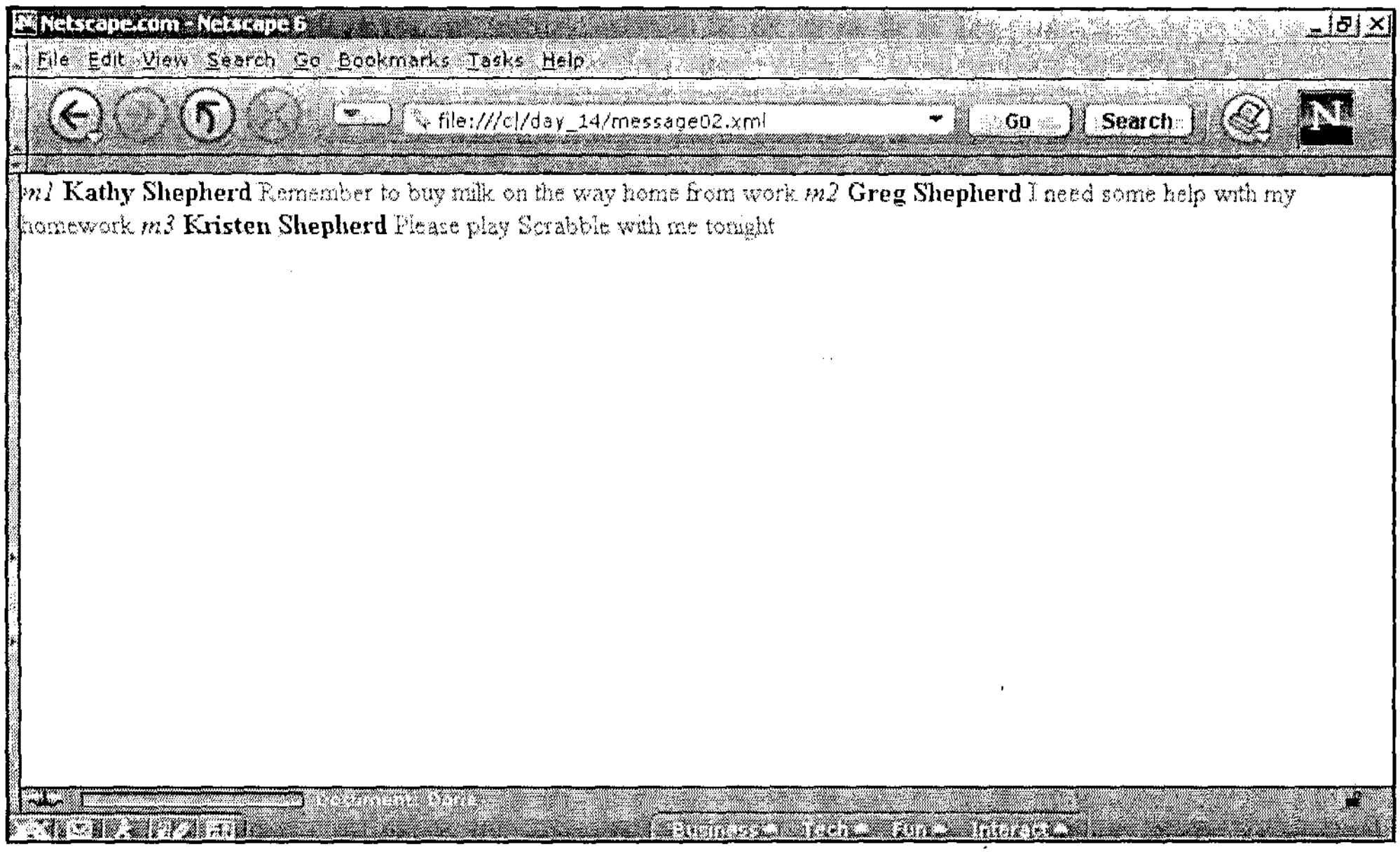


Рис. 14.1. Документ `message02.xml`, оформленный в соответствии с правилами из файла `style01.css` и отображенный в окне броузера Netscape 6

Свойства каскадных таблиц стилей CSS

Рассмотрение всех существующих свойств каскадных таблиц стилей CSS просто-напросто выходит за рамки настоящей книги; однако часто используемые свойства я представлю в виде таблиц. Полный список свойств каскадных таблиц стилей CSS вы найдете на многочисленных Web-узлах. Например, алфавитный список свойств вы найдете по адресу `http://www.bloorberry.com/indexdot/css/propindex/all.htm`. Полный список свойств с указанием их совместимости с различными броузерами вы найдете по адресу `http://www.webreview.com/style/css1/charts/mastergrid.shtml`. Лучший способ изучить все эти стили — экспериментировать с различными свойствами и их значениями. Поэкспериментируйте с документом `message02.xml`, изменяя содержания файла `style01.css` по мере изучения урока. В конце раздела сравните ваш документ CSS и результат его применения с приведенными примерами.

Свойства стилей текста

Свойства стилей текста определяют, как именно текст будет представлен на странице. Эти свойства влияют на выравнивание текста, высоту строку, расстояние между буквами и т.д. Например, предположим, вам необходимо, чтобы текст в элементе `message` в окне броузера был выровнен по левому краю и подчеркнут. Если вы повторно использует файл CSS, который создали раньше, свойство `color` уже определено для элемента `message`; таким образом, ваше новое правило CSS будет выглядеть следующим образом:

```
message
        {color:blue;
        text-align:left;
        text-decoration:underline}
```

При выражении нескольких свойств элемента с помощью правила CSS каждое объявление необходимо разделять точками с запятой. Весьма полезные свойства стилей текста приведены в табл. 14.2.

Таблица 14.2. Полезные свойства стилей текста CSS

Свойство	*Назначение*	*Допустимые значения*
`letter-spacing`	Контролирует расстояния между буквами в фрагменте текста	`normal` (задается по умолчанию); количество пикселей
`line-height`	Контролирует расстояния между буквами в фрагменте текста	`normal` (задается по умолчанию); количество пикселей

`text-align`	Контролирует выравнивание фрагмента текста	Выбирает броузер (задается по умолчанию)
		`left`
		`right`
		`center`

`text-decoration`	Контролирует внешний вид текста	`none` (задается по умолчанию)
		`underline`
		`overline`
		`line-through`
		`blink`

`text-indent`	Контролирует отступ первой строки в фрагменте текста	`0` (задается по умолчанию); количество пикселей или процентное значение

`text-transform`	Изменяет регистр символов в фрагменте текста	`none` (задается по умолчанию)
		`uppercase`
		`lowercase`
		`capitalize`

`vertical-alignment`	Изменяет выравнивание фрагмента текста по вертикали	`baseline` (задается по умолчанию)
		`sub`
		`super`
		`top`
		`text-top`
		`middle`
		`bottom`
		`text-bottom`

`word-spacing`	Контролирует расстояние между словами. На момент написания книги это свойство еще не работало	`normal` (задается по умолчанию); количество пикселей

Свойства шрифтов

Свойства шрифтов, поддерживаемые каскадными таблицами стилей CSS, предоставляют вам широкие возможности контроля над внешним видом символов, используемых при отображении тех или иных элементов. При создании первого файла CSS

вы использовали свойства font-style и font-weight. Вы определили, что содержание элемента id отображается курсивным начертанием, а содержание элемента from — полужирным. Внесите в файл CSS изменения, необходимые для того, чтобы содержание элемента id отображалось курсивом и размером шрифта, составляющим 250% от обычного размера символов. Для того чтобы это сделать, вам необходимо определить следующее правило CSS для элемента id:

```
id
        {font-style:italic;
         font-size:250%}
```

Основные свойства шрифтов, поддерживаемые каскадными таблицами стилей CSS, перечислены в табл. 14.3.

Таблица 14.3. Основные свойства шрифтов CSS

Свойство	Назначение	Допустимые значения
font-family	Контролирует тип шрифта, отображаемого на экране	Выбирает броузер (задается по умолчанию). Семейство шрифта, например Arial
font-size	Контролирует размер шрифта	medium (задается по умолчанию); количество пикселей или процентное значение
font-style	Контролирует начертание шрифта	normal (задается по умолчанию) italic oblique
font-variant	Контролирует вид шрифта	normal (задается по умолчанию) small-caps
font-weight	Контролирует толщину шрифта	normal (задается по умолчанию) lighter bold bolder

Свойства цвета и фона

Эти свойства позволяют вам контролировать цвет шрифтов, фона и включение изображений. Когда вы создавали файл CSS раньше в настоящем уроке, вы использовали свойство color для отображения содержания элемента message синим цветом. Если вы еще не сделали этого, содержание элемента from отображается полужирным. Добавьте к элементу from правило CSS, отображающее его содержание цветом морской волны. Это выглядит следующим образом:

```
from
     {font-weight:bold;
     background-color:aqua}
```

Некоторые из свойств цвета и фона приведены в табл. 14.4.

Таблица 14.4. Некоторые свойства цвета и фона CSS

Свойство	*Назначение*	*Допустимые значения*
`background-attachment`	Контролирует прокрутку фона	`scroll` (задается по умолчанию) `fixed`
`background-color`	Контролирует цвет фона	`transparent` (задается по умолчанию); название цвета
`background-image`	Позволяет определять изображение фона	`none` (задается по умолчанию); адрес URL изображения
`background-position`	Контролирует положение фона на странице	`0% 0%` (задается по умолчанию); значение в пикселях, например `{20,20}`, или процентное отношение, например `{5%, 7%}`

`background-repeat`	Позволяет изменят узоры, используемые в качестве фона страницы	`repeat` (задается по умолчанию)
		`repeat-x`
		`repeat-y`
		`no-repeat`

`color`	Контролирует цвет текста	Решение принимает броузер (задается по умолчанию); название цвета

Свойства рамки

Можно, благодаря использованию каскадных таблиц стилей CSS, заключать в рамки содержание некоторых элементов. Создайте правило CSS, которое будет добавлять рамки к содержанию элемента `from`. Если вы продолжаете работать с двумя предыдущими примерами, правило CSS будет выглядеть следующим образом:

```
from
    {font-weight:bold;
    background-color:aqua;
    border-style:double}
```

Некоторые из свойств рамок приведены в табл. 14.5.

Таблица 14.5. Некоторые свойства рамок CSS

Свойство	*Назначение*	*Допустимые значения*
`border-bottom-width`	Контролирует ширину нижней стороны рамки	`medium` (задается по умолчанию) `thin` `thick`; количество пикселей
`border-color`	Контролирует цвет рамки	Стандартный цвет текста (задается по умолчанию); название цвета

Свойство	Назначение	Допустимые значения
border-left-width	Контролирует ширину левой стороны рамки	medium (задается по умолчанию) thin thick; количество пикселей
border-right-width	Контролирует ширину правой стороны рамки	medium (задается по умолчанию) thin thick; количество пикселей
border-style	Контролирует стиль рамки	none (задается по умолчанию) solid double
border-top-width	Контролирует ширину верхней стороны рамки	medium (задается по умолчанию) thin thick; количество пикселей
border-width	Контролирует ширину рамки	Не определено (задается по умолчанию) thin medium thick; количество пикселей
clear	Определяет наличие других разделов по бокам текущего	Нет (задается по умолчанию) left right
float	Определяет возможность "плавающего" состояния раздела	Нет (задается по умолчанию) left right
height	Определяет высоту раздела	auto (задается по умолчанию); количество пикселей или процентное значение
margin-bottom	Контролирует ширину нижнего поля	0 (задается по умолчанию); количество пикселей или процентное значение
margin-left	Контролирует ширину левого поля	0 (задается по умолчанию); количество пикселей или процентное значение
margin-right	Контролирует ширину правого поля	0 (задается по умолчанию); количество пикселей или процентное значение
margin-top	Контролирует ширину верхнего поля	(задается по умолчанию)
padding-bottom	Контролирует ширину заполняющих элементов с указанной стороны	0 (задается по умолчанию); количество пикселей или процентное значение

Окончание табл. 14.5

Свойство	Назначение	Допустимые значения
padding-left	Контролирует ширину заполняющих элементов с указанной стороны	0 (задается по умолчанию); количество пикселей или процентное значение
padding-right	Контролирует ширину заполняющих элементов с указанной стороны	0 (задается по умолчанию); количество пикселей или процентное значение
padding-top	Контролирует ширину заполняющих элементов с указанной стороны	0 (задается по умолчанию); количество пикселей или процентное значение
width	Контролирует ширину раздела	auto (задается по умолчанию); количество пикселей или процентное значение

Свойства отображения

Эти свойства предоставляют специальные инструкции броузеру по поводу характеристик отображения символов и расстояний между ними. Например, z-index — это специальное свойство, используется для контроля за порядком следования элементов при их наложении. Подобные задачи при использовании каскадных таблиц стилей CSS реализуются благодаря указанию относительных и абсолютных положений элементов. Подобные вопросы выходят за рамки сегодняшнего рассмотрения, однако вы сможете найти подробные сведения, касающиеся контроля за порядком следования элементов при их наложении, по адресу http://www.w3c.org/TR/REC-CSS2/.

При использовании стилей XML очень полезна пара свойство——пара display: block. Значение block указывает броузеру, что программа должна вставить разрыв строки до и перед текстом элемента. В своем файле CSS создайте правило, которое заставит каждый элемент msg отображаться в виде отдельного блока. Ваше новое правило может выглядеть следующим образом:

```
msg
    {display:block}
```

Доступные свойства отображения перечислены в табл. 14.6.

Таблица 14.6. Некоторые свойства отображения CSS

Свойство	Назначение	Допустимые значения
white-space	Контролирует использование пробелов в разделе	normal (задается по умолчанию) pre nowrap
display	Контролирует отображение раздела	block (задается по умолчанию) inline list-item нет
visibility	Контролирует видимость элемента	inherit (задается по умолчанию) visible hidden
z-index	Контролирует расположение элемента в определенном слое	auto (задается по умолчанию) число

Полученный стиль

Если вы выполнили все упражнения в настоящем уроке, ваш документ CSS должен выглядеть так, как показано в листинге 14.4.

Листинг 14.4. Полный пример таблицы стилей `style03.css`

```
1: /* листинг 14.4 style03.css */
2:
3:  id
4:       {font-style:italic;
5:        font-size:250%}
6:
7:  from
8:       {font-weight:bold;
9:        background-color:aqua;
10:       border-style:double}
11:
12: message
13:    {color:blue;
14:       text-align:left;
15:       text-decoration:underline}
16:
17: msg
18:    {display:block}
```

Когда вы свяжете эту таблицу стилей с документом XML, созданным раньше в настоящем уроке, результат отображения в окне броузера должен быть таким, как показано на рис. 14.2.

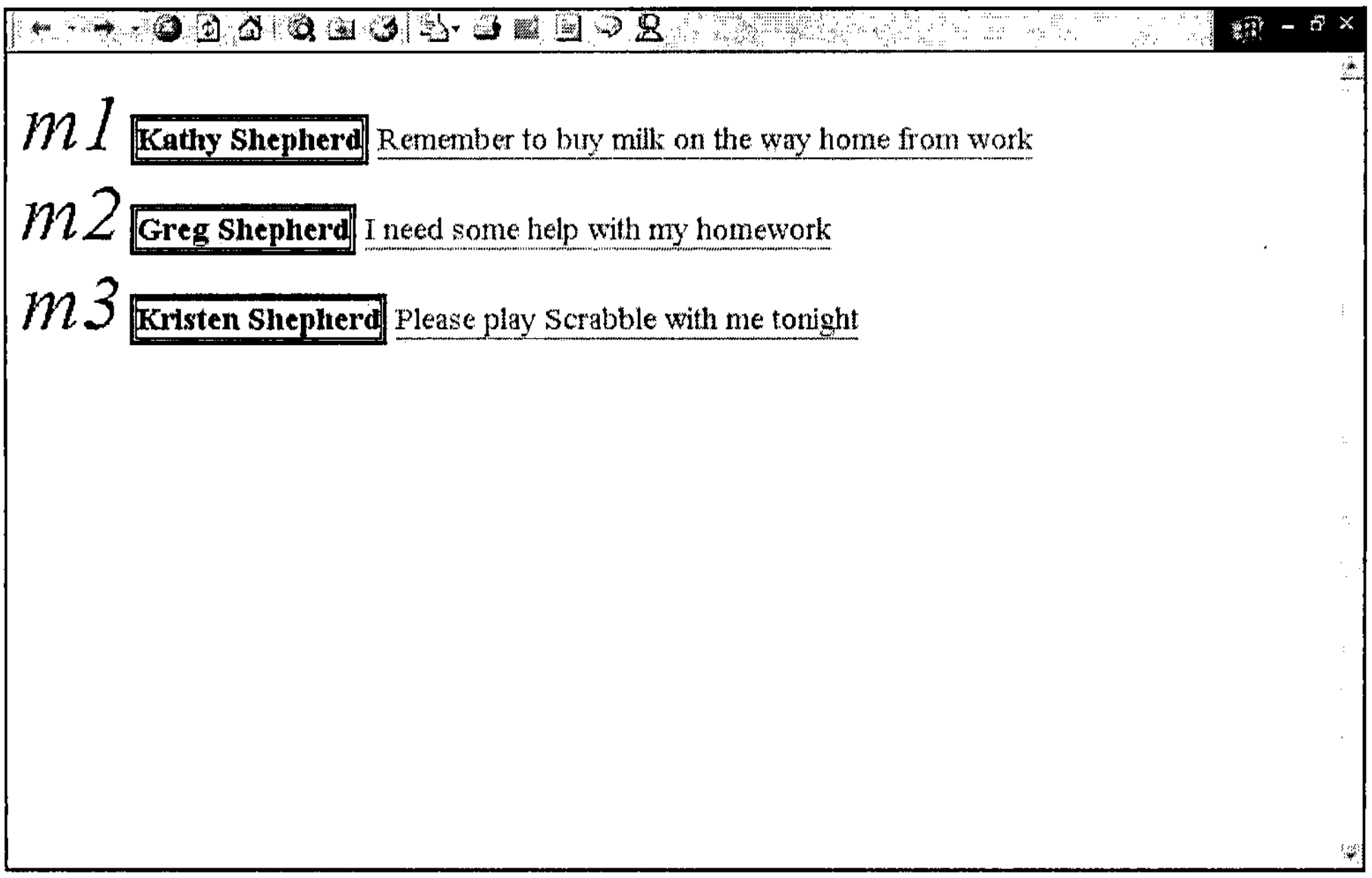


Рис. 14.2. Результат применения каскадной таблицы стилей CSS, полученный с помощью Microsoft Internet Explorer 5.5

Использование нескольких элементов в качестве селекторов

Если вы хотите применить одинаковый стиль к нескольким элементам, перечислите все элементы, разделяя их запятыми. Например, если вы хотите, чтобы содержание элементов `id` и `from` отображалось красным цветом, создайте правило CSS, выглядящее следующим образом:

```
id, from
    {color:red}
```

Ограничения CSS

Вы убедились в том, что каскадные таблицы стилей CSS контролируют много свойств отображения и достаточно просты. Однако для подхода, базирующегося на применении каскадных таблиц стилей CSS, характерен ряд ограничений. Например, Джон Бозак (Jon Bosak), спикер собрания Web-разработчиков, проведенного консорциумом *W3C* 11 апреля 1997 года, обратил внимание на следующие ограничения.

- Каскадные таблицы стилей CSS не могут "взять" элемент — например, заголовок главы — в одном месте и использовать снова его в другом — например, в верхнем колонтитуле страницы.
- В каскадных таблицах стилей CSS отсутствует концепция определенных отношений элементов. Например, нельзя написать каскадную таблицу стилей CSS, которая будет отображать содержание каждого абзаца полужирным.
- Каскадные таблицы стилей CSS не являются разновидностью языка программирования; не поддерживают структуры принятия решения и возможность расширения.
- Каскадные таблицы стилей CSS не позволяют выполнять вычисления и сохранять переменные. Это означает, что нельзя сохранять часто используемые параметры для упрощения выполнения таких действий, как обновление их значений.
- Каскадные таблицы стилей CSS не способны генерировать текстовые данные, например номера страниц.
- Каскадные таблицы стилей CSS используют готовую модель форматирования, поддерживаемую современными броузерами, которую нельзя расширить для использования более сложными приложениями разметки.
- Каскадные таблицы стилей CSS не поддерживают вертикальное расположение текста, характерное для ряда азиатских языков.

Источник: `http://www.webreview.com/1997/11_28/webauthors/11_28_97_5`.

Многие из этих ограничений преодолены в спецификации XSL, с которой вы познакомитесь в уроке дня 15-го.

Резюме

Сегодня вы создали несколько простых каскадных таблиц стилей CSS и использовали их для стилевой разметки документов XML при отображении в окне броузера. Этот подход поддерживается основной частью современных броузеров и представляет обработку на стороне клиента, простую и гибкую для реализации и понимания. Вы познакомились с правилами стилей CSS, которые выражаются в виде объявлений пар свойство—значение и определяют, каким образом агент пользователя отображает содержание того или иного элемента.

Вопросы и ответы

Какова роль проверки документов XML при использовании каскадных таблиц стилей CSS?

Каскадные таблицы стилей CSS предназначены для стилевого оформления документов XML, которые, как минимум, правильны. Если экземпляр документа содержит схему, он также должен быть и действительным. Это требование не каскадной таблицы стилей CSS, а обработчика XML или броузера, который обязательно попытается проверить документ на действительность, если тот содержит схему. Таким образом, проверка документов XML с каскадными таблицами стилей CSS явным образом никак не связана.

Каким образом использование каскадных таблиц стилей CSS позволяет сократить время, требуемое для управления Web-узлом?

Если вы имеете дело с Web-узлом, который содержит большое количество документов, значительная часть которых должна выглядеть подобным образом, используйте каскадную таблицу стилей CSS для придания документам абсолютно одинакового внешнего вида. Когда придет время изменять внешний вид узла, вам достаточно внести необходимые изменения в каскадные таблицы стилей CSS, даже не затрагивая при этом документы с данными. Все новые документы, добавляемые к Web-узлу, могут связываться с каскадной таблицы стилей CSS для сохранения общего стиля Web-узла.

Упражнение

Проверьте свои знания, полученные в результате изучения настоящего урока. Ответ вы найдете в Приложении А "Ответы на вопросы".

Используйте свойства, с которыми вы познакомились сегодня, чтобы применить стили к документу `CD.xml`, после чего он будет выглядеть так, как показано на рис. 14.3.

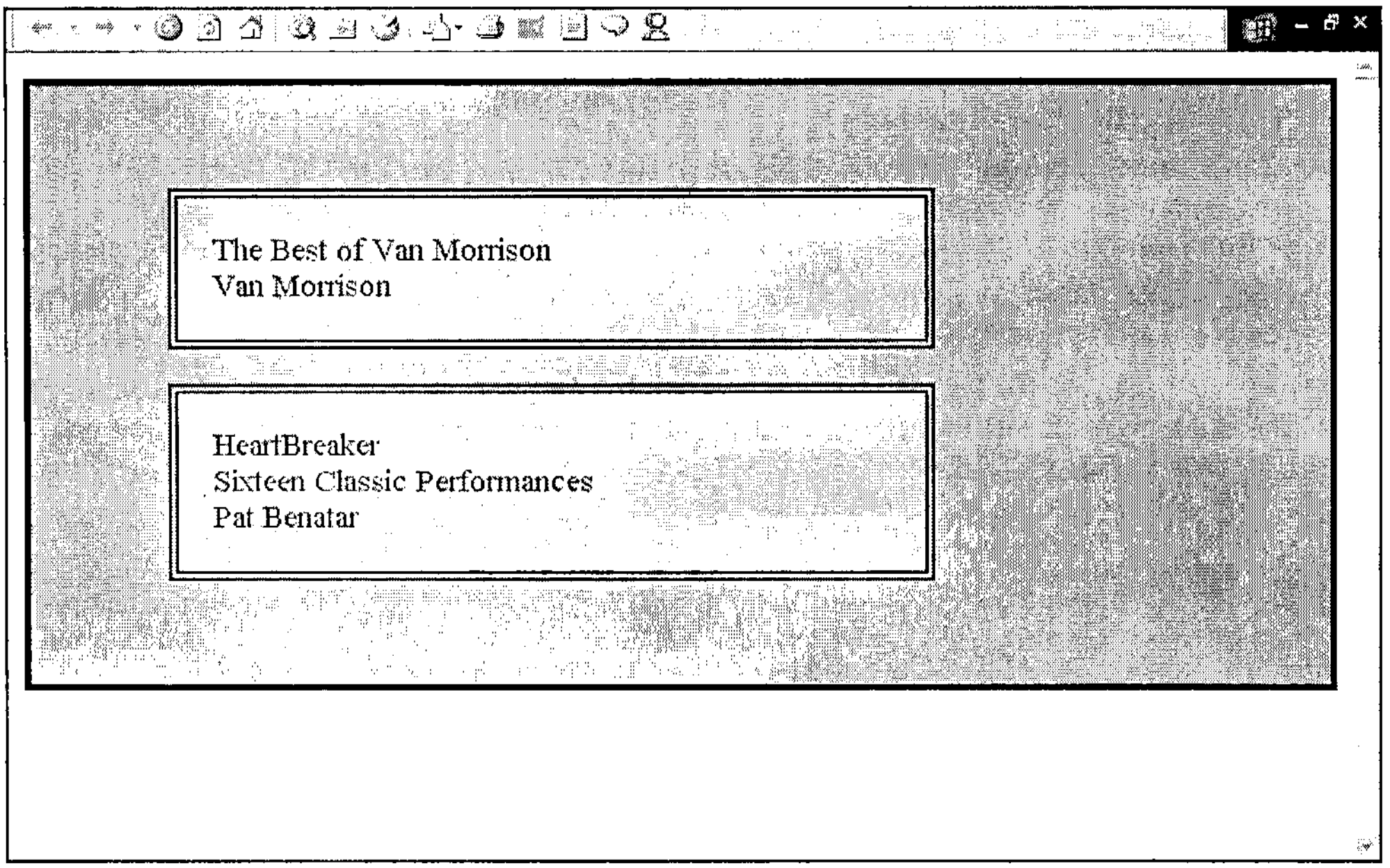
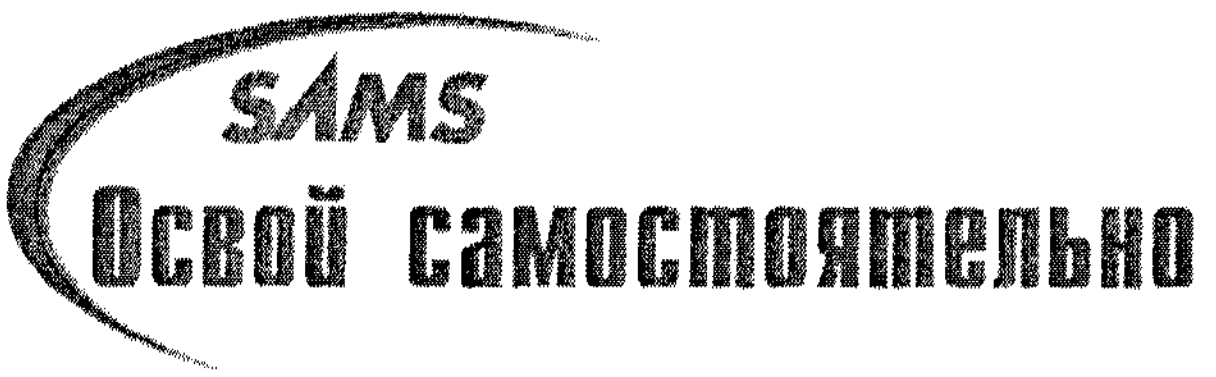


Рис. 14.3. Результат применения каскадной таблицы стилей CSS `cd.css` *к документу* `CD.xml`

Для получения подобных результатов вам необходимо располагать следующими сведениями.

- Графическое изображение с логотипом издательства *Sams Publishing* находится по адресу `http://samspublishing.com/images/samslogo.gif`, однако вам не следует его загружать.
- Весь текст отображен шрифтом размером 14 пунктов.
- Расстояние между внешней границей и краями внутренних рамок — 60 пикселей.
- Цвет фона под логотипом издательства *Sams Publishing* — HEX 999966.
- Цвет внутренних рамок — HEX 99cc99.
- Внутренние рамки занимают только три четверти доступного пространства по горизонтали.

Неделя III



Заставьте XML работать

15. Расширяемый язык таблиц стилей и его объекты форматирования
16. Язык преобразований XSLT
17. Связывание данных XML с элементами HTML
18. Технологии XBase и XInclude
19. Интеграция XML с корпоративными бизнес-моделями
20. Электронная коммерция и XML
21. Разработка Web-приложения с помощью XML

День 15-й

Расширяемый язык таблиц стилей и его объекты форматирования

Семейство технологий XML включает в себя расширяемый узел стилей XSL (Extensible Stylesheet Language), который предлагает широкие возможности форматирования, реализованные в виде словаря XML. XSL — это язык преобразований и специализированный диалект для описания объектов форматирования.

Сегодня рассматриваются следующие темы.

- Отношения между XSL, XSL-FO и XSLT.
- Операционный процесс в XSL-FO.
- Использование интерпретатора Apache.
- Создание PDF-файлов на основе документов XSL-FO.

Добавление форматирования к структуре

Web обеспечивает визуальное отображение информации, а это благодаря использованию мультимедиа предоставляет целый ряд возможностей представления документов XML. Концептуальное разделение *стиля* и *структуры* в XML открывает дополнительные возможности по сравнению с другими языками разметки для семантического определения элементов. Вчера, в предыдущем уроке, вы уже познакомились с одной из подобных возможностей, когда создавали таблицы каскадных стилей CSS для форматирования вывода документов XML с помощью соответствующих броузеров. Язык XSL предоставляет еще один вариант стилевого оформления данных в документах XML для отображения на экране, на печатных носителях, а также при синтезе речи.

Как уже говорилось в уроке дня 2-го, вы можете представлять себе элементы XML как *существительные*, а атрибуты — как *прилагательные*. Продолжая эту аналогию, вы можете представить себе инструкции XML как *глаголы* разметки XML. На самом деле, XML способен не только выражать семантику форматирования с помощью стилей. XSL способен полностью преобразовать экземпляр документа XSL к другой разметке.

XSL версии 1.0 — это кандидат в рекомендации *W3C*, описывающей язык для описания стилей.

- Язык преобразования документов XML, или язык преобразований XSLT (Extensible Stylesheet Language Transformations)

- Словарь для определения семантики форматирования, или объекты форматирования XSL-FO (Extensible Stylesheet Language Formatting Objects).

Можно сказать, что таблицы стилей XSL определяют представление отдельных классов документов XML. XSL использует словарь форматирования для описания того, каким образом экземпляр класса преобразуется или изменяется в новой структуре дерева. Однако язык преобразований XSLT можно использовать независимо от XSL-FO, и на первый взгляд это кажется противоречивым.

Соглашения об использовании имен XSL

Предлагая впервые XSL консорциуму *W3C* в августе 1997 года, авторы (компании *Microsoft*, *Inso* и *ArborText*, Эдинбургский университет и Джеймс Кларк) использовали аббревиатуру XSL для описания расширяемого языка стилей (Extensible *Style* Language). Однако при рассмотрении второго черновика рекомендации, опубликованного в декабре того же года, речь уже шла о расширяемом языке таблиц стилей (Extensible *Stylesheet* Language). Возможности спецификации постоянно расширялись, что привело к разделению двух процессов использования стилей в документах XML. Первый из них касается структурных преобразований: выбора, группировки и переупорядочения элементов. Второй процесс часто касается представления элементов с помощью конкретных агентов пользователя, отображающих документа экране, на бумаге, на других носителях и даже в виде речи. В апреле 1999 года разделение этих процессов было формализовано, когда язык преобразований XSLT описали в отдельном документе. Таким образом, XSL состоит из XSLT, предназначенного для определения преобразований, и XSL-FO, предназначенного для предоставления инструкций.

В официальной рекомендации консорциума *W3C* аббревиатура XSL-FO не используется, однако она широко известна в отрасли; и я буду использовать ее в настоящем уроке для ясности. Вам понадобится уделить особое внимание содержанию при чтении информации об использовании XSL в Internet, поскольку аббревиатуры XSL, XSL-FO, XSL:DO, XSLFO, XSL/FO и другие варианты используются тогда, когда речь идет об одном и том же.

В июле 1999 года стандарт XML Path Language (XPath), с которым вы познакомились в уроке дня 9-го, отделен от стандарта XSLT для описания общего синтаксиса и семантики, общих для XSL Transformations (XSLT) и XML Pointer Language (XPointer). Вспомните из урока дня 11-го, что XPointer объединяет идентификаторы URI с выражениями XPath при использования фрагментов удаленных документов XML для дальнейшей обработки.

Создание и использование объектов форматирования

Объекты форматирования XSL (XSL Formatting Objects) обычно получаются в результате преобразования документа XML, предназначенного для печати. Однако вы можете создать код XSL-FO без преобразования документа, используя только соответствующие синтаксис и семантику. Однако это не те действия, к которым вам придется часто прибегать на практике. Создание объектов форматирования вручную позволяет изучить синтаксис FO.

В настоящем уроке вы познакомитесь с XSL, самостоятельно создав и обработав объекты форматирования XSL-FO, а завтра, в следующем уроке, вы более подробно познакомитесь с языком преобразований XSLT. Я рассматриваю только самые важные и полезные объекты форматирования, предоставив ссылки на источники, из которых вы сможете получить сведения о других объектах форматирования.

Таким образом, объекты форматирования — это только один из аспектов многоэтапного процесса, который позволяет изменить древовидную структуру исходного докумен-

та XML. Полученное после преобразований дерево уже содержит объекты форматирования. Таким образом, язык XSLT для получения объектов форматирования выполняет преобразование древовидной структуры документа XML к структуре, содержащей объекты форматирования. XSLT можно представить как язык программирования, который выражается с помощью словаря XML и предназначен для преобразования древовидных структур документов. XSL-FO — это просто еще один словарь XML, в котором объекты представляют сведения, а также их свойства, на экране или странице.

Операционный XSL

Вы уже познакомились с несколькими процессами и словарями, используемыми для изменения документа XML при применении стилей к его содержанию. При преобразовании стилей для получения объектов форматирования используются следующие компоненты.

- Исходный документ XML, который с точки зрения языка преобразований XSLT представляется как иерархическое дерево узлов
- Таблица стилей XSL, также древовидная структура, заключает в себе семантику преобразований к объектам форматирования.
- Обработчик XSLT, который анализирует исходную древовидную структуру и древовидную структуру таблицы стилей в памяти для получения объектов форматирования.
- Древовидная структура XSL-FO, полученная после преобразования.
- Один или несколько интерпретаторов объектов форматирования, которые считывают экземпляры FO и представляют документы в соответствующем формате, таком как текст, PDF (Portable Document Format — переносимый формат документов), PCL (Printer Control Language — язык управления печатью) или MIF (Management Information File — файл управляющей информации), наиболее подходящем для агентов пользователя.



В будущем возможна поддержка объектов форматирования со стороны агентов пользователя, таких как Web-броузеры, сотовые телефоны, а также другие устройства клиентов. Это позволит использовать действительно *широкие* возможности форматирования в платформно-независимых решениях.

Предположим, у вас есть документ XML, который хранится в локальной сети intranet и содержит сведения обо всех правилах работы, принятых в вашей организации. Возможно, этот документ регулярно обновляется для отражения любых изменений, касающихся работы сотрудников, бухгалтерского учета и т.д. Вам необходимо периодически обрабатывать этот документ для создания руководства в формате PDF. Использование формата PDF позволяет распространять сведения среди клиентов, которые смогут просматривать их без использования исходного приложения. Кроме того, бо́льшая часть современных броузеров поддерживают печать документов в формате PDF, распространяемых через Web, используя специальные дополнительные модули. Таким образом, вы получаете возможность распространять данные, которые не зависят от возможностей конкретных версий броузеров и других приложений.

Преобразования XSL-FO предназначены для решения подобных задач. Экземпляр документа XML (`corp.xml`) содержит сведения о корпоративных политиках и другую информацию, с которыми связана таблица стилей XSL (`corp.xsl`), определяющая семантику объектов форматирования для размера текста заголовков, выделенных фраз, полей и т.д. Средство XSLT обрабатывает два этих документа XML (`corp.xml` и

corp.xsl) как объединенные исходные древовидные структуры, после чего выдает результат преобразований в виде документа corp.fo. После этого интерпретатор FO обрабатывает полученный результат corp.fo, который представляет собой экземпляр XML, для преобразования объектов форматирования к выражениям, сохраняемым в виде файла PDF (corp.pdf). Это позволит вам рассылать электронные копии файла PDF всем сотрудникам, независимо от того, на каких платформах выполняются их клиент-приложения. Другими словами, независимо от того, используют ваши сотрудники программное обеспечение для UNIX, Linux, Macintosh или Windows, документы PDF будут представлены для них совершенно одинаково. Каждый из сотрудников сможет запустит клиент-приложение, поддерживаемое используемой им операционной системой, которое считает файлы PDF и выведет их на печать совершенно одинаково. Соответствующий пример приведен на рис. 15.1.

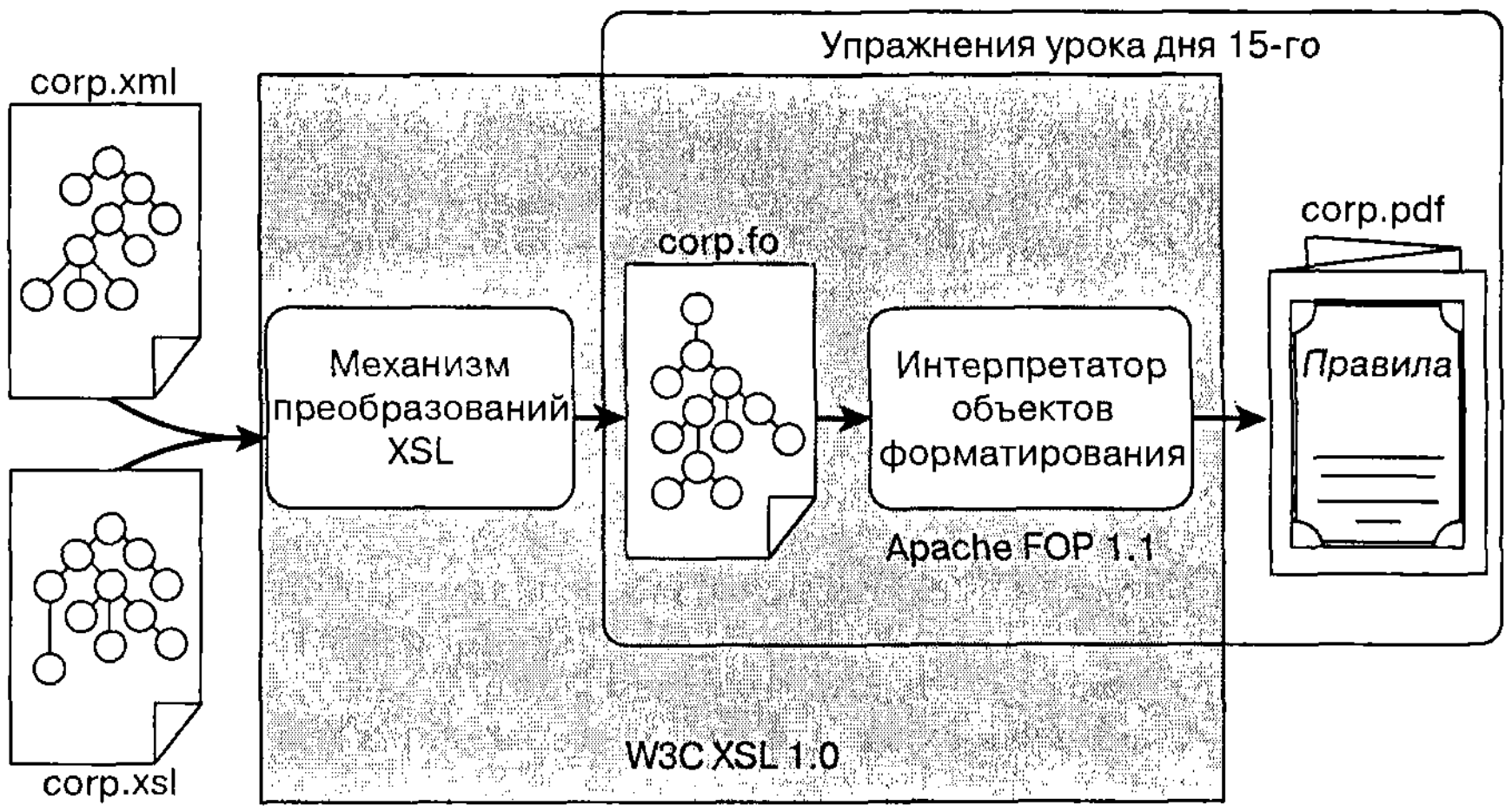


Рис. 15.1. Преобразования XSLT с использованием объектов форматирования FO

Полученная древовидная структура FO — это всего лишь документ XML, содержащий элементы, которые относятся к пространству имен FO, определяемому идентификатором URI http://www.w3.org/1999/XSL/Format. В настоящем уроке, вместо того чтобы выполнять преобразования для получения древовидной структуры FO, вы будете использовать текстовый редактор для создания документов FO, точно так же как вы поступаете при создании документов XML. На самом деле, вы как будто пропускаете этап преобразования. После этого вы обработаете его с помощью средства FOP (Formatting Object Processor — обработчик объектов форматирования), которое представляет собой обработчик печати, управляемый с помощью XSL-FO. Раздел, обозначенный на рис. 15.1 как "Упражнения урока дня 15-го", демонстрирует часть типичной операции XSL, с которой вы познакомитесь сегодня.

Обработчик объектов форматирования Apache FOP

FOP — это бесплатное Java-приложение, распространяемое через Internet организацией *Apache Software* (http://xml.apche.org/fop/). Вы можете выбрать среди нескольких подобных обработчиков, включая следующие.

- Renderx XEP, написанный на Java коммерческий обработчик XSL-DO (http://www.rendrerx.com). Доступна для загрузки пробная версия программы.

- Arbotext Epic E-Content Engine (E3), коммерческий набор приложений, который обеспечивает поддержку обработки объектов форматирования (`http://www.arbotext.com`).
- Подробный список обработчиков вы найдете на Web-узле консорциума *W3C* по адресу `http://www.w3.org/Style/XSL/`.



Обработчик FOP, который вы будете использовать в этом уроке, не поддерживает в полной мере спецификацию XSL 1.0 консорциума *W3C*. Поскольку на момент написания настоящей книги эта спецификация еще не получила статус рекомендации, еще возможны в ней некоторые изменения. Самые подробные сведения по этому вопросу вы найдете по адресу `http://www.w3.org/Style/XSL/`

FOP — это Java-приложение, которое считывает дерево объектов форматирования, после чего преобразует его в документ PDF. Его можно представлять и в нескольких других форматах, включая текстовый формат, MIF, PCL, AWT, а также вывод непосредственно на принтер. FOP принимает деревья FO в виде документов XML-FO, которые раньше создавались с помощью текстового редактора или средства XSLT. FOP поддерживает и непосредственную передачу данных от синтаксических анализаторов SAX и DOM без предварительного сохранения в виде файла FO. В настоящем уроке вы получите документы XML в виде экземпляров XSL-FO, после чего обработаете их с помощью приложения FOP.

Загрузите официальную версию FOP с Web-узла `http://xml.apache.org/fop/`. Если вы Java-разработчик и предпочитаете работать с самими свежими версиями программ, получите предварительную копию FOP. Однако в этом случае вы должны скомпилировать обработчик FOP самостоятельно, используя компилятор `javac` из состава Java 2 Standard Edition компании *Sun Microsystems*. Какой бы путь вы ни избрали, вам обязательно потребуется установить на своем компьютере определенные компоненты Java, прежде чем вы сможете использовать приложение FOP. Версии этих компонентов постоянно изменяются, поэтому всегда читайте сопровождающую их документацию, чтобы знать, что на вашем компьютере установлены все необходимые компоненты и что все они настроены должным образом.



Версия FOP, использованная для обработки примеров в сегодняшнем уроке, требует установки виртуальной машины Java из состава Java 2 Standard Edition компании *Sun Microsystems*, доступного по адресу `http://java.sun.com/j2se`.

Текущая версия FOP включает в себя несколько вспомогательных библиотек, к которым относятся следующие.

- Синтаксический анализатор XML Xerces-J, который поддерживает спецификации SAX и DOM.
- XSLT-обработчик Xalan.
- SVG-библиотека `w3c.jar`, соответствующая кандидату в рекомендации консорциума *W3C* SVG Candidate Recommendation (2 ноября 2000 года).
- Библиотека для работы с изображениями Jimi от компании *Sun Microsystems*

Как и в случае с другими Java-приложениями, вами следует побеспокоиться о названиях папок, в которых содержатся файлы пакета библиотек, о переменных окружения, которые определяют путь к библиотекам, а также о выполнении программ. Внимательно изучите указания по установке, предоставляемые вместе с версией FOP для вашей операционной системы. В следующих сегодняшних упражнениях вы создадите документы FO, после чего обработаете их для получения документов XML. Вер-

сия FOP, доступная на момент написания настоящей книги, вызывается вводом в командной строке операционной системы следующего выражения:

```
fop -fo файл_ввода.fo -pdf файл_вывода.pdf
```

Эта строка сообщает обработчику о том, что в качестве файла ввода вы предоставляете документ FO (*файл_ввода.fo*), а также ожидаете, что программа интерпретирует объекты форматирования и предоставит результаты в виде документа PDF (*файл_вывода.pdf*).

Однако версия FOP, которую загрузите вы, может требовать указания в командной строке другого выражения. Например, предварительная версия FOP — CVS, доступная на момент написания книги, использует следующий синтаксис:

```
java org.apache.fop.apps.CommandLine файл_ввода.fo файл_вывода.pdf
```



> Постоянно выпускаются новые версии FOP. Поэтому обязательно внимательно изучите документацию, сопровождающую загруженную вами версию FOP, чтобы узнать обо всех внесенных изменениях, касающихся инструкций, которые вводятся в командной строке, переменных окружения, а также основных требований.

Для просмотра полученных документов PDF вам потребуется специальная программа просмотра. Вы можете получить бесплатную версию Adobe Acrobat Reader версии 5.0, загрузив ее с Web-узла компании *Adobe Systems* по адресу `http://www.adobe.com/products/acrobat/readstep.html`.

Если вы решили использовать собственный просмотрщик FOP, программа Adobe Acrobat Reader вам не потребуется. Для запуска программы просмотра обратитесь к такому разделу документации по FOP, как *Running FOP*. Как в случае многих других подобных программ, синтаксис командной строки зависит от загруженной и установленной вами версии программы. В документации содержатся сведения, касающиеся именно той версии, которую вы загрузили.

Например, в случае Windows-версии FOP синтаксис командной строки следующий:

```
Fop -fo файл_ввода.fo -awt
```

В версии CVS используется следующий синтаксис:

```
java org.apache.fop.apps.AWTCommandLine файл_ввода.fo
```

Результатом ввода этой строки будет то, что документ FO (*файл_ввода.fo*) обрабатывается, а полученные результаты отображаются на экране, используя написанную на Java программу просмотра из состава FOP, а не сохраняются в формате PDF.

Объекты форматирования

Вчера, в предыдущем уроке, вы познакомились с различными возможностями форматирования, доступными при использовании каскадных таблиц стилей CSS. Спецификация XSL-FO также предлагает широкий спектр объектов, представляющих стили, доступные при использовании CSS.

Объекты способны контролировать разбивку на страницы и общую разметку документа, например объект `root`. Объект `root` не оказывает влияния на форматирование страницы, но определяет корневой элемент и обеспечивает контейнер для оставшихся объектов. Некоторые объекты форматирования содержатся в других объектах. Например, объект `fo:external-graphic` представляет ссылку на двоичный графический файл, который находится за пределами результирующего дерева FO. Еще один пример, объект `page-number` содержится в рамках других объектов разметки страницы. XSL-FO предоставляет богатую коллекцию объектов, используемых для форматирования таблиц. Каждый объект таблицы может содержать характеристики, указанные как

атрибуты свойств, контролирующие внешний вид объектов `fo:table-cell`, `fo:table-row`, `fo:table-column` или `fo:table`. К атрибутам свойств для элементов таблиц относятся `padding`, `border`, `background-color` и т.д.

В табл. 15.1 представлены объекты форматирования, определенные в спецификации XSL-FO консорциума *W3C* (`http://www.w3.org/Style/XSL/`), а также указано, какие именно из этих объектов поддерживаются версией 1.1 программы FOP. Специфические объекты, которые не реализованы в FOP версии 1.1, отмечены словом Недоступен в столбце FOP 1.1 таблицы. Список свойств, доступных для этих объектов, достаточно широк.

Если вы планируете весьма часто использовать спецификацию XSL-FO, вам следует хорошенько разобраться с документацией о ней. Описания свойств вы найдете в разделе 5.3 спецификации по адресу `http://www.w3.org/TR/xsl/slice5.html#compcorr`.

Таблица 15.1. Объекты форматирования W3C, поддерживаемые в FOP версии 1.1

Объект форматирования	Описание	FOP 1.1
`fo:basic-link`	Представляет начальный ресурс простой ссылки	+
`fo:bidi-override`	Используется при необходимости замены стандартного двунаправленного алгоритма на другие (или вложенные) сценарии в документах, созданных с помощью различных языков	Недоступен
`fo:block`	Широко используется для форматирования абзацев, заголовков, названий разделов, подписей к рисункам и таблиц и т.д.	+
`fo:block-container`	Используется для создания области ссылки на уровне блока	Недоступен
`fo:character`	Представляет символ, связанный с глифом	+
`fo:color-profile`	Используется для представления цветового профиля таблицы стилей	Недоступен
`fo:conditional-page-master-reference`	Используется для идентификации главной страницы, которая используется при выполнении поставленных условий	+
`fo:declarations`	Используется для группировки глобальных объявлений таблицы стилей	+
`fo:external-grahic`	Используется для указания внешних графических данных, находящихся за пределами результирующего дерева XML	+
`fo:float`	Используется для того, чтобы при нормальном размещении элементов определенные связанные сведения отображались в отдельной области в начале страницы (или одной из следующих страниц), либо в тех ситуациях, когда какая-нибудь область должна "плавать" с определенной стороны страницы	Недоступен
`fo:flow`	Представляет текстовые данные, которые занимают несколько страниц	+

Объект форматирования	Описание	FOP 1.1
fo:footnote	Используется для представления знака сноски и соответствующей сноски	+
fo:footnote-body	Используется для представления содержания сноски	+
fo:initial-property-set	Определяет свойства форматирования для первой строки объекта fo:block	Недоступен
fo:inline	Преимущественно используется для форматирования части текста вместе с фоном или связывания его с границей	+
fo:inline-container	Используется для представления внутренней ссылки	Недоступен
fo:instream-foreign-object	Используется для представления внутреннего графического изображения или другого "графического" объекта; данные являются потомками объекта fo:instream-foreign-object	+
fo:layout-master-set	Используется для представления главных объектов в документе	+
fo:leader	Используется для получения начальных элементов, состоящих или из правила, или строки символов, или повторяющихся последовательностей символов, которые используются для объединения двух объектов форматирования текста	+
fo:list-block	Используется для форматирования списков	+
fo:list-item	Содержит название и тело элемента списка	+
fo:list-item-body	Содержит тело элемента списка	+
fo:list-item-label	Содержит заголовок элемента списка; обычно используется для перечисления или идентификации тела элемента списка	+
fo:marker	Используется совместно с объектом fo:retrieve-marker для получения верхних и нижних колонтитулов	Недоступен
fo:multi-case	Используется для содержания (вместе с объектом fo:multi-switch) альтернативного дерева объектов форматирования, среди которых родительский объект fo:multi-switch может выбрать только один, скрыв все остальные	Недоступен
fo:multi-properties	Используется для переключения между двумя или несколькими свойствами, которые связаны с определенными данными	Недоступен

Объект форматирования	Описание	FOP 1.1
fo:multi-properties-set	Используется для указания альтернативного набора свойств форматирования, которые, в зависимости от состояния агента пользователя, применяются к содержанию	Недоступен
fo:multi-switch	Представляет несколько деревьев объектов форматирования (каждое из таких деревьев начинается с объекта fo:multi-case) и контролирует (благодаря объекту fo:multi-toggle) переключение между ними	Недоступен
fo:multi-toggle	Используется вместе с объектом fo:multi-case для переключения к другому дереву объектов форматирования	Недоступен
fo:page-number	Представляет номер текущей страницы	+
fo:page-number-citation	Используется для указания номера страницы, на которой в первый раз встречается указанный объект	+
fo:page-sequence	Определяет, как образовать последовательность страниц в рамках документа, например отдельную главу отчета. Содержание этих страниц образуется дочерними объектами объекта fo:page-sequence	+
fo:page-sequence-master	Определяет последовательности page-master, необходимых для получения последовательности страниц	+
fo:region-after	Определяет контрольное окно, расположенное “после” части документа fo:region-body	
fo:region-before	Определяет контрольное окно, расположенное “до” части документа fo:region-body	+
fo:region-body	Определяет пару “контрольное окно–ссылка”, которая находится в “центре” объекта fo:simple-page-master	+
fo:region-end	Определяет контрольное окно, расположенное в “конце” части документа fo:region-body	+
fo:region-start	Определяет контрольное окно, расположенное в “начале” части документа fo:region-body	+
fo:repeatable-page-master-alternatives	Определяет последовательность, образованную повторяющимися экземплярами набором альтернативных объектов page-master; количество повторений может быть связано	+

Объект форматирования	Описание	FOP 1.1
fo:repeatable-page-master-reference	Определяет последовательность, образованную повторяющимися экземплярами одного объекта page-master; количество повторений может быть связано	+
fo:retrieve-marker	Используется совместно с объектом fo:marker для создания верхних и нижних колонтитулов	Недоступен
fo:root	Состоит из объектов форматирования; это самый верхний узел в дереве XSL	+
fo:simple-page-master	Используется для получения страниц и определения их структуры; каждую страницу можно разделить на несколько областей	+
fo:single-page-master reference	Определяет последовательность, состоящую из одного экземпляра объекта page-master	+
fo:static-content	Содержит последовательность или дерево объектов форматирования, которые представляются в одной области страницы или связанных областях одной или нескольких страниц; часто используется при создании верхних и нижних колонтитулов	+
fo:table	Используется для форматирования данных в таблице	+
fo:table-and-caption	Используется для форматирования таблицы и ее заголовка	Недоступен
fo:table-body	Содержит основные данные таблицы	+
fo:table-caption	Содержит объекты форматирования, содержащие заголовок таблицы только при использовании объекта fo:table-and-caption	Недоступен
fo:table-cell	Используется для объединения данных, помещаемых в ячейку таблицы	+
fo:table-column	Определяет характеристики, применимые к ячейкам таблицы, относящимся к определенному столбцу	+
fo:table-footer	Содержит нижний колонтитул таблицы	+
fo:table-header	Содержит верхний колонтитул таблицы	+
fo:table-row	Используется для объединения ячеек таблицы в строки	+
fo:title	Связывает заголовок с определенным документом	+
fo:wrapper	Используется для определения наследуемых свойств для группы объектов форматирования	+

Основная конструкция объекта форматирования

В качестве первого упражнения настоящего урока предположим, что вам необходимо создать простой документ PDF, содержащий всего одну строку текста, например только заголовок страницы (`My Messages`) для системы управления сообщениями, с которой вы работаете уже несколько последних дней. Вы создадите файл FO, после чего выберете формат PDF как формат вывода при использовании интерпретатора FOP.

Вы добавите другие компоненты страницы по мере изучения настоящего урока, однако заголовок страницы — совсем неплохая точка для начала. Используя этот простой пример, вы сохраните полученный результат как экземпляр документа FO, после чего воспользуетесь программой FOP для получения файла PDF. Если на вашем компьютере установлен пакет Java 2 или Java Swing, вы можете получить результирующий файл, указав параметр `-awt` в командной строке интерпретатора FOP. Оба результата скоро предстанут перед вами.

Вам необходимо применить к заголовку страницы соответствующую разметку, чтобы получить файл PDF. Наши действия по разметке страницы создадут неплохую основу для дальнейших, более сложных действий. Вы должны помнить, что документ FO — это всего лишь экземпляр документа XML. Вы создадите его с помощью простого текстового редактора; однако обязательно сохраните его с расширением `.fo`, а не `.xml`. Вам также придется использовать префикс `fo:` для пространства имен объектов форматирования (`xmlns:fo=http://www.w3.org/1999/XSL/Format`). Этого же префикса требует и приложение FOP.

Пространство имен FO объявляется в корневом элементе документа. Корневой элемент документа XSL-FO всегда имеет следующий вид:

```
<fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">
```

Корневой элемент — это всего лишь контейнер. С ним не связана никакая стилевая разметка, хотя корневой элемент содержит все остальные объекты в документе.

Если вы быстро просмотрите несколько страниц настоящей книги, вы поймете, что при верстке употребляются такие понятия, как поля, размещение заголовков и остальных элементов страницы. В издательском деле все эти характеристики задаются в элементах, которые называются *главными элементами страницы*. Ваш любимый текстовый процессор наверняка поддерживает подобную концепцию. Например, на рис. 15.2. показано диалоговое окно Параметры страницы программы Microsoft Word 2002, в котором задаются как абсолютные значения полей, так и их расстояние от колонтитулов. Вы можете вводить значения параметров в соответствующие текстовые поля в соответствии с необходимым макетом страницы.

Эти настройки, которые влияют на весь документ, содержатся в элементе `fo:layout-master-set`. Другими словами, элемент `fo:layout-master-set` — это контейнер, содержащий все свойства главного набора. В элементе `fo:layout-master-set` содержатся элементы `fo:simple-page-master` и элементы-контейнеры, которые определяют разметку документа точно так же, как в текстовых процессорах. В XSL-FO может быть несколько элементов `fo:simple-page-master`, что позволяет при необходимости контролировать формат каждой отдельной страницы документа.

Элементы `fo:simple-page-master` требуют наличия атрибута `master-name`, которому вы присваиваете необходимое значение. Этот атрибут необходим для того, чтобы вы могли различить несколько элементов. Синтаксис элемента `fo:simple-page-master` приведен ниже:

```
<fo:simple-page-master master-name="имя_страницы">
```

Элемент `fo:simple-page-master` определяет пять областей на стандартной странице: верхний колонтитул, нижний колонтитул, тело документа, левое и правое поля. Область левого поля называется начальной, а правого — конечной. Каждой этой области соответствует определенный элемент XSL-FO, содержащий инструкции разметки.

Область верхнего заголовка представлена объектом форматирования `fo:region-before`, поскольку следует до основной части документа в логической последовательности ее разметки. Область левого поля представлена объектом форматирования `fo:region-start`, область правого поля — объектом форматирования `fo:region-end`, область нижнего колонтитула представлена объектом форматирования `fo:region-after`. Относительное расположение этих объектов приведено на рис. 15.3.

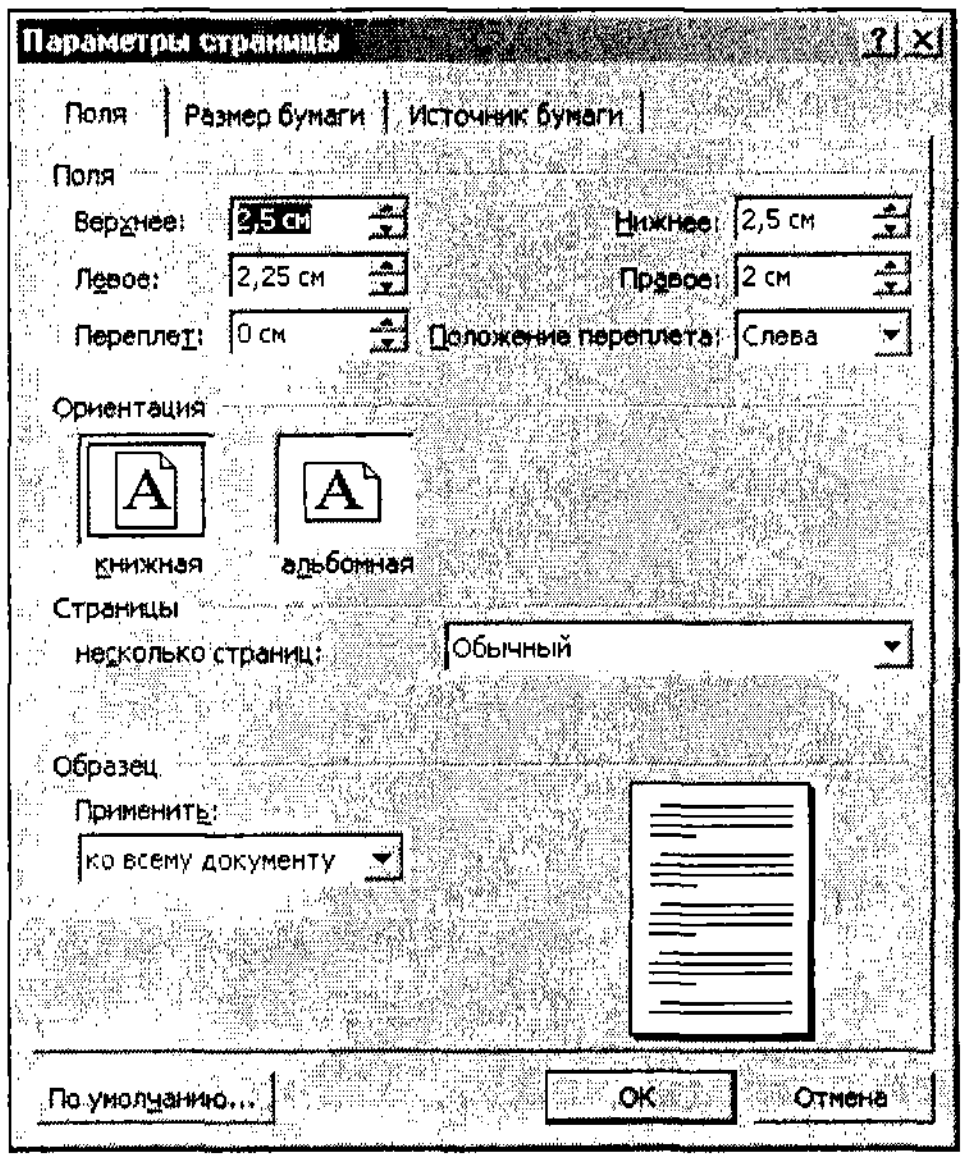


Рис. 15.2. Диалоговое окно Параметры страницы программы Microsoft Word 2002

Такое расположение характерно для таких языков, как, например, русский, в которых текст читается слева направо. В то же время, для арабского языка или иврита объект форматирования `fo:region-start` должен находится справа, а объект форматирования `fo:region-end` — слева.

Каждая из областей страницы может содержать свою стилевую разметку, указанную с помощью соответствующих объектов форматирования. Например, если вам необходимо, чтобы текст верхнего заголовка находился на расстоянии 5 см от левого поля, вы можете просто добавить соответствующий атрибут к объекту форматирования `fo:region-before`. В данном случае элемент разметки будет выглядеть следующим образом:

```
<fo:region-before margin-left="2in">
```

Предположим, вам необходимо разместить тело полученной страницы, создаваемой вами с помощью XSL-FO, в 3,75 см от левого поля и 2,5 см от верхнего точно так же, как это показано в диалоговом окне Параметры страницы программы Microsoft Word 2002 на рис. 15.2. Вы можете создать элеамент `fo:simple-page-master`, содержащий элемент `fo:region-body` с соответствующими значениями атрибутов `margin-left` и `margin-top`. Если предположить, что вы назвали первый главный элемент страницы `message-page`, полный элемент `fo:simple-page-master` выглядит следующим образом:

```
<fo:layout-master-set>
    <fo:simple-page-master master-name="message-page">
      <fo:region-body margin-left="1.5in" margin-top="1in"/>
    </fo:simple-page-master>
  </fo:layout-master-set>
```

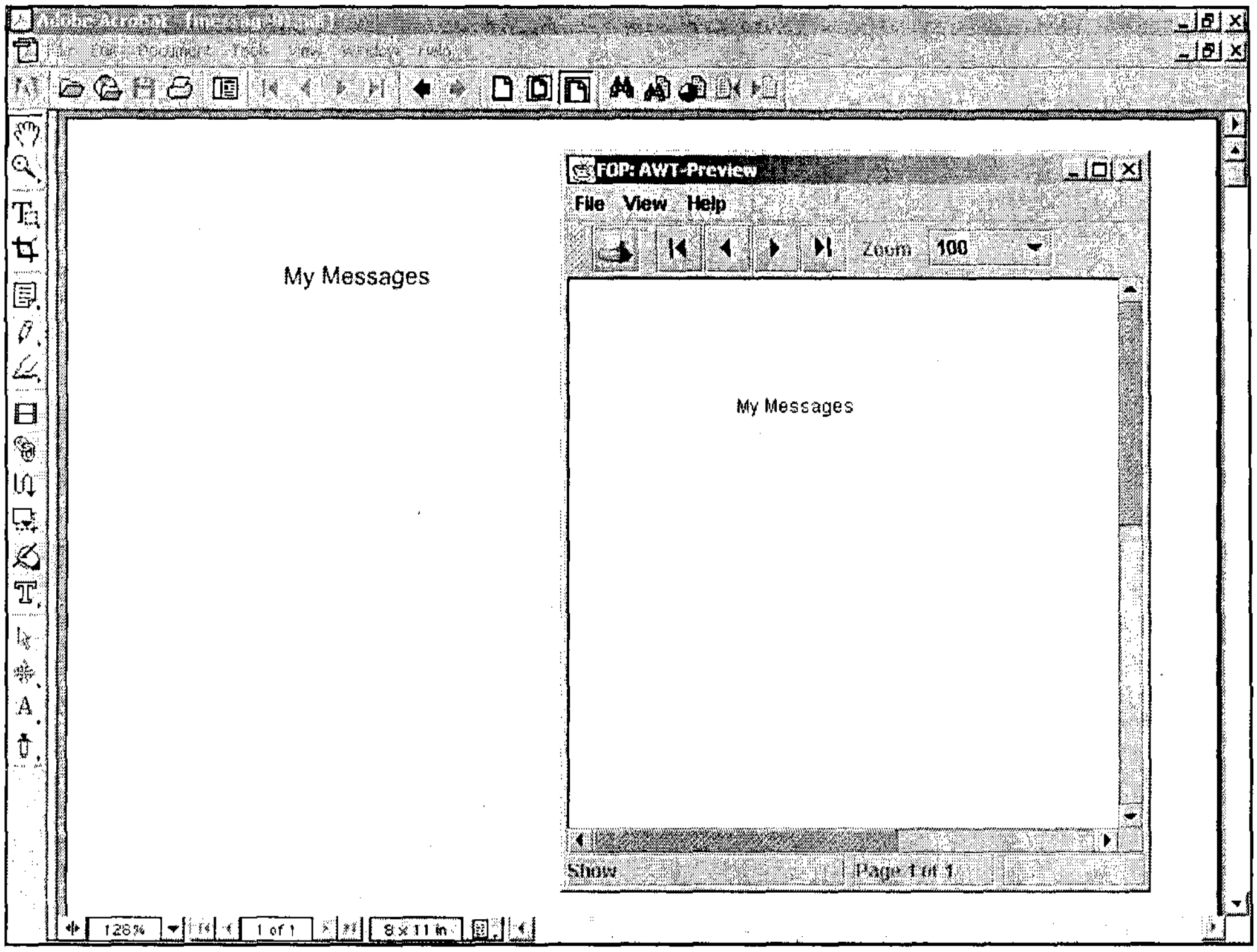


Рис. 15.3. Области форматирования страницы для языков, для которых характерно чтение слева направо

Вы можете видеть, что элемент `fo:layout-master-set` содержит один дочерний элемент `fo:simple-page-master` с атрибутом `master-name`, равным `message-page`. Элемент `fo:simple-page-master` содержит дочерний элемент `fo:region-body` с соответствующими атрибутами полей.

В XSL-FO страницы документа группируются в последовательности, каждая из которых начинается с новой страницы. Последовательности используются для явного определения того, как именно должны представляться страницы. Например, вы можете проинструктировать обработчик применять одну и ту же разметку до тех пор, пока не будет завершена обработка всего документа. Вы можете этого добиться, включив элемент `fo:repeatable-page-master-reference` с атрибутом `master-name`, значение которого совпадает с элементом `fo:simple-page-master`, который используется при обработке всего документа. Вам не нужно этого делать при выполнении первого упражнения, но этот прием очень полезен при работе с более сложными объектами форматирования. Сейчас вам следует запомнить, что последовательности задаются с помощью элементов `fo:page-sequence`, самых важных элементов в случае сложной разметки страницы. В первом документе XSL-FO вы будете использовать элемент `fo:page-sequence` для хранения текстовых данных. Он выглядит следующим образом:

```
<fo:page-sequence master-name="message-page">
другие элементы
  </fo:page-sequence>
```

К другим элементам, содержащимся в элементе `fo:page-sequence`, относится такой элемент-контейнер, как `fo:flow`, содержащий все текстовые данные в документе. Все эти данные форматируются в виде областей страниц в рамках указанной последовательности

страниц. В вашем документе вы будете использовать атрибут `flow-name` со значением `message-page` для связывания потока с определенной областью страницы. В частности, текст будет содержаться в *теле* страницы, поскольку именно эта область указана в качестве значения атрибута. Вы уже знаете, что речь идет именно об этой области страницы, поскольку элемент `fo:region-body` содержится в элементе `fo:simple-page-master`, значение атрибута `master-name` которого равно `my-page`. В настоящее время некоторые названия и связи между ними могут запутать вас, но вы уже познакомились со всеми элементами и атрибутами, необходимыми для создания первого документа XSL-FO. Объединив все эти элементы, вы получите документ, приведенный в листинге 15.1. Введите текст документа и сохраните его под именем `message01.fo`.

Листинг *15.1.* Первый документ XSL -FO `message01.fo`

```
1: <?xml version="1.0"?>
2: <!-- листинг 15.1 - message01.fo -->
3:
4: <fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">
5:
6:   <fo:layout-master-set>
7:     <fo:simple-page-master master-name="message-page">
8:       <fo:region-body margin-left="1.5in" margin-top="1in"/>
9:     </fo:simple-page-master>
10:  </fo:layout-master-set>
11:
12:  <fo:page-sequence master-name="message-page">
13:    <fo:flow flow-name="xsl-region-body">
14:      <fo:block>My Messages</fo:block>
15:    </fo:flow>
16:  </fo:page-sequence>
17:
18:</fo:root>
```



Строка 4 содержит открывающий дескриптор корневого элемента (`fo:root`) вместе с указанным пространством имен (`xmlns:fo="http://www.w3.org/1999/XSL/Format"`). Строки 6–10 содержат элемент `fo:layout-master-set`, который определяет структуру разметки всего документа. Элемент `fo:layout-master-set` может содержать любое количество элементов `fo:simple-page-master`, но в нашем случае он всего один и называется `message-page`. Тело документа находится на расстоянии 3,75 см от левого поля и 2,5 см от верхнего (строка 8). Строки 12–16 содержат элемент `fo:page-sequence` и его дочерние элементы, которые обеспечивают форматирование области, соответствующей элементу `fo:simple-page-master` со значением атрибута `master-name`. Элемент `fo:flow` (строки 13–15) содержит текст, который должен располагаться на странице в виде блока.

После того как вы ввели и сохранили документ `message01.fo`, обработайте его с помощью программы FOP. Для создания документа PDF введите в командной строке инструкцию, соответствующую установленной у вас версии FOP, например, следующую:

```
fop -fo диск:\путь\message01.fo -pdf диск:\путь\message01.pdf
```

Для просмотра результатов на экране введите в командной строке другую инструкцию, соответствующую установленной у вас версии FOP, например, следующую:

```
fop -fo диск:\путь\message01.fo -awt
```

Если вы внимательно изучите рис. 15.4, вы увидите, что на нем представлены оба варианта. На заднем плане показан документ PDF, полученный с помощью FOP и открытый с помощью Adobe Acrobat. На переднем плане показана результирующая страница, открытая с помощью встроенной программы просмотра FOP, написанной на Java.

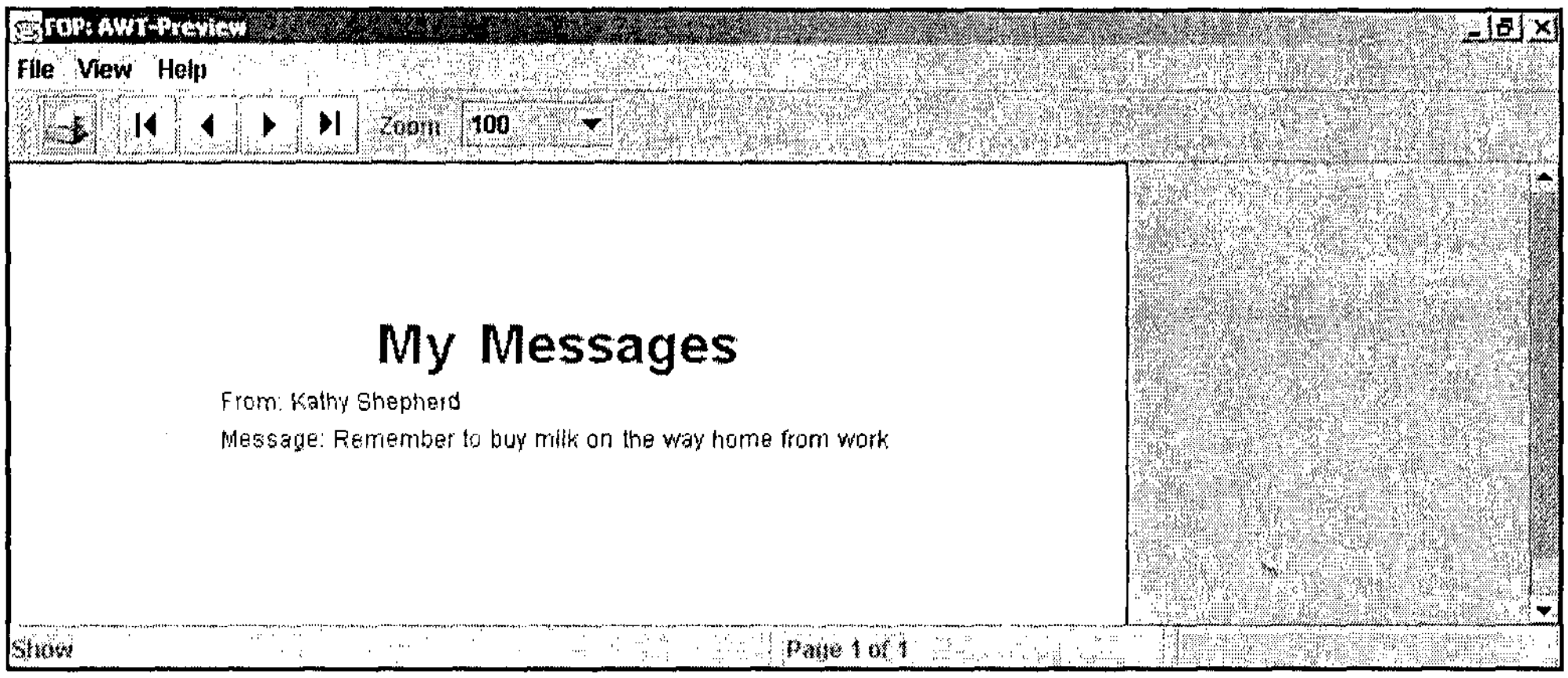


Рис. 15.4. Два результата обработки документа XSL-FO

Атрибуты шрифта

Остальные виды форматирования, с которыми вы познакомитесь, получаются в результате комбинирования атрибутов стилей и их значений с элементами, на которые они должны влиять. Вы уже познакомились с основной структурой документа XSL-FO. Эта структура оказывается практически неизменной, независимо от того, насколько сложным становится документ. Добавляя новые страницы, последовательности и потоки, вы можете создать все необходимые области любой страницы, а также различные стили. Таким образом, XSL-FO очень похож на каскадные таблицы стилей CSS. Многие из свойств, используемых в обоих подходах, тесно пересекаются. Консорциум *W3C* стремится к тому, чтобы стандарт XSL предлагал все возможности CSS, но при этом был лишен всех недостатков каскадных таблиц стилей, о которых мы говорили в предыдущем уроке.

В следующем упражнении вы добавите сообщение, которое будет включать имя отправителя и текст сообщения. Кроме того, вы добавите некоторые атрибуты текста и шрифтов для применения стилей к полученным результатам. Начните с перемещения заголовка страницы `My Messages` в центр, для чего воспользуйтесь атрибутом `text-align="center"` элемента `fo:block`:

```
<fo:block
                text-align="center">My Messages</fo:block>
```

Добавьте необходимые атрибуты для того, чтобы текст заголовка отображался шрифтом размером 28 пунктов, шрифтом без засечек, полужирным начертанием и синим цветом. Тогда элемент `fo:block` будет выглядеть следующим образом:

```
<fo:block
                text-align="center"
                  font-size="28pt"
                  font-family="sans-serif"
                  font-weight="bold"
                  color="blue">My Messages
              </fo:block>

    <fo:block
```

Кроме того, добавьте два элемента `fo:block` с тем же значением `xsl-region-body`, содержащих следующие текстовые данные:

```
From: Kathy Shepherd
Message: Remember to buy milk on the way home from work
```

Пусть оба элемента отображаются шрифтом без засечек размером 12 пунктов. Сохраните полученные результаты в виде файла `message02.fo`. Проверьте, добавили ли вы в него следующие фрагменты:

```
<fo:block
                font-size="12pt"
                font-family="sans-serif">From: Kathy Shepherd
          </fo:block>

     <fo:block
                font-size="12pt"
                font-family="sans-serif">Message:
                   Remember to buy milk on the way home from work
          </fo:block>
```

Внеся эти дополнения и сохранив документ, обработайте его с помощью программы FOP, после чего просмотрите полученный результат. Получили ли вы именно то, что хотели? Внимательно изучите заголовок. Он действительно отображается по середине страницы или все-таки смещен к ее правому краю? Обеспечить размещение заголовка строго посередине можно, если добавить в элемент `fo:region-body` атрибут `margin-right`, значение которого строго совпадает со значением атрибута `margin-left`. Вы можете задать это следующим образом:

```
<fo:region-body margin-left="1.5in"
                margin-right="1.5in" margin-top="1in"/>
```

Окончательный вариант кода приведен в листинге 15.2.

Листинг 15.2. Пример документа XSL-FO с использованием атрибутов шрифтов (message02.fo)

```
1: <?xml version="1.0"?>
2:  <!-- листинг 15.2 - message02.fo -->
3:
4:  <fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">
5:
6:    <fo:layout-master-set>
7:      <fo:simple-page-master master-name="message-page">
8:        <fo:region-body margin-left="1.5in"
9:                margin-right="1.5in" margin-top="1in"/>
10:     </fo:simple-page-master>
11:   </fo:layout-master-set>
12:
13:   <fo:page-sequence master-name="message-page">
14:     <fo:flow flow-name="xsl-region-body">
15:
16:       <fo:block
17:            text-align="center"
18:              font-size="28pt"
19:              font-family="sans-serif"
20:              font-weight="bold"
21:              color="blue">My Messages
22:         </fo:block>
23:
24:       <fo:block
```

```
25:                font-size="12pt"
26:                font-family="sans-serif">From: Kathy Shepherd
27:           </fo:block>
28:
29:        <fo:block
30:                font-size="12pt"
31:                font-family="sans-serif">Message:
32:                   Remember to buy milk on the way home from work
33:           </fo:block>
34:
35:           </fo:flow>
36:    </fo:page-sequence>
37:
38: </fo:root>
```

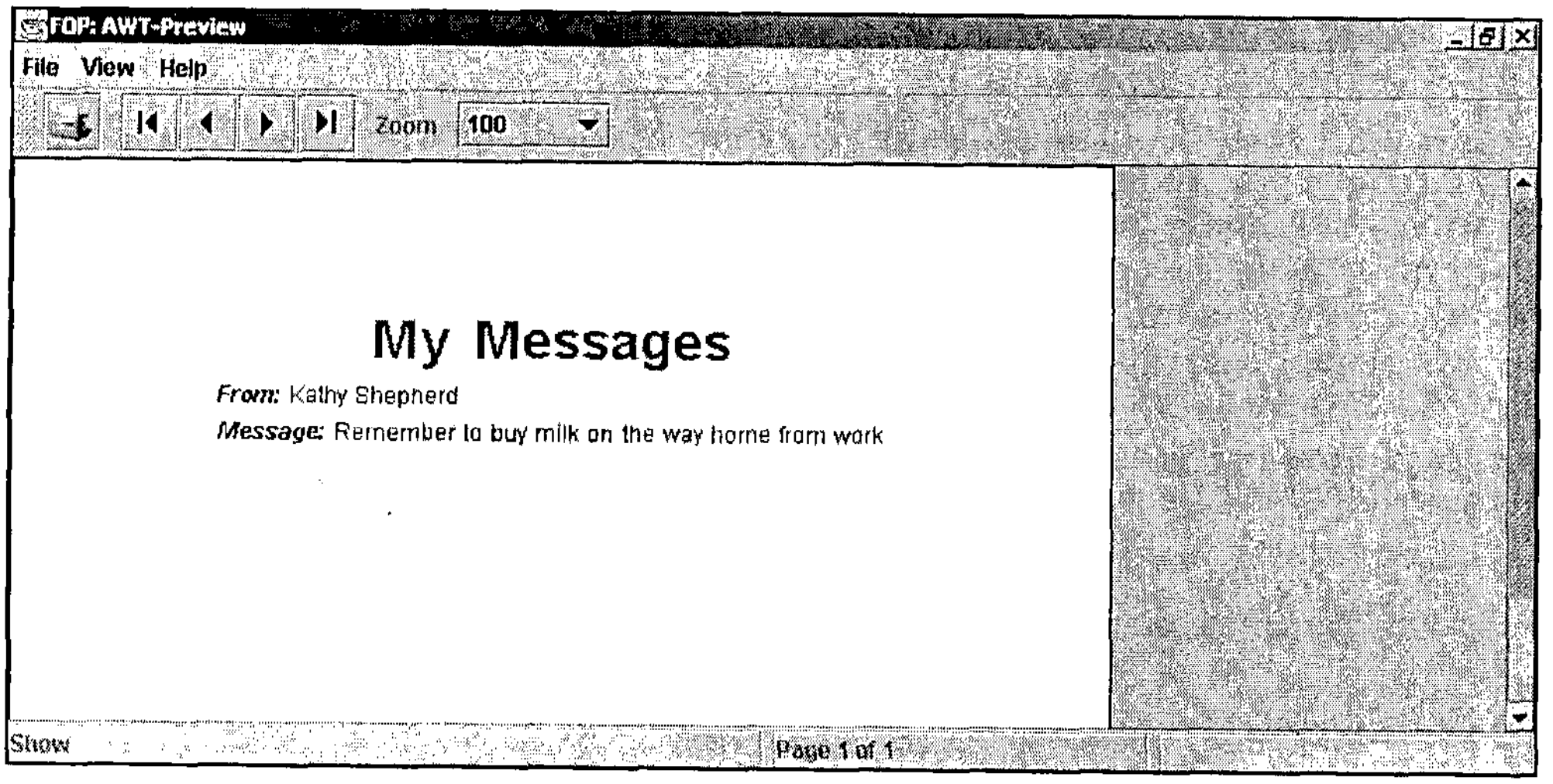


Рис. 15.5. Применение стилей к шрифтам в документе XSL-FO

Два новых блока начинаются в строках 24 и 29. Каждый из них содержит атрибуты `font-size` и `font-family`. Строка 9 содержит атрибут `margin-right="1.5in"`, который добавлен для того, чтобы элемент `region-body` располагался строго по середине, а значит, текстовый блок `My Messages` отображается по центру страницы. Результат показан на рис. 15.5.

Стили элементов

Если вы хотите изменить стилевое форматирование текстового фрагмента `From:` таким образом, чтобы он отличался от остальной части строки, `Kathy Shepherd`, вам необходимо выделить его в отдельный блок. Однако это приведет к разделению двух текстовых фрагментов с помощью разделительной линии. XSL-FO предоставляет очень простой способ разделения внутренних стилей с помощью вложенных элементов. Элемент `fo:inline` можно добавить к фрагменту текста, стиль которого должен отличаться от форматирования всей остальной части строки. Например, если вы хотите, чтобы такие части вашего документа, как `From:` и `Message:`, отображались полужирным курсивным начертанием, добавьте в документ следующие строки:

```
<fo:inline font-weight="bold"
          font-style="italic">From: </fo:inline>
```

Внесите необходимые изменения в документ и сохраните его под именем `message03.fo`. Результат показан на рис. 15.6.

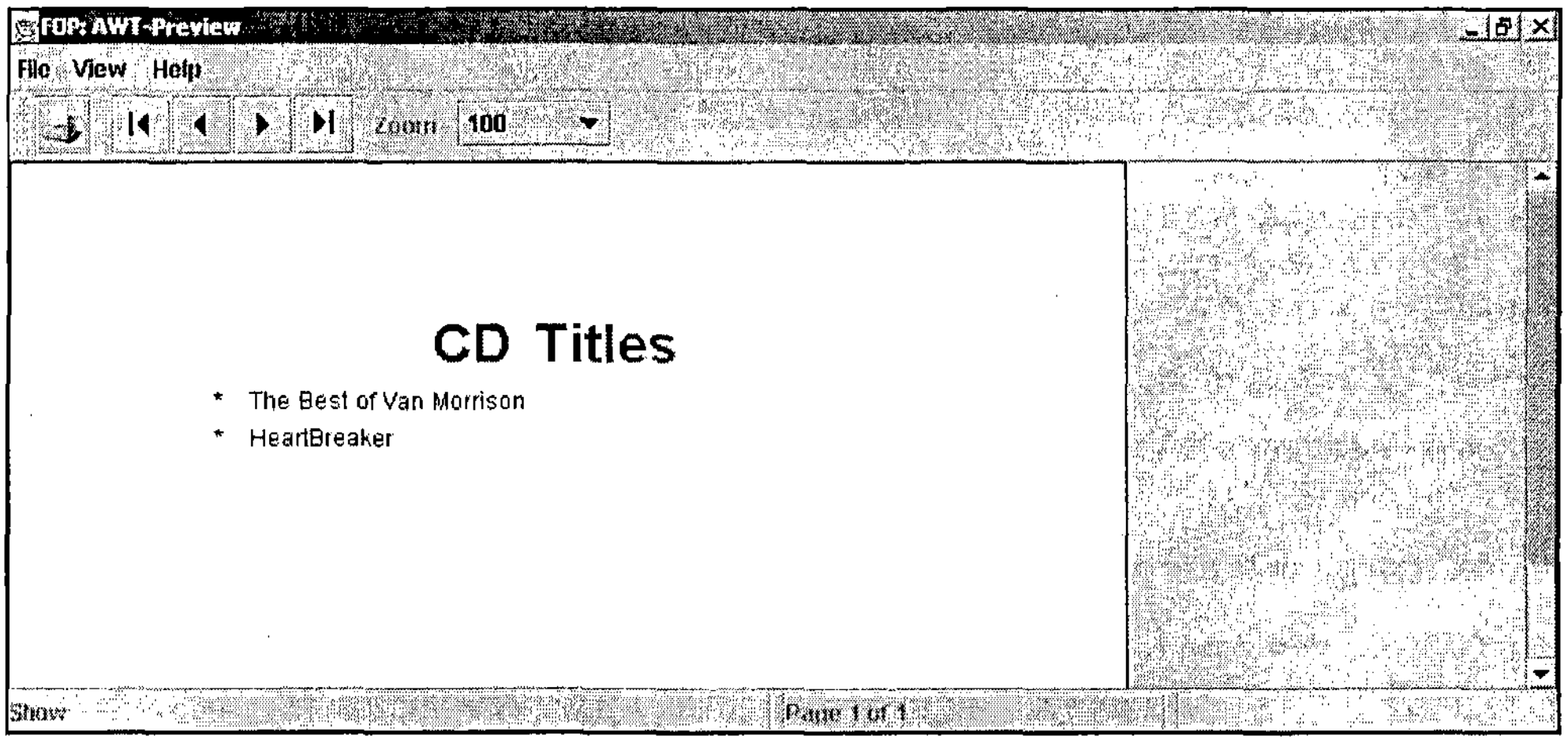


Рис. 15.6. Использование внутренних стилей XSL-FO

Таблицы XSL-FO

Таблицы в XSL-FO также состоят из строк и столбцов. Вы измените текст сообщения таким образом, чтобы оно отображалось в таблице, содержащей 2 строки и 2 столбца. Основной синтаксис представления таблицы приведен ниже:

```
<fo:table>

    <fo:table-column column-width="значение" />
    <fo:table-column column-width="значение"/>

    <fo:table-body>

        <fo:table-row>
            <fo:table-cell>
            <fo:block>
               Данные ячейки
            </fo:block>
             </fo:table-cell>
                ...сведения о других ячейках

        </fo:table-row>
                ...сведения о других строках

    </fo:table-body>
</fo:table>
```

Спецификация требует, чтобы вы определили строки и задали элемент `fo:table-column` с атрибутом `column-width` для каждого столбца в таблице. В нашем упражнении ширина левого столбца равна 2,5 см, а правого — 11,25 см:

```
<fo:table-column column-width="1in" />
<fo:table-column column-width="4.5in"/>
```

Отступ в 1,25 см (`padding=".05in"`) добавлен к ячейкам для того, чтобы края ячеек не касались границ таблицы. Цвет фона для элементов `From:` и `Message:` выбран `aqua` (цвет морской волны). Соответствующий фрагмент для первой ячейки приведен ниже:

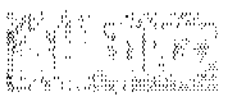
```
<fo:table-cell padding=".05in">
    <fo:block background-color="aqua">
        From:
    </fo:block>
 </fo:table-cell>
```

Как всегда, любые специальные пары атрибут–значение, которые вы хотите применить для стилевой разметки, например для определения цветов, границ ячеек или таблиц, цвета фона и т.д., должны задаваться как атрибуты соответствующих элементов. Полный документ, сохраненный под именем `message04.fo`, приведен в листинге 15.3.

Листинг 15.3. Кодирование таблицы XSL-FO `message04.fo`

```
1: <?xml version="1.0"?>
2:  <!-- листинг 15.3 - message04.fo -->
3:
4:  <fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">
5:
6:    <fo:layout-master-set>
7:      <fo:simple-page-master master-name="message-page">
8:        <fo:region-body margin-left="1.5in"
9:                margin-right="1.5in" margin-top="1in"/>
10:      </fo:simple-page-master>
11:    </fo:layout-master-set>
12:
13:    <fo:page-sequence master-name="message-page">
14:      <fo:flow flow-name="xsl-region-body">
15:
16:        <fo:block
17:             text-align="center"
18:               font-size="28pt"
19:               font-family="sans-serif"
20:               font-weight="bold"
21:               color="blue">My Messages
22:          </fo:block>
23:
24:      <fo:table border-style="solid">
25:
26:        <!-- column-width is required -->
27:        <fo:table-column column-width="1in" />
28:        <fo:table-column column-width="4.5in"/>
29:
30:        <fo:table-body>
31:
32:          <!-- first row of table -->
33:          <fo:table-row>
34:
35:             <!-- upper-left cell -->
36:             <fo:table-cell padding=".05in">
37:              <fo:block background-color="aqua">
38:                 From:
39:              </fo:block>
40:             </fo:table-cell>
41:
42:
```

```
43:            <!-- верхняя правая ячейка -->
44:            <fo:table-cell  padding=".05in">
45:             <fo:block>
46:                Kathy Shepherd
47:             </fo:block>
48:            </fo:table-cell>
49:          </fo:table-row>
50:
51:          <!-- вторая строка таблицы -->
52:          <fo:table-row>
53:
54:            <!-- нижняя левая ячейка -->
55:            <fo:table-cell  padding=".05in">
56:             <fo:block background-color="aqua">
57:                Message:
58:             </fo:block>
59:            </fo:table-cell>
60:
61:
62:            <!-- lower-right cell -->
63:            <fo:table-cell  padding=".05in">
64:             <fo:block>
65:                Remember to buy milk on the way home from work
66:             </fo:block>
67:            </fo:table-cell>
68:
69:          </fo:table-row>
70:
71:        </fo:table-body>
72:      </fo:table>
73:
74:      </fo:flow>
75:    </fo:page-sequence>
76:
77: </fo:root>
```



Строки 1—22 не претерпели никаких изменений по сравнению с предыдущим примером. Таблица начинается со строки 24 и включает атрибут `border-style`, который задает непрерывную линию границы вокруг всей таблицы. Ширины левого и правого столбцов задаются в строках 27 и 28 соответственно. Первая строка таблицы начинается в строке 33. Элементы, относящиеся к этой строке таблицы, находятся в строках 34—48. Верхняя левая ячейка таблицы задана в строках 36—40. Цвет фона ячеек в левом столбце — морской волны (строки 37 и 56). Параметры каждой из трех ячеек повторяются в строках 44, 55 и 63 соответственно. Строки 69—77 предназначены для закрытия всех открытых раньше элементов.

Резюме

Сегодня вы создали объекты форматирования XSL-FO и использовали их для стилевой разметки документа XML при отображении его на экране. Вы также научились создавать файлы PDF для печати документов или их распространения в электронной форме. Для проверки полученного кода использовался бесплатный интерпретатор FOP, доступный для различных платформ; его можно интегрировать в пользовательские Java-приложения. Вы узнали, что стандарт XSL Formatting Objects является под-

множеством рекомендации XSL. Остальную часть спецификации XSL образуют преобразования XSLT. С преобразованиями XSLT и их использованием для преобразования структуры документов XML вы познакомитесь завтра, в следующем уроке.

Вопросы и ответы

Какая связь между XSL, XSL-FO и XSLT?

Рекомендация XSL (Extensible Stylesheet Language) консорциума *W3C* состоит из двух частей — преобразований (XSLT) и языка описания объектов форматирования (XSL-FO). Язык преобразований XSLT, способный изменить структуру документа XML к новой структуре разметки, может функционировать независимо от XSL-FO. Язык преобразований XSLT может преобразовать структуру документа XML к другому документу XML, документу HTML, а также к любой другой разметке. Объекты форматирования XSL-FO идеально подходят для интерпретации при печати документов и при использовании различных агентов пользователей.

Какова роль исходного файла XML при использовании объектов форматирования XSL-FO? Каким образом можно превратить документ XML в документ PDF?

Документ XSL-FO — это экземпляр документа XML, который используется при выполнении операции, которую можно разделить на несколько этапов. Для преобразования документа XML в документ PDF первый сначала нужно обязательно преобразовать таким образом, чтобы он был документом XSL-FO. Для того чтобы это сделать, вы связываете исходный документ XML с таблицей стилей XSL для дальнейшей обработки с помощью синтаксического анализатора XSLT. В результате такой обработки мы получаете древовидную структуру XSL-FO. После этого древовидная структура XSL-FO обрабатывается с помощью интерпретатора, который преобразует XSL-FO к другому стандарту, например к формату PDF.

Каким образом интерпретатор FOP можно использовать в приложении для получения конечного результата на основе объектов форматирования?

Программу FOP можно внедрить в Java-приложение, инициализировав объект `org.apache.fop.apps.Drirer`. Всю необходимую документацию вы найдете по адресу `http://xml.apache.org/fop/`. Вы также можете добавить обработчик FOP в такой синтаксический анализатор SAX, как Xerces, используя указания, которые вы найдете на том же Web-узле. Версии синтаксического анализатора Xerces, написанные на Java, C++ и PERL, вы найдете по адресу `http://xml.apache.org/`. Это позволит вам получать результаты FOP, вызывая события SAX для получения объектов форматирования.

Могут ли графические изображения включаться в документ XSL-FO?

Внешние графические файлы могут связываться с документом XSL-FO с помощью элемента `fo:external-graphic`, значение атрибута `src` которого равно соответствующему идентификатору URI. Благодаря использованию элемента `fo:external-graphic` обработчик в состоянии добавлять графические изображения к полученным результатам. Синтаксис элемента `fo:external-graphic` приведен ниже:

```
<fo:external-graphic src="URI"/>
```

Упражнение

Проверьте свои знания, полученные в результате изучения настоящего урока. Ответ вы найдете в Приложении А "Ответы на вопросы".

Используя таблицу, в которой перечислены доступные объекты форматирования и данные из файла `cd.xml`, создайте маркированный список, содержащий названия компакт-дисков.

День 16-й

Язык преобразований XSLT

Язык преобразований XSLT (XSL Transformation Language) преобразует дерево документа XML в новую древовидную структуру. Эта новая древовидная структура представляет другой документ XML, документ HTML или документ в любом другом формате. XSLT — это описательный, управляемый событиями, основанный на правилах язык программирования, диалект XML.

Сегодня мы рассмотрим следующие темы.

- Каким образом язык XSLT преобразует документы XML из одной структуры к другой.
- Использование XSLT-обработчика XT и XML-редактора Architag.
- Богатый набор элементов XSL.T
- Манипулирование узлами, сортировка и применение стилей к данным вывода с использованием XSLT.

Преобразование из одной структуры к другой

Как вы узнали из урока дня 15-го "Расширяемый язык таблиц стилей и его объекты форматирования", расширяемый язык стилей XSL (Extendible Stylesheet Language) включает в себя язык преобразований XSLT и язык форматирования XSL-FO. Третий компонент первоначального предложения XSL, Xpath, вырос в отдельную спецификацию, выступая в качестве инструмента других технологий XML. Как и XSL-FO, XSLT — это приложение XML, что позволяет вам легко создавать и редактировать соответствующие документы с помощью обычного редактора. Из предыдущего урока вы узнали, что XSLT способен функционировать отдельно от XML. Сегодня вы преобразуете документы XML в правильные HTML-страницы, обойдясь без средств языка XSL-FO.

Несмотря на то, что вы будете работать с языком преобразований XSLT в HTML, XSLT предоставляет намного более широкие возможности. XSLT может преобразовать XML к другим формам текстовых документов или разметки, даже к другим документам XML. Преобразования XML -XML очень часто используются, когда надо привести документы в соответствие с требованиями определенных приложений. Например, вы хотите применить к документу XML преобразования XSLT для передачи сведений своему деловому партнеру. Или у вас счет оформлен в виде документа XML, а его нужно преобразовать к документу XML с другой структурой для ввода данных в систему заказов. XSLT предоставляет простые средства для подобных преобразований.

На момент написания настоящей книги XSL еще считается кандидатом в рекомендации консорциума *W3C*, все равно XSLT версии 1.0 — это уже полноценная рекомендация. Полный текст спецификации вы найдете по адресу `http://www.w3.org/TR/xslt`. Стандарт XSLT находится в более законченном состоянии, чем XSL, а это позволяет использовать язык преобразований XSLT независимо от XSL. Вы можете использовать язык преобразований XSLT независимо, что вы и будете делать в настоящем уроке, однако в соответствии с положениями консорциума *W3C* основное назначение XSLT — использование преобразований в рамках общего применения стилей к данным XML.

Использование языка преобразований для Web-дизайна — достаточно обычное явление. Неписаное правило гласит, что для решения проблемы вы можете использовать любой инструмент, который лучше всего для этого подходит. Вы когда-нибудь встречали человека, который стремился бы использовать для решения любых задач один определенный инструмент? Если вы каждый день используете программу электронных таблиц, разве вы будете вводить в ячейке 80 символов, после чего нажимать клавишу <Enter> и продолжать ввод текста, тем самым превращая ее в текстовый процессор?

Используя XML для интеллектуального хранения данных и для их эффективного преобразования, вы сможете воспользоваться рядом сильных качеств этого стандарта. Например, преобразуя XML к HTML, вы представите данные в формате, понятном для всех современных Web-броузеров. Как вы уже знаете, представление данных XML еще не реализовано в броузерах в полной мере, чего, конечно же, нельзя сказать об HTML.

Преобразование также можно встроить в приложения как часть стандартных преобразований документа. Давайте рассмотрим, каким образом броузер Internet Explorer отображает результат, когда вы загружаете в него страницу XML. Как вы уже знаете из урока дня 2-го, на самом деле Internet Explorer данные XML не отображает; вместо этого броузер выполняет преобразование XSL, используя стандартную таблицу стилей. Одновременно с установкой Internet Explorer на своем компьютере вы устанавливаете и целый ряд дополнительных компонентов, в том числе синтаксический анализатор MSXML, а также связанные с ним библиотеки. В состав пакета входит и специальная таблица стилей. При загрузке файла XML в Internet Explorer программа сначала проверяет, предоставили ли вы таблицу стилей, которая должна использоваться при отображении результатов. В противном случае Internet Explorer использует свою стандартную таблицу стилей — именно ту, которая обеспечивает выделение дескрипторов цветом, а также свернутое (раскрытое) дерево разметки и содержания. Если вы хотите увидеть стандартную таблицу стилей, запустите Internet Explorer версии 5.0 или более позднюю и введите следующий внутренний идентификатор URI в адресной строке:

```
res://msxml.dll/defaultss.xsl
```

Вы увидите уже знакомую древовидную структуру достаточно сложной таблицы стилей XSL, рассмотрение которой выходит за рамки настоящего урока. Однако уже в конце сегодняшнего занятия вы сможете определить основные правила и компоненты в этой таблице стилей.

Операционный XSLT

Преобразования имеют место в тот момент, когда средство преобразований XSLT обрабатывает документ XML и таблицу стилей XSL. Вы используете язык преобразований XSLT для получения таблиц стилей XSL, которые также являются документами XML. Таблица стилей XSL содержит инструкции для преобразования исходных документов XML от одного типа документов к другому, например от XML к HTML. Если говорить терминами структуры документов, то таблица стилей XSL определяет преобразование одного дерева узлов к другому. Значительная часть современных средств преобразования и редакторов уже обладает встроенной поддержкой XSLT.

Параметры обработки XSLT

Преобразования XSLT используются различными способами, поэтому вам следует выбирать инструменты, которые лучше всего соответствуют вашим задачам. Например, вы можете выполнить локальные преобразования, используя инструмент, запущенный на вашем компьютере. В этом случае вы запускаете инструмент и передаете ему имена документа XML и связанной с ним таблицы стилей. После этого инструмент должен выполнить все необходимые преобразования, а затем выдать результат. Подобные средства доступны для практически любых платформ, а многие еще и распространяются бесплатно и интегрируются в другие приложения. Объединив мощь средства преобразований с вашим кодом, вы сможете создать очень сложные решения. Полный список средств преобразования, редакторов и утилит, поддерживающих язык XSLT, вы можете найти по адресу `http://www.xslt.com/xslt_tools_editors.html` или `http://www.w3.org/Style/XSL/`.

Загрузите и установите обработчик XSLT (XT), написанный на Java, с Web-узла `http://jclark.com/xml/xt.html` для выполнения упражнений настоящего урока. Этот обработчик функционирует совместно с синтаксическим анализатором SAX, который вы установили еще в уроке 13.

Microsoft Internet Explorer поддерживает преобразование XML на стороне клиента с использованием таблиц стилей XSL. Это одна из наиболее практичных сред, в которой вы можете контролировать работу броузера. Броузеры компании *Netscape*, например, не поддерживают преобразование на стороне клиента; вам необходимо обеспечить, что все пользователи работают с совместимыми версиями Microsoft Internet Explorer. В любом случае, преобразование на стороне клиента обеспечивает простую быструю проверку полученных результатов. Для выполнения преобразований XML на стороне клиента вы станете использовать Internet Explorer уже сегодня.

Некоторые текстовые редакторы также обладают поддержкой преобразований XSLT, что предоставляет широкие возможности при разработке таблиц стилей. Вы можете загрузить бесплатный редактор и испытать его уже сейчас (`http://www.architag.com/xray`).

Вы также можете использовать сценарии на стороне сервера для вызова обработчика XML с целью преобразования документа XML к стандарту HTML перед отправкой его на Web-сервер. Этот подход оказывается очень важным в некоторых Web-ориентированных приложениях.

Правила поведения в организации: используем XSLT

В предыдущем уроке вы рассмотрели документ XML, содержащий правила поведения в организации и преобразуемый в объекты форматирования XSL для представления интерпретатору, чтобы получить документ в формате PDF. Немного изменив задачу, отказавшись от использования объектов форматирования и формата PDF, рассмотрим тот же документ XML, но преобразуем его к формату HTML для представления в локальной сети. Вам все равно потребуется несколько процессов и словарей для изменения документа XML с целью применения стилей к его содержанию. В частности, для преобразования документа XML к формату HTML вам потребуются следующие компоненты.

- Исходный документ XML, который с точки зрения преобразований XSLT представляет собой иерархическое дерево узлов.
- Таблица стилей XSL и древовидная структура, которая содержит сведения о семантике преобразований.
- Обработчик XSLT, который преобразует исходный документ XML и таблицу стилей XSL, представляя полученный результат в виде древовидной структуры.

В нашем примере вам потребуется документ XML в локальной сети, содержащий все правила работы, принятые в вашей организации. Как и раньше, этот документ обновляется регулярно для отражения любых изменений, которые вносятся в каждый из образующих его документов. Однако, вместо того чтобы создавать документ PDF, вам необходимо периодически обрабатывать документ для подготовки HTML-страницы, к которой сотрудники организации смогут при необходимости обращаться. Превратив подобное преобразование в периодический процесс, вы обеспечите то, что содержание Web-страницы всегда отражает текущее положение.

Язык XSLT идеально подходит для решения подобной задачи. Как и раньше, документ XML, который содержит сведения о внутреннем распорядке организации (`corp.xml`), связывается с таблицей стилей (`corp.xsl`). Эта таблица стилей идентифицирует правила совпадения и действия, необходимые для того, чтобы указать обработчику XSLT, каким образом выполнять подстановку элементов XML дескрипторами HTML для определения размера заголовков, выделения отдельных фраз, полей на страницах и т.д. Средство XSLT обрабатывает оба документа XML (`corp.xml` и `corp.xsl`) при преобразовании, после чего получается документ HTML (`corp.html`). Полная операция содержит меньше действий, чем процесс создания файла PDF с использованием объектов форматирования. В нашем случае получается результирующая древовидная структура HTML, а не структура FO. После получения документа HTML процесс преобразования считается законченным. Новая HTML-страница содержит необходимые сведения из документа `corp.xml`. Весь только что описанный процесс проиллюстрирован на рис. 16.1.

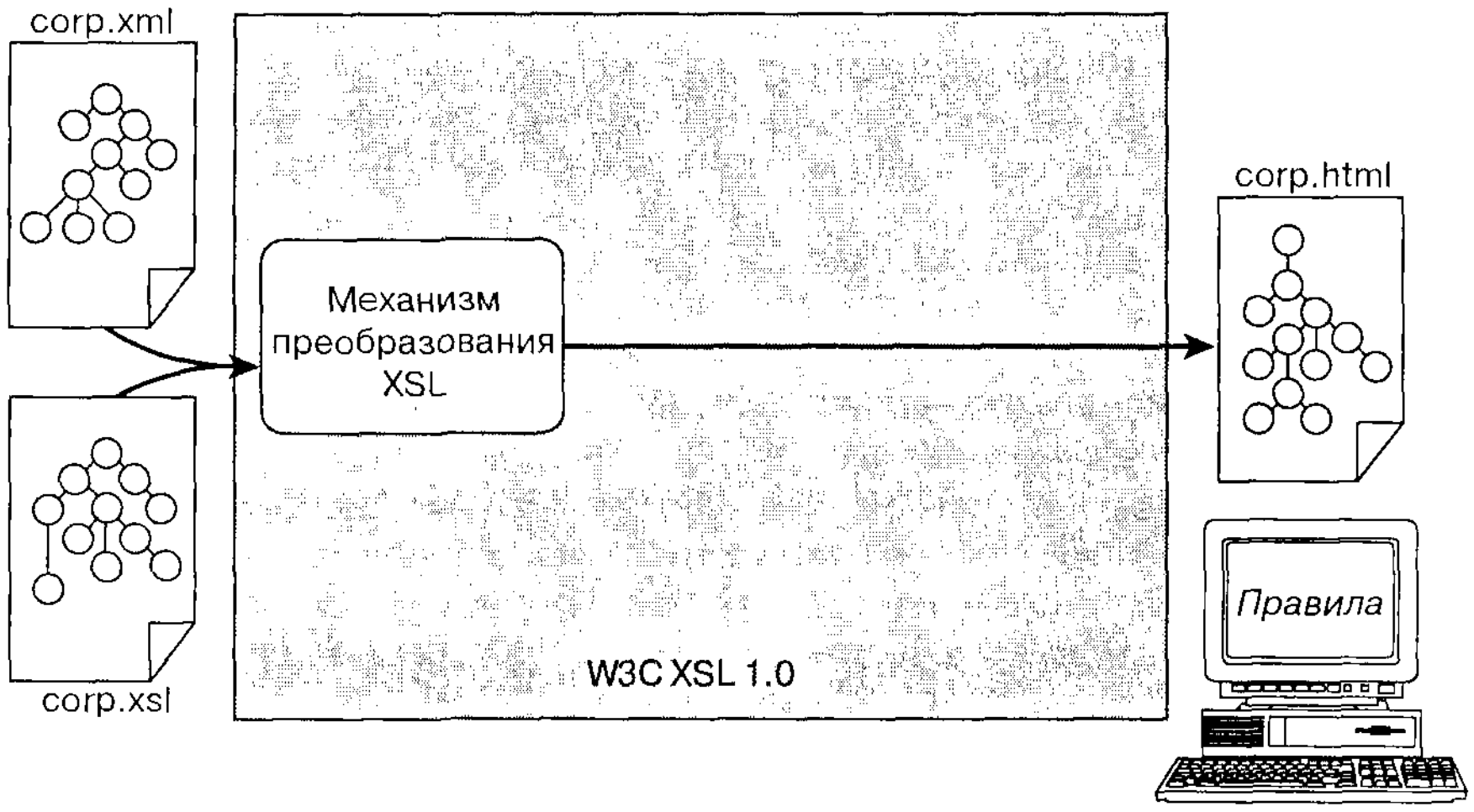


Рис. 16.1. Процесс преобразования XSL

К концу настоящего урока вы создадите таблицы стилей и используете обработчик XSLT независимо от такой части спецификации XSL, как XSL-FO. Созданные вами таблицы стилей XSL будут использоваться для преобразования экземпляров документов XML в правильные HTML-страницы для просмотра с помощью броузера. Как уже отмечалось, вы познакомитесь с несколькими средствами обработки.

Установка инструментов XSLT

Сейчас вы загрузите и установите несколько инструментов. Первый из них — это написанный на Java обработчик XT. К остальным относятся следующие.

- Бесплатный Java-обработчик SAXON (`http://users.iclway.co.uk/mhkay/saxon/instant.html`).
- XSLT-обработчик Napa, написанный на C++ (`http://www.tfitechnology.-com/xml/napa.html`).
- Apache Xalan (`http://xml.apache.org/xalan`).
- Fourthought 4Suite, написанный на Python (`http://4suite.org/`)
- Infoteria iXSLT, написанный на C++ (`http://www.infoteria.com/`).

XSLT-обработчик, работающий из командной строки

Обработчик, который вы будете использовать, называется XT. Это бесплатный XSLT-обработчик, написанный на языке Java Джеймсом Кларком (James Clark), одним из авторов спецификации XSLT. Загрузите установочные файлы обработчика XT с Web-узла `http://jclark.com/xml/xt.html`. Вам следует внимательно следовать всем указаниям, поскольку, как и в случае со всеми Java-программами, необходимо указать все необходимые пути, такие как `classpath` и `homepath`, а также создать все необходимые переменные окружения. В загруженной вместе с файлами обработчика документации вы найдете все необходимые инструкции, касающиеся версии обработчика для используемой вами операционной системы; по этой причине я не буду останавливаться на рассмотрении соответствующих действий в настоящем уроке.

Если вы работаете на компьютере под управлением операционной системы Windows, загрузите XSLT-обработчик XT в виде выполняемого файла Win32. Это самая простая версия программы с точки зрения установки. Сюда входит и SAX-обработчик XP, также написанный Джеймсом Кларком. Вам потребуется виртуальная машина Microsoft Java Virtual Machine (VM), поскольку обработчик XT написан таким образом, чтобы пользоваться всеми преимуществами этой виртуальной машины при работе в среде Windows. Если вам необходима копия виртуальной машины Java, загрузите ее с Web-узла `http://www.microsoft.com/java/vm/dl_vm40.htm`. Однако, если у вас уже установлен броузер Internet Explorer версии 4.01 или более поздней, отдельно загружать виртуальную машину Java вам не потребуется. (Вам все-таки придется это сделать, если ваш компьютер работает под управлением Windows XP, поскольку из-за трений, возникших с компанией *Sun Microsystems*, компании *Microsoft* пришлось убрать виртуальную машину Java из стандартной поставки операционной системы Windows XP. — *Прим. ред.*) Это означает, что если на вашем компьютере установлена операционная система Windows и броузер Internet Explorer версии 4.01 или более поздней, вам достаточно загрузить и установить синтаксический анализатор XT, распространяемый в виде выполняемого файла Win32. Пользователи других операционных систем, таких как Unix, Linux и Macintosh, должны загрузить полную версию XT и выполнить все указания по его установке.

Установив синтаксический анализатор XT, вы сможете использовать его для преобразований XSL. При этом вы будете вызывать обработчик из командной строки, используя следующий синтаксис:

```
java -Dcom.jclark.xsl.sax.parser=драйвер_SAX
      com.jclark.xsl.sax.Driver source.xml stylesheet.xsl result.html
```

Замените *`source.xml`* на полный путь и имя файла вашего исходного документа XML. Замените *`stylesheet.xsl`* в командной строке полностью определенным именем вашей таблицы стилей XSL, а *`result.html`* — именем файла, который вы создаете. Основная часть файлов, созданных в настоящем уроке, будет иметь расширение `.html`, поскольку вы обычно будете использовать преобразования для превращения документов XML в документы XSL. Однако синтаксический анализатор XT можно

использовать совместно с другой таблицей стилей, что позволит получить другую структуру XML, если необходимо.

Установив выполняемый файл Win32, вы будете вызывать обработчик из командной строки, используя следующий синтаксис:

```
xt source.xml stylesheet.xsl result.html
```

Редакторы XML, поддерживающие преобразования XSLT

Несколько редакторов XML поддерживают работу с таблицами стилей XSL. Некоторые из них способны выполнять преобразования, используя встроенные или указанные локальные средства, а также применять средства просмотра HTML для демонстрации полученных результатов. К редакторам, на которые следует обратить внимание, относятся следующие.

- XMLwriter — коммерческий редактор XML, поддерживающий преобразования XSLT (`http://xmlwriter.net/indexshtml`).
- XML-Spy — коммерческий программный продукт (`http://www.xmlspy.com`).
- IBM XSL Editor (`http://www.alphaworks.ibm.com/tech/xsleditor`).
- XSLDebugger (`http://vbxml.com/xsldebugger`).

Редактор, который вы будете использовать сегодня при выполнении упражнений, называется XRay XML Editor. Его можно загрузить с Web-узла `http://www.architag.com/xray`. Этот редактор обладает полной поддержкой проверки правильности и действительности документов XML с использованием схем DTD, XDR и XSD. Правда, сейчас вы будете использовать его возможности для преобразований XSLT. Можно создать документ XML в одном окне программы, а таблицу стилей XSL — в другом. После этого вы выполните преобразования в третьем окне и просмотрите результаты HTML или XML — в четвертом.

Загрузите и установите редактор XRay прямо сейчас, согласившись со всеми параметрами установки, заданными по умолчанию. Вы будете использовать XRay дальше в настоящем уроке для создания и проверки таблиц стилей XSLT.

Использование XSLT

При вызове обработчик XSLT начинает считывать дерево исходного документа XML, а также дерево документа XSL. Таблица стилей содержит набор правил для преобразований исходной древовидной структуры к древовидной структуре другого типа. Правила выражаются в виде наборов шаблонов, связанных с элементами исходного документа XML. Каждый шаблон содержит выражение XPath, что позволяет обработчику XSLT обнаружить целевой узел в исходном документе. В случае совпадения правила из шаблона применяются к содержанию обнаруженного элемента. Этот процесс продолжается до тех пор, пока все шаблоны и совпадения не будут успешно применены. Преобразования XSLT — это зависящий от отдельных событий процесс, который базируется на технологии обработки SAX. Правила, которые вы конструируете, приказывают обработчику выполнять определенные операции, в частности такие.

- Замена разметки XML разметкой HTML.
- Сортировка содержания элементов в соответствии с предоставленными алгоритмами.
- Сокрытие и отображение информации под управлением программы.
- Преобразование таблиц к графическому формату, используя, например, язык SVG.

Замечания о пространстве имен XSLT

Поскольку документ XSLT — это и документ XML, он состоит из элементов и атрибутов. Элементы в документе XSLT связываются с пространством имен XSLT:

```
http://www.w3.org/1999/XSL/Transform
```

Как вы помните из урока дня 8-го, пространство имен — это всего лишь метка, но особенная, поскольку она известна обработчику XML. Обработчик связывает выражения с указанным пространством имен. Таким образом, поведение программы связано с членами указанного набора элементов. Это означает, что обработчик XSLT, например, выполняет определенные инструкции в зависимости от того, с какими элементами он имеет дело. Каждый элемент в таблице стилей XSLT соответствует "событийной" инструкции, которая передается обработчику, а он, в свою очередь, вызывает событие, выполняющее необходимую функцию. К сожалению, еще не все обработчики реализуют одинаковый набор событий и функций. Спецификация консорциума *W3C* предлагает некоторый стандартный набор, но отличия от него все еще существуют. Иногда для определения иных наборов элементов указываются другие пространства имен. Это приводит вас к выбору пространств имен при использовании XSLT. Пространство имен консорциума *W3C* подходит для некоторых обработчиков, однако далеко не для всех из них. Например, если вы решили выполнить преобразования на стороне клиента в Internet Explorer, вам следует указать в программе XSLT другое пространство имен:

```
http://www.w3.org/TR/WD-xsl
```

К счастью, в большей части преобразований, с которыми вы будете иметь дело сегодня, пространство имен потребуется только для того, чтобы выбрать преобразования на стороне клиента.

Обработка событий на основе правил

Как уже отмечалось раньше, процесс, выраженный на языке XSLT, описывает правила преобразования исходной древовидной структуры к результирующей. Каждое правило шаблона — это отдельный объект, который никак себя не проявляет до тех пор, пока совпадение не приведет к активизации содержащихся в нем правил. По мере того как обработчик XSLT обрабатывает исходный документ узел за узлом, он выполняет правила в случае совпадения элементов или других конструкций. При совпадении выполняются все инструкции, указанные в правиле.

Вы можете представить себе, что выполнение каждого правила шаблона аналогично выполнению процедуры в другом языке программирования. Процедуры сохранены в коллекции или в документе XSL и вызываются при необходимости. Порядок следования правил в документе XSL никакой роли не играет. Каждое правило можно вызвать в случае совпадения выражения XPath. Другими словами, обработчик XSLT считывает исходный документ XML и передает содержание элементов соответствующим процедурам при обнаружении совпадений. Содержание совпавшего элемента передается процедуре XSLT, или правилу шаблона, для дальнейшей обработки. В упражнениях, которые вы выполните в настоящем уроке, обработка будет представлять собой размещение элементов HTML в древовидной структуре результатов.

Таким образом, инициализация шаблона для совпадающего элемента в исходном документе приводит к созданию части древовидной структуры результатов. Шаблон также может содержать специальные инструкции, которые указывают обработчику на то, какие именно действия необходимы для совпавших элементов.

Процедура выполняет определенный набор стандартных функций. Например, предположим, вам необходимо создать разметку HTML для определенного элемента,

например части маркированного списка на Web-странице. Подобные действия очень легко выполняются с шаблоном XSLT. Совпадение с элементом приведет к вызову инструкций, необходимых для помещения соответствующих дескрипторов в древовидную структуру результатов. После этого, как правило, выполнение определенной процедуры завершается. Элементы предоставляют функции языку XSLT. В этом уроке вы будете преимущественно использовать один из следующих вариантов выхода из шаблона, или функции XSLT, позволяющие завершить использование шаблона.

- Содержание элемента, который обрабатывается, доступно всем остальным правилам в наборе правил XSL (при этом используется элемент `xsl:apply-templates`).
- Содержание выводится непосредственно в древовидную структуру результатов (при этом используется элемент `xsl:value-of`).

Другие функции, доступные в XSLT, мы также рассмотрим. Основные функции XSLT перечислены в табл. 16.1.

Сначала обнаруживаются совпадения и затем выполняются определенные действия до тех пор, пока не будет преобразована вся исходная древовидная структура. Шаблон позволяет выполнять вычисления, а затем получать результаты на их основе. Он также способен предоставлять статическое содержание, которое размещается в древовидной структуре результатов. Независимо от того, какие действия выполняются шаблоном, вы можете использовать два варианта выхода из них, уже описанные раньше. Конечно же, доступны и другие функции выхода, но в этом уроке мы ограничимся элементами `xsl:apply-templates` и `xsl:value-of`.

Язык преобразований XSLT содержит элементы и атрибуты, как и документ XML. Особенность XSLT состоит в том, что с каждым элементом связана определенная функция. Элементы `xsl:apply-templates` и `xsl:value-of` — это всего два элемента, принадлежащих диалекту XSLT. Список элементов XSLT и связанных с ними функций приведен в табл. 16.1.

Таблица 16.1. Элементы XSLT

Элемент	Описание
`xsl:apply-imports`	Применяет шаблон из импортированной таблицы стилей
`xsl:apply-temlates`	Применяет шаблон к текущему элементу
`xsl:attribute`	Добавляет атрибут к ближайшему элементу
`xsl:attribute-set`	Определяет именованный набор атрибутов
`xsl:call-template`	Предоставляет способ вызова шаблона
`xsl:choose`	Предоставляет способ выбора среди нескольких вариантов в зависимости от выполнения тех или иных условий
`xsl:comment`	Создает комментарий XML
`xsl:copy`	Копирует текущий узел без его дочерних узлов и атрибутов в древовидную структуру результатов
`xsl:copy-of`	Копирует текущий узел со всеми его дочерними узлами и атрибутами в древовидную структуру результатов
`xsl:decimal-format`	Определяет символ (или строку), который используется при преобразовании чисел к строкам
`xsl:element`	Добавляет новый узел элемента в древовидную структуру результатов
`xsl:fallback`	Предоставляет способ определения альтернативы инструкциям, которые не реализованы

Окончание табл. 16.1

Элемент	Описание
`xsl:for-each`	Предоставляет способ создания цикла в потоке результирующих данных
`xsl:if`	Предоставляет способ создания условной инструкции
`xsl:import`	Импортирует таблицу стилей
`xsl:include`	Добавляет таблицу стилей
`xsl:key`	Предоставляет способ определения ключевого элемента
`xsl:message`	Помещает сообщение в результирующий документ
`xsl:namespace-alias`	Предоставляет способ связывания пространства имен с другим пространством имен
`xsl:number`	Помещает число определенного формата в результирующий документ
`xsl:otherwise`	Указывает, что именно должно выполняться в том случае, если ни одно из условий, указанных в элементе `xsl:when`, вложенном в элемент `xsl:choose`, не будет удовлетворено
`xsl:output`	Предоставляет способ контроля над преобразованием результирующего документа
`xsl:param`	Предоставляет способ определения параметров
`xsl:preserve-space`	Предоставляет способ определения обработки пробелов, отмечая их как значимые
`xsl:processing-instruction`	Заносит инструкцию на обработку в результирующий документ
`xsl:sort`	Предоставляет способ определения способа сортировки
`xsl:strip-space`	Предоставляет способ определения обработки пробелов, отмечая их как незначимые
`xsl:stylesheet`	Определяет корневой элемент таблицы стилей
`xsl:template`	Определяет шаблон для результирующего документа
`xsl:text`	Заносит текстовые данные в результирующий документ
`xsl:transform`	Определяет корневой элемент таблицы стилей
`xsl:value-of`	Создает текстовый узел и помещает значение в результирующую древовидную структуру
`xsl:variable`	Предоставляет способ определения переменной
`xsl:when`	Определяет условие, которое нужно проверить, а также действие, которое выполняется в случае положительного результата. Этот элемент всегда является дочерним по отношению к элементу `xsl:choose`
`xsl:with-param`	Предоставляет способ передачи параметров шаблонам

Преобразования XML в HTML

Лучший способ разобраться в природе правил шаблона — это самостоятельно их написать, после чего проверить. В первом упражнении этого урока вы возьмете простой документ XML и примените к нему преобразования XSLT для получения резуль-

тирующей древовидной структуры HTML, которую можно представить в окне броузера. Документ XML соответствует системе управления сообщениями, рассматриваемой нами на протяжении нескольких дней обучения. После создания документа XML вы разработаете документ XSLT.

Для связывания этих двух документов используйте элемент `stylesheet` в инструкции обработки в исходном документе XML. Этот элемент имеет два атрибута — `type` и `href`. Атрибут `type` может принимать два возможных значения — `text/xsl` для указания ссылки на таблицу стилей XSL или `text/css` для указания ссылки на каскадную таблицу стилей CSS. Атрибут `href` содержит идентификатор URI для связанного документа XSL. Синтаксис инструкции обработки приведен ниже:

```
<?xml-stylesheet type="text/xsl" href="таблица_стилей.xsl"?>
```

Эта инструкция не поддерживается всеми обработчиками XSLT, которые вы можете использовать для преобразований. Замечания об инструкциях обработки вы найдете на Web-узле консорциума *W3C* по адресу `http://www.w3.org/TR/xml-stylesheet/`. Иногда вы будете встречать этот элемент, указанный как `xml:stylesheet` (с двоеточием), а не `xml-stylesheet` (с дефисом). Консорциум *W3C* рекомендует использовать дефис. Когда обработчик, который может непосредственно связывать таблицу стилей с экземпляром документа XML, сталкивается с инструкцией обработки, он выполняет соответствующее связывание. Такой подход срабатывает как в Microsoft Internet Explorer, так и в Netscape 6.0. Консорциум *W3C* поддержал концепцию связывания таблиц стилей, чтобы поддержать инициативу основных поставщиков броузеров.

В листинге 16.1 приведен полный документ XML. Создайте этот документ с помощью текстового редактора и сохраните его под именем `message01.xml`.

Листинг 16.1. Исходный документ XML `message01.xml`

```
1: <?xml version="1.0"?>
2: <!-- листинг 16.1 - message01.xml -->
3:
4: <?xml-stylesheet type="text/xsl" href="msg01.xsl"?>
5:
6: <note>
7:      <head>My Messages</head>
8:
9: </note>
```



Инструкция обработки содержится в строке 4. Она связывает таблицу стилей `msg01.xsl`. Корневой элемент `note` (строки 6–9) содержит дочерний элемент `head` (строка 7) с текстовыми данными.

Преобразование в этом упражнении приведет к помещению содержания элемента `head` (`My Messages`) в раздел `<BODY>` результирующей HTML-страницы. Корневым элементом документа XSL является элемент `xsl:stylesheet`, в котором префикс `xsl:` связывает элементы XSL с пространством имен `http://www.w3.org/1999/XSL/Transform` или другим пространством имен, что определяется используемым обработчиком. Элемент `xsl:stylesheet` содержит атрибут пространства имен, а также атрибут `version`. В настоящее время существует только одна версия языка преобразований XSLT — `version = "1.0"`. Полностью элемент `xsl:stylesheet` выглядит следующим образом:

```
<xsl:stylesheet version = "1.0"
     xmlns:xsl = "http://www.w3.org/1999/XSL/Transform">
```

В этом упражнении вам потребуется всего одно правило шаблона. Напишите правило шаблона, которое будет преобразовывать содержание элемента `note`. Содержание элемента `note` — это элемент `head`. Вы будете использовать элемент `xsl:value-`

of для выбора (select) содержания элемента head и помещения его в результирующую древовидную структуру. Вам необходимо включить стандартные дескрипторы разметки HTML, необходимые для создания HTML-страницы. Полное правило шаблона выглядит следующим образом:

```
<xsl:template match = "note">
            <HTML>
                  <BODY><xsl:value-of select="head"/></BODY>
            </HTML>
</xsl:template>
```

Это правило выполняется в том случае, если обработчик обнаруживает совпадение для элемента note. После этого оно заносит дескрипторы <HTML> и <BODY> в результирующую древовидную структуру. В этот момент обработчик сталкивается с атрибутом xsl:value-of. Как уже отмечалось раньше, это приводит к переносу выбранного содержания в результирующий документ. После этого шаблон должным образом завершает дескрипторы <HTML> и <BODY>. Полный вариант документа XSL, сохраненный под именем msg01.xsl, приведен в листинге 16.2.

Листинг 16.2. Документ преобразования XSL msg01.xsl

```
1: <?xml version="1.0"?>
2: <!-- листинг 16.2 - msg01.xsl -->
3:
4: <xsl:stylesheet version = "1.0"
5:           xmlns:xsl = "http://www.w3.org/1999/XSL/Transform">
6:
7:       <xsl:template match = "note">
8:             <HTML>
9:                   <BODY><xsl:value-of select="head"/></BODY>
10:            </HTML>
11:       </xsl:template>
12:
13: </xsl:stylesheet>
```

Корневой элемент (xsl:stylesheet) начинается в строке 4, вместе с атрибутами xmlns и version.

Создав и сохранив оба документа, обработайте их с помощью XSLT-обработчика XT для создания результирующего документа msg01.html. Если ваш компьютер работает под управлением Windows, инструкция в командной строке должна выглядеть приблизительно следующим образом:

```
xt message01.xml msg01.xsl msg01.html
```

Если вы используете другую версию XT, инструкция командой строки может выглядеть следующим образом:

```
java -Dcom.jclark.xsl.sax.parser=драйвер_SAX
      com.jclark.xsl.sax.Driver message01.xml msg01.xsl msg01.html
```

Если вы не получили сообщений об ошибках, вы должны получить HTML-документ msg01.html, который при просмотре с помощью броузера должен выглядеть так, как показано на рис. 16.2. Поскольку вы получили документ HTML, его можно просмотреть с помощью любого броузера, который поддерживает стандарт HTML. Для представления полученных результатов мы будет использовать несколько различных версий броузеров для того, чтобы подчеркнуть данный момент.

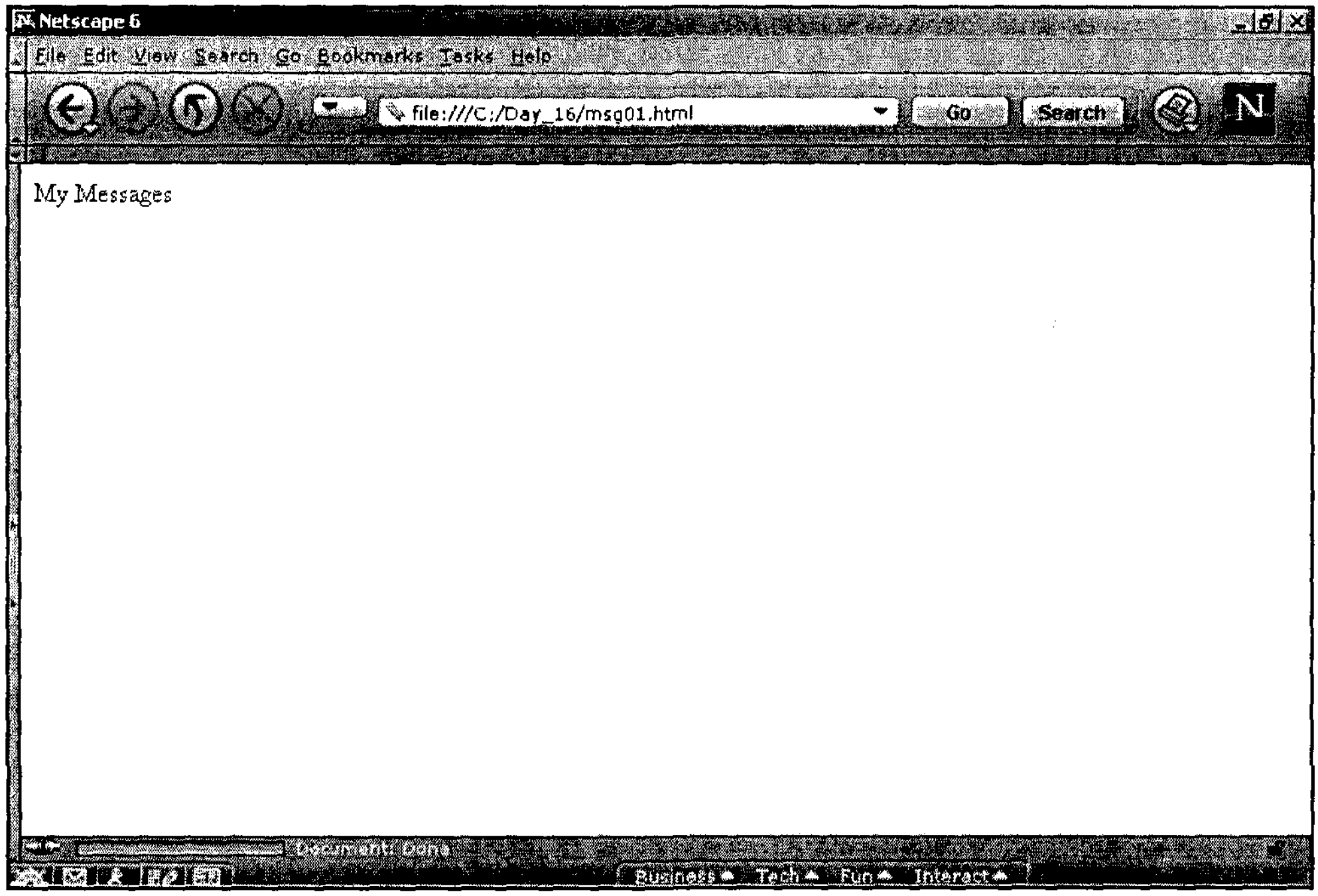


Рис. 16.2. Документ HTML, представленный в окне Netscape 6

Не забывайте о том, что язык преобразований XSLT способен представлять результаты в одном из нескольких форматов. HTML — это только один из возможных вариантов. Он полезен в том случае, если вам нужно получить результат для представления в Web, чтобы его можно было просмотреть с помощью различных броузеров. Однако, используя XSLT, вы можете создать и другие диалекты XML или текстовые файлы, как уже отмечалось раньше. Это обеспечит некоторые преимущества, если вам необходимо создать, например, документ XML с использованием языка WML (Wireless Markup Language) для отображения с помощью беспроводных устройств с поддержкой Web-технологий, таких как сотовые телефоны или персональные цифровые помощники PDA.

Итерационная обработка XSLT

Поскольку порядок следования правил роли не играет, в XSLT можно очень легко использовать определенное правило шаблона многократно. Все, что необходимо для выполнения правила, — наличие успешного совпадения. Если в вашем экземпляре документа XML присутствуют повторяющиеся элементы, набор правил можно настроить для обработки каждого из них. В следующем упражнении у вас появится возможность увидеть результат подобного итеративного процесса.

Начните с того, что увеличьте размер документа XML. Вам потребуются некоторые данные для последующих упражнений. Документ должен содержать корневой элемент `note` с несколькими дочерними элементами `msg`. Каждый из элементов `msg` содержит дочерние элементы `source`, `from` и `message` с текстовыми данными. Этими данными могут быть любые сведения, однако структура элементов должна быть именно такой, как указано; только в этом случае преобразования будут успешно выполнены. В листинге 16.3 приведен пример документа XML. Создайте подобный документ и сохраните его под именем `message02.xml`.

Листинг 16.3. Более сложный исходный документ XML message02.xml

```
<?xml version="1.0"?>
<!-- листинг 16.3 - message02.xml -->

<?xml-stylesheet type="text/xsl" href="msg02.xsl"?>

<note>
    <msg>
      <source>phone</source>
      <from>Kathy Shepherd</from>
      <message>Remember to buy milk on the way home from work</message>
    </msg>

    <msg>
      <source>e-mail</source>
      <from>Greg Shepherd</from>
      <message>I need some help with my homework</message>
    </msg>

    <msg>
      <source>e-mail</source>
      <from>Kristen Shepherd</from>
      <message>Please play Scrabble with me tonight</message>
    </msg>

    <msg>
      <source>pager</source>
      <from>Kathy Shepherd</from>
     <message>Buy a bottle of wine to take to the party tonight</message>
    </msg>

    <msg>
      <source>phone</source>
      <from>Kristen Shepherd</from>
      <message>Can you drive me to my friend's house?</message>
    </msg>

    <msg>
      <source>e-mail</source>
      <from>Greg Shepherd</from>
      <message>Meet me at the library</message>
    </msg>

    <msg>
      <source>e-mail</source>
      <from>Kathy Shepherd</from>
      <message>Bob returned your phone call</message>
    </msg>

    <msg>
      <source>pager</source>
      <from>Kathy Shepherd</from>
      <message>Pick up your shirts from the Dry Cleaners</message>
    </msg>

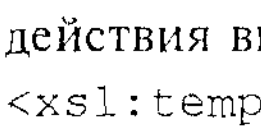
</note>
```



Инструкция обработки XSL в строке 4 определяет таблицу стилей XSL `msg02.xsl`. Корневой элемент `note`, который начинается в строке 6, содержит несколько дочерних элементов `msg`. Каждый из элементов `msg` содержит дочерние элементы `source`, `from` и `message` с текстовыми данными.

В этом упражнении вы создадите два шаблона. Один из них относится к элементу `note` и приводит к созданию стандартных дескрипторов HTML, добавляемых к содержанию элемента `note`. Для этого вам потребуется использовать элемент `xsl:template`, значение атрибута `match` которого должно совпадать с `note`. После этого вы можете добавить стандартные дескрипторы HTML, такие как `HTML`, `HEAD` или `BODY`. Первый шаблон выглядит следующим образом:

```
xsl:template match="note">
        <HTML>
          <HEAD>
             <TITLE>Messages</TITLE>
          </HEAD>
          <BODY>
             <H1>My Messages</H1>
                 <xsl:apply-templates/>
          </BODY>
        </HTML>
</xsl:template>
```



При выполнении этого правила элементы `<HTML>`, `<HEAD>`, `<TITLE>Messages </TITLE>`, `</HEAD>`, `<BODY>` и `<H1>My Messages</H1>` добавляются непосредственно в древовидную структуру, которая в данном случае представляет собой HTML-страницу, которую вы создаете. Следующая строка — элемент `xsl:apply-templates`. Этот элемент "предоставляет" содержание совпавшего элемента, `note`, всем остальным правилам в документе XSLT. Поскольку это корневой элемент, он содержит только элементы. В частности, в элементе `note` содержатся элементы `msg`. Поскольку для элемента `msg` также определен шаблон, обработчик приступит к работе над этим шаблоном. После этого он возвратится к обработке дальнейшей части документа, чем поместит в результирующий документ элементы `BODY` и `HTML`.

Второй шаблон относится к элементу `msg`. Именно к этому шаблону переходит обработчик, когда встречает элемент `xsl:apply-templates` в первом шаблоне. Правила этого шаблона создадут несколько элементов разметки HTML и разрешат содержанию элемента `msg` находиться в результирующей древовидной структуре. Более подробные действия вы выполните дальше. Соответствующее правило шаблона приведено ниже:

```
<xsl:template match="msg">
          <P>Here is a message:<BR/>
            <xsl:apply-templates/>
          </P><BR/>
</xsl:template>
```

Элемент `xsl:apply-templates` во втором правиле касается содержания элемента `msg`; поскольку дочерние элементы не содержат правил, элемент `xsl:apply-templates` передает содержание элемента `msg` непосредственно в результирующее дерево. В некоторых случаях выполнение подобных действий оказывается проблематичным.

Полный документ XSLT приведен в листинге 16.4. Создайте его и сохраните под именем `msg02.xsl`.

Листинг 16.4. Итеративная обработка с использованием XSLT `msg02.xsl`

```
1: <?xml version="1.0"?>
2: <!-- листинг 16.4 - msg02.xsl -->
3:
4: <xsl:stylesheet version="1.0"
5:       xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
6:
7:      <xsl:template match="note">
8:         <HTML>
9:           <HEAD>
10:             <TITLE>Messages</TITLE>
11:          </HEAD>
12:          <BODY>
13:             <H1>My Messages</H1>
14:                 <xsl:apply-templates/>
15:          </BODY>
16:        </HTML>
17:      </xsl:template>
18:
19:      <xsl:template match="msg">
20:            <P>Here is a message:<BR/>
21:              <xsl:apply-templates/>
22:            </P><BR/>
23:      </xsl:template>
24: </xsl:stylesheet>
```



Первое правило шаблона (строки 7–17) относится к элементу `note` (строка 7). Несколько стандартных дескрипторов HTML помещаются в результирующую древовидную структуру (строки 8–13), после чего элемент `xsl:apply-templates` передает содержание элемента `note` всем остальным правилам шаблона. Содержанием элемента `note` оказывается элемент `msg`, поэтому второй шаблон применяется именно к нему. Второй шаблон вступает в силу, поэтому в ходе обработки в результирующий документ помещаются дескрипторы HTML из стоки 20. Элемент `xsl:apply-templates` передает содержание элемента `msg` всем остальным правилам шаблона. Никакие другие правила не применяются. Содержание дочерних элементов элемента `msg` передаются в результирующую древовидную структуру в момент обработки элемента `xsl:apply-templates`. После обработки всех элементов `msg` управление передается строке 15, в которой закрываются все дескрипторы HTML.

Создав пример кода, используйте обработчик XT для получения документа `msg02.html` и просмотра его в окне броузера. Как уже отмечалось раньше, поскольку вы получаете обычный HTML, его можно обработать практически любым броузером. На самом деле, одна из причин того, что XSLT очень часто используется для преобразования документов XML к формату HTML, заключается в том, что создается результат, который не зависит от определенного броузера, операционной системы или платформы. Например, на рис. 16.3 приведен результат обработки документа `msg02.html` с помощью Web-броузера консорциума *W3C* Amaya. Amaya -- это броузер и редактор, распространяемый консорциумом *W3C* в качестве Web-инструмента общего назначения (`http://www.w3.org/Amaya/`).

Некоторые интересные моменты в работе кода, представленного в листинге 16.4, могут привести к тому, что дальнейшая обработка XSLT покажется запутанной. Второе правило передает содержание дочерних элементов элемента `msg`, несмотря на то, что

вы ему "подобных" указаний не давали. Этот подход срабатывает замечательно, однако существуют и интуитивно более понятные способы передачи содержания элементов напрямую в результирующую древовидную структуру.

Поскольку XSLT позволяет использовать выражения XPath в качестве значений определенных атрибутов, вы можете значительно улучшить свой контроль над выбором узлов, используя логику, с которой вы познакомились при изучении урока дня 9-го. Например, вы можете выбрать текстовые узлы, используя выражение XPath `text()` или узел `self` с точкой (.). Эти выражения позволят заменить явные имена элементов, использованные раньше в качестве значений атрибутов `match` или `select`.

В текущем упражнении вы можете, а возможно, и должны создать шаблон, который совпадает с узлами XPath `text()` и передает полученный текст на HTML-страницу с помощью элемента `xsl:value-of`, о котором мы говорили раньше. Явно обработав текстовые данные подобным образом, вы получите необходимые результаты, после чего сможете добавить к ним необходимое форматирование. Если вы хотите создать подобное правило, оно может выглядеть достаточно просто:

```
<xsl:template match="text()">
        <xsl:value-of select="."/>
 </xsl:template>
```

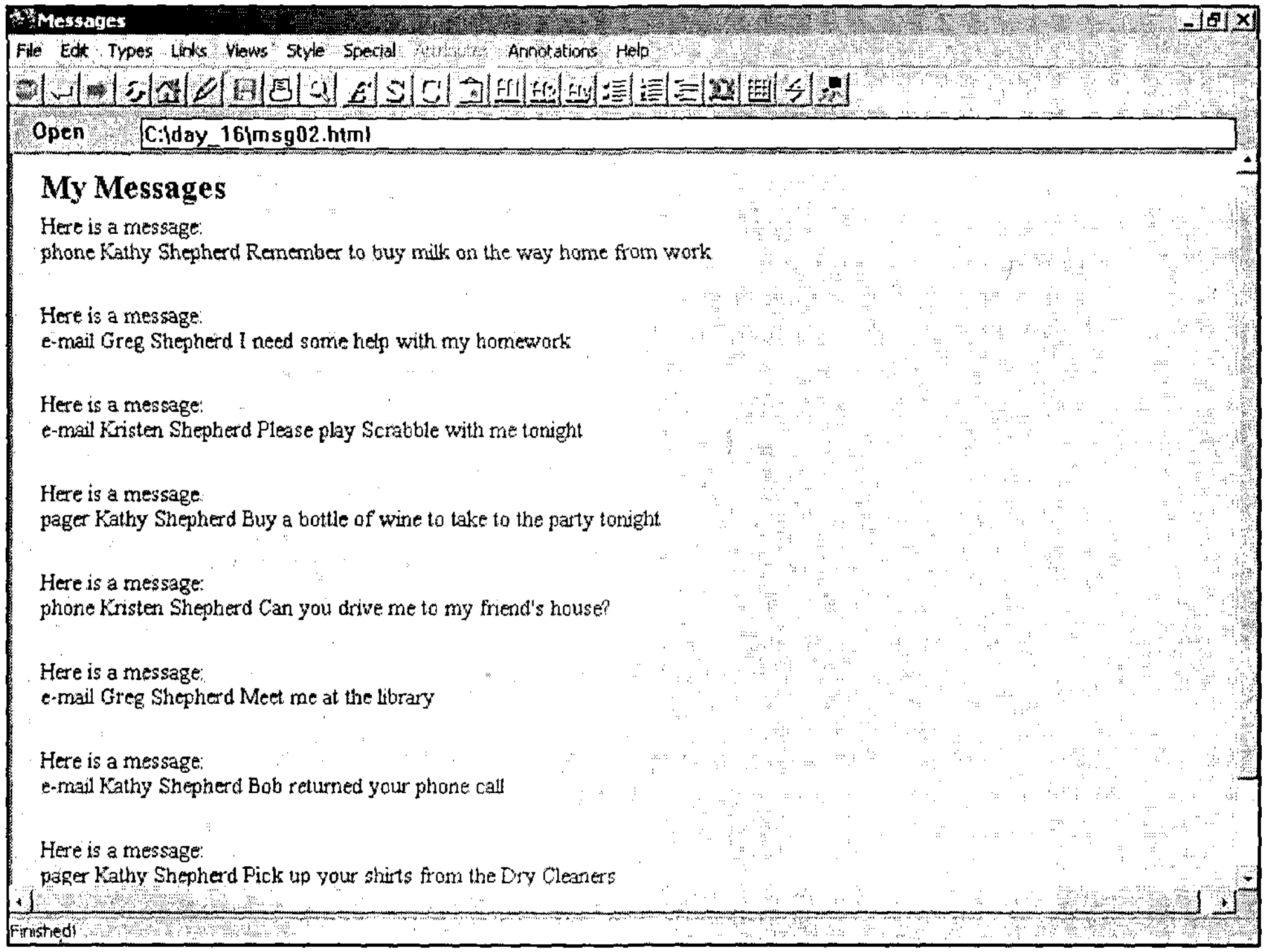


Рис. 16.3. Результаты итерационной обработки XSLT, представленные в окне броузера Amaya

Добавьте это правило шаблона в свой документ XSL и сохраните его под именем `msg03.xsl`. Для того чтобы убедиться в том, что все работает должным образом, преобразуйте документ `message02.html` с помощью обработчика XT и схемы `msg03.xsl` для создания HTML-файла `msg03.html` с целью просмотра его содержания в окне

броузера. В документе `message02.html` измените атрибут `href` (строка 4 в листинге 16.3) в инструкции обработки `xsl:stylesheet` для указания на файл `msg03.xsl`. Полученная инструкция примет следующий вид:

```
<?xsl:stylesheet type="text/xsl" href="msg03.xsl"?>
```

Полный вариант документа `msg03.xsl` приведен в листинге 16.5.

Листинг *16.5*. Добавление совпадения узлов `msg03.xsl`

```
1:  <?xml version="1.0"?>
2:  <!-- листинг 16.5 - msg03.xsl -->
3:
4:  <xsl:stylesheet version="1.0"
5:       xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
6:
7:      <xsl:template match="note">
8:         <HTML>
9:           <HEAD>
10:              <TITLE>Messages</TITLE>
11:           </HEAD>
12:           <BODY>
13:              <H1>My Messages</H1>
14:                 <xsl:apply-templates/>
15:           </BODY>
16:        </HTML>
17:     </xsl:template>
18:
19:     <xsl:template match="msg">
20:           <P>Here is a message:<BR/>
21:             <xsl:apply-templates/>
22:           </P><BR/>
23:     </xsl:template>
24:
25:     <xsl:template match="text()">
26:           <xsl:value-of select="."/>
27:     </xsl:template>
28:
29: </xsl:stylesheet>
```

АНАЛИЗ Шаблон, совпадающий с текстовыми узлами (`match="text()"`), добавлен в строках 25–27 настоящего листинга. Выражение XPath `"text()"` возвращает дочерние текстовые узлы контекстного узла. Выражение XPath `xsl:value-of select="."` указывает обработчику на необходимость передачи всего связанного с текущим узлом текста в результирующую древовидную структуру (строка 26). Вы должны помнить из урока дня 9-го, что точка (.) указывает на текущий узел в дереве XML.

Создание HTML-страницы с помощью XSLT

В предыдущем примере вы преобразовали XML в HTML, однако полученный результат не совсем вас удовлетворил. Содержание элементов `source`, `from` и `message` в результирующем документе представлялось в виде одной строки символов. В этом упражнении вы создадите правила шаблона, которые будут совпадать с этими элементами и добавлять содержание каждого из них в ячейку HTML-таблицы. Это очень полезный процесс, который очень легко реализовать, даже в случае весьма большого исходного документа XML.

Первый шаблон будет относиться к корневому элементу. Однако теперь, вместо того чтобы указывать корневой элемент, используйте выражение XPath для корневого узла, которое, как вы можете помнить из урока 9-го, представляется в виде косой (/). Независимо от того, определили вы корневой узел явно или использовали для этого выражение XPath, результат будет одинаковым. Оба подхода работают одинаково и приводят к одному и тому же результату. Несмотря на то, что вы увидите оба подхода в действии, краткая форма выражения набирает все большую популярность. В первом шаблоне добавьте дескрипторы HTML для создания таблицы, со значением атрибута `border`, равным `1`, и значением атрибута `width`, равным `100%`. Первый шаблон выглядит следующим образом:

```
<xsl:template match="/">
        <HTML>
          <HEAD>
             <TITLE>Messages</TITLE>
          </HEAD>
          <BODY>
             <H1>My Messages</H1>
             <TABLE BORDER="1" WIDTH="100%">
                 <xsl:apply-templates/>
             </TABLE>
          </BODY>
        </HTML>
</xsl:template>
```

Новыми элементами являются дескрипторы `TABLE`, а также выбор корневого узла с помощью косой. Затем вам необходимо создать шаблон для строк таблицы, который бы соответствовал элементам `msg`. Он выглядит следующим образом:

```
<xsl:template match="msg">
        <TR>
            <xsl:apply-templates/>
        </TR>
</xsl:template>
```

Следующее правило должно обеспечить наполнение ячеек таблицы содержанием элементов `source`, `from` и `message`. Вы можете поинтересоваться, зачем вам всего один шаблон для обработки содержания трех элементов. Благодаря интеграции выражений XPath в XSLT вы можете использовать оператор "|" в качестве логического ИЛИ между именами элементов, к которым вы хотите применить один и тот же шаблон. В результате правило может выглядеть следующим образом:

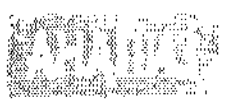
```
<xsl:template match="source | from | message">
        <TD>
            <xsl:apply-templates/>
        </TD>
</xsl:template>
```

Объединив все это, вы получите результат, представленный в листинге 16.6.

Листинг 16.6. Создание таблиц с помощью XSLT `msg04.xsl`

```
1: <?xml version="1.0"?>
2: <!--листинг 16.6 - msg04.xsl -->
3:
4: <xsl:stylesheet version="1.0"
5:      xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
6:
7:     <xsl:template match="/">
8:        <HTML>
9:          <HEAD>
```

```
10:             <TITLE>Messages</TITLE>
11:          </HEAD>
12:          <BODY>
13:             <H1>My Messages</H1>
14:             <TABLE BORDER="1" WIDTH="100%">
15:                 <xsl:apply-templates/>
16:             </TABLE>
17:          </BODY>
18:       </HTML>
19:    </xsl:template>
20:
21:    <xsl:template match="msg">
22:        <TR>
23:            <xsl:apply-templates/>
24:        </TR>
25:    </xsl:template>
26:
27:    <xsl:template match="source | from | message">
28:        <TD>
29:            <xsl:apply-templates/>
30:        </TD>
31:    </xsl:template>
32:
33: </xsl:stylesheet>
```



Строка 7 соответствует корневому узлу документа XML и применяет первый шаблон ко всему экземпляру документа XML. Дескрипторы TABLE в строках 14 и 16 образуют остальную часть результирующей древовидной структуры. Шаблон в строках 21–25 определяет две строки, выбирая элементы-контейнеры msg. В строке 27 показано, каким образом можно применять одно правило к трем различным элементам.



> В некоторых примерах в этом уроке используется обработчик XT, а в других — XML-редактор XRay от компании *Architag*. Вы можете следовать этому и использовать именно те инструменты, о которых говорится в том или ином упражнении, а можно использовать какие-то другие инструменты. Например, XRay работает только на Windows-системах, а пользователи UNIX могут выбирать среди XT, Xalan и целого ряда других инструментов. Независимо от того, какое средство XSLT вы выберете, код, показанный в примерах, можно обработать без внесения каких-либо значительных изменений. В любом случае тщательно изучите документацию, поставляемую вместе с тем или иным средством, чтобы ознакомиться с доступными параметрами выполнения.

Если вы хотите полностью следовать моим рекомендациям, используйте XML-редактор XRay для обработки примера, выполнив следующие действия.

1. Сохранив оба документа, запустите редактор XRay и откройте документ `message02.xml`. Документ появится в окне приложения.
2. Откройте документ `msg04.xsl`. Документ появится в новом окне приложения.
3. Открыв оба документа, выберите команду New XSLT Transform из меню File. Появится третье окно программы, содержащее большое пустое поле в центре экрана, а также два текстовых поля в верхней части окна. Верхнее текстовое поле называется XML Document.
4. Щелкните на кнопке со стрелкой, чтобы раскрыть текстовое поле XML Document и выбрать документ `message02.xml`.

5. Выполнив те же действия, выберите документ `msg04.xsl` во втором текстовом поле XSL Stylesheet. После этого результаты преобразования будут представлены в большом текстовом поле.
6. Выберите команду New HTML View из меню File, после этого раскройте список, чтобы выбрать новый вариант представления результатов, Transform 1. Однако обратите внимание, что в вашем случае этот вариант может называться Transform 2 или Transform 3. Это связано с тем, что после каждого преобразования редактор XRay увеличивает соответствующее число. Вы должны увидеть результат, подобный тому, что вы видите при использовании броузера Internet Explorer 5.0. Таблица, отображенная в окне редактора XRay, показана на рис. 16.4.
7. Из меню Window выберите один из вариантов взаимного расположения окон. Теперь вы должны иметь возможность увидеть все окна редактора XRay одновременно. Все окна динамически связываются; если вы внесете изменения в одном из них, вы немедленно увидите измененные результаты и других окнах. Подобные возможности превращают редактор XRay в замечательный инструмент разработки.

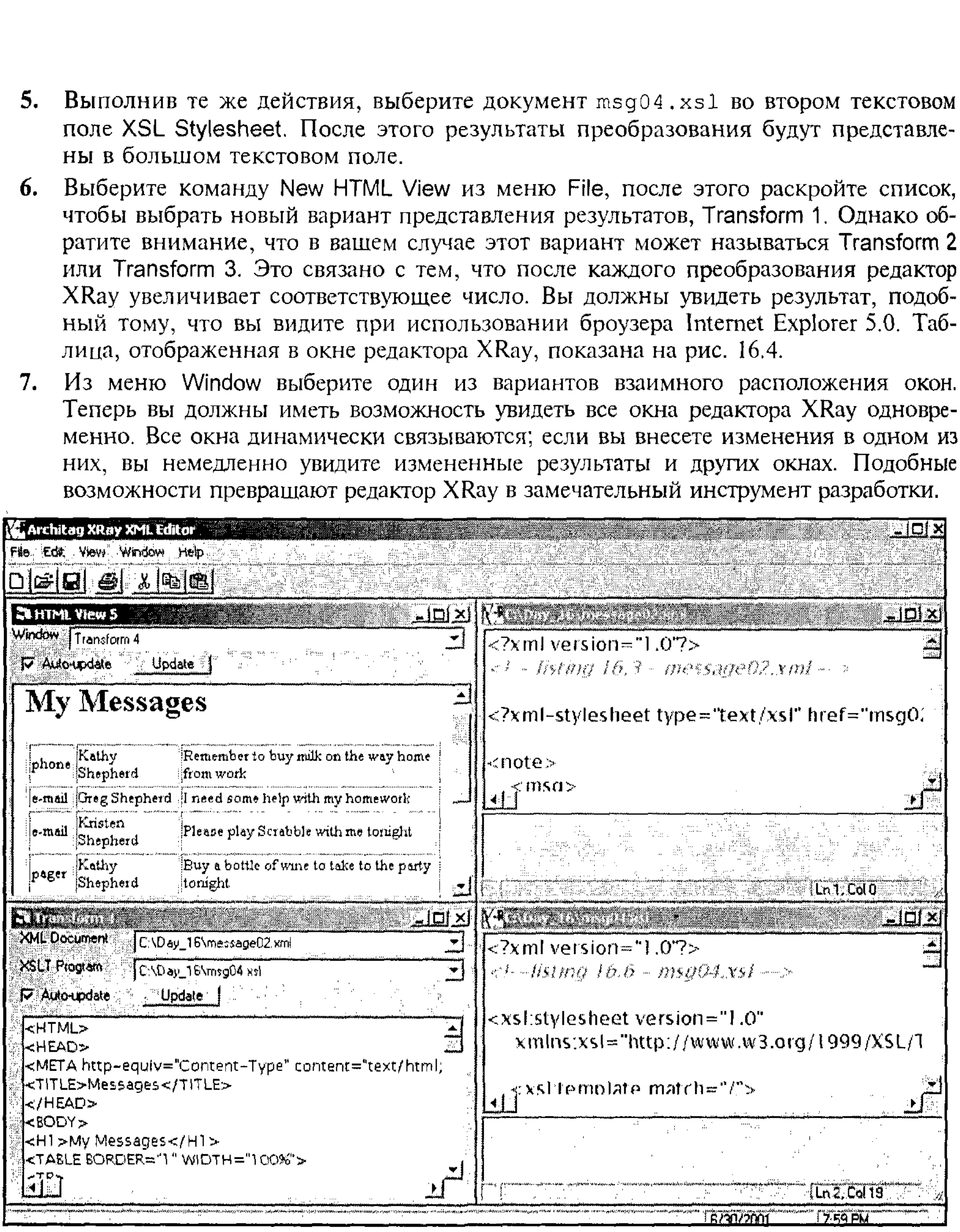


Рис. 16.4. Преобразование XSLT в виде таблицы представлено в окне редактора XRay

Сортировка XSLT

XSLT предлагает широкие возможности для сортировки, поэтому вы можете легко упорядочить выбранные узлы в порядке возрастания или убывания. Кроме того, вы можете упорядочить или по символам верхнего регистра, или по символам нижнего регистра. Вы также можете упорядочить узлы по типам данных. В следующем упражнении вы упорядочите две строки таблицы на основе столбца `from` в порядке возрас-

тания по алфавиту. Для того чтобы это сделать, добавьте еще один шаблон к существующей программе XSLT. Этот шаблон будет относиться к элементу `note`. Затем используйте элементы `xsl:apply-templates`, но добавьте атрибуты для выбора элемента `msg`. Это прикажет средству XSL обрабатывать только выбранные дочерние узлы. Например, в данном документе элемент `msg` — это единственный дочерний элемент по отношению к элементу `note`, однако соответствующий атрибут приведен для того, чтобы вы могли понять, каким образом его использовать в других ситуациях.

Элемент `xsl:sort` включен как содержание элемента `xsl:apply-templates`. Для элемента `xsl:sort` указываются два атрибута — `select` и `order`. Атрибут `select` указывает, какой элемент должен быть упорядочен, а атрибут `order` — каким образом. Полный список атрибутов различных элементов вы найдете по адресу `htpp://www.w3.org/TR/xslt`. Полностью новое правило выглядит следующим образом:

```
<xsl:template match="note">
            <xsl:apply-templates select="msg">
                  <xsl:sort select="from" order="ascending"/>
            </xsl:apply-templates>
</xsl:template>
```

Добавьте это правило в документ `msg04.xsl` и сохраните полученный результат под именем `msg05.xsl`. Полный вариант нового документа представлен в листинге 16.7.

Листинг 16.7. Сортировка строк таблиц с помощью XSLT `msg05.xsl`

```
<?xml version="1.0"?>
<!--листинг 16.7 - msg05.xsl -->

<xsl:stylesheet version="1.0"
     xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

    <xsl:template match="/">
       <HTML>
         <HEAD>
            <TITLE>Messages</TITLE>
          </HEAD>
          <BODY>
            <H1>My Messages</H1>
            <TABLE BORDER="1" WIDTH="100%">
                <xsl:apply-templates/>
            </TABLE>
          </BODY>
       </HTML>
    </xsl:template>

    <xsl:template match="msg">
        <TR>
            <xsl:apply-templates/>
        </TR>
    </xsl:template>

    <xsl:template match="note">
            <xsl:apply-templates select="msg">
                  <xsl:sort select="from" order="ascending"/>
            </xsl:apply-templates>
    </xsl:template>

    <xsl:template match="source | from | message">
        <TD>
```

```
35:                 <xsl:apply-templates/>
36:             </TD>
37:         </xsl:template>
38: </xsl:stylesheet>
```

АНАЛИЗ Новое правило шаблона представлено в строках 27–31. Правило применимо к корневому элементу `note` (строка 27). Шаблон применяется только к дочернему элементу `msg`. Дочерние элементы `msg` обрабатываются по алфавиту в порядке возрастания. Остальная часть примера осталась без изменений.

Обратили ли вы внимание на то, что, по мере ввода символов в окне редактора XRay, изменялось содержимое всех четырех окон? Синтаксический анализатор пытается проверять экземпляры документов XML после нажатия любой клавиши. Когда закончите, ваша таблица будет иметь такой вид, как показано на рис. 16.5.

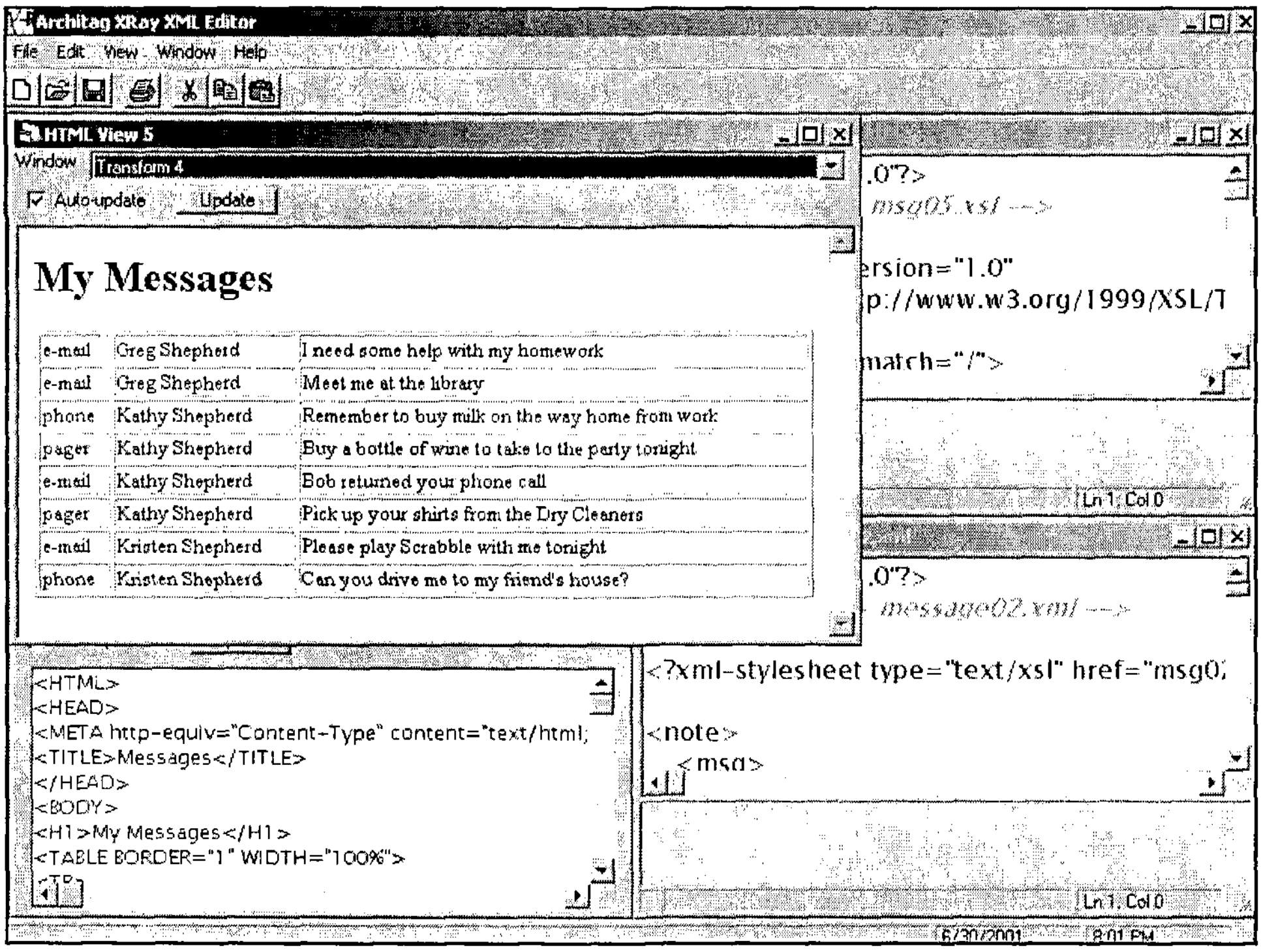


Рис. 16.5. Таблица, отсортированная в порядке возрастания по значению столбца `from`

Добавление атрибутов стилей с помощью XSLT

В следующем примере вы добавите атрибуты цвета, размера и начертания шрифта к дескриптору `HTML` для стилевой разметки сообщений в одном из столбцов таблицы. Этот пример показывает каким образом вы можете использовать элемент `xsl:attribute-set` в XSLT. Этот специальный элемент должен быть непосредственным дочерним элементом по отношению к корневому элементу и может содержать элементы `xsl:attribute` для определения атрибутов. Это предоставляет разработчику широкие возможности по созданию сложных отношений между элементами и атрибутами в результирующей древовидной структуре.

Каждый из элементов `xsl:attribute` содержит обязательный атрибут `name`, который определяет имя создаваемого атрибута. Например, предположим, вам необходимо, чтобы текст сообщения отображался в таблице красным цветом. Вы можете создать соответствующий атрибут, указав его в необходимом месте документа. В нашем примере элемент `xsl:attribute-set` выглядит следующим образом:

```
<xsl:attribute-set name="message-attributes">
      <xsl:attribute name="size">+1</xsl:attribute>
      <xsl:attribute name="color">blue</xsl:attribute>
      <xsl:attribute name="face">Verdana</xsl:attribute>
</xsl:attribute-set>
```

Атрибут `name="message-attributes"` элемента `xsl:attribute-set` представляет собой ссылку, которая вызывается при выполнении следующего действия. Три элемента `xsl:attribute` создают три атрибута с указанными значениями. Эти атрибуты передаются элементу в результирующей древовидной структуре во время процесса преобразований. В этом случае атрибутами являются такие атрибуты HTML, как `FONT`, которые должны быть связаны с дескриптором `FONT` при преобразованиях. Новое правило шаблона создается для того, чтобы связать атрибуты с элементом `FONT`. Поскольку вы применяете стили только к элементам `message` в исходном документе XML, правило выполняется по отношению к каждому элементу `message`. Правило выглядит следующим образом:

```
<xsl:template match="message">
        <TD>
           <FONT xsl:use-attribute-sets="message-attributes">
               <xsl:apply-templates/>
           </FONT>
        </TD>
</xsl:template>
```

Как видно, дескриптор `<FONT xsl:use-attribute-sets="message-attributes">` определяет атрибуты для дескриптора `FONT` в результирующей древовидной структуре. Вам необходимо выполнить еще одно дополнительное действие для того, чтобы это все сработало: удалить элемент `message` из атрибута `match` предыдущего шаблона. Окончательный вариант кода для нашего примера приведен в листинге 16.8.

Листинг 16.8. Применение стилей атрибутов с помощью XSLT `msg06.xsl`

```
1:  <?xml version="1.0"?>
2:  <!--листинг 16.8 - msg06.xsl -->
3:
4:  <xsl:stylesheet version="1.0"
5:       xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
6:
7:      <xsl:template match="/">
8:         <HTML>
9:           <HEAD>
10:             <TITLE>Messages</TITLE>
11:          </HEAD>
12:          <BODY>
13:             <H1>My Messages</H1>
14:             <TABLE BORDER="1" WIDTH="100%">
15:                 <xsl:apply-templates/>
16:             </TABLE>
17:          </BODY>
18:        </HTML>
19:     </xsl:template>
20:
```

```
21:     <xsl:attribute-set name="message-attributes">
22:       <xsl:attribute name="size">+1</xsl:attribute>
23:       <xsl:attribute name="color">blue</xsl:attribute>
24:       <xsl:attribute name="face">Verdana</xsl:attribute>
25:     </xsl:attribute-set>
26:
27:     <xsl:template match="message">
28:         <TD>
29:             <FONT xsl:use-attribute-sets="message-attributes">
30:                 <xsl:apply-templates/>
31:             </FONT>
32:         </TD>
33:     </xsl:template>
34:
35:     <xsl:template match="msg">
36:         <TR>
37:             <xsl:apply-templates/>
38:         </TR>
39:     </xsl:template>
40:
41:     <xsl:template match="note">
42:             <xsl:apply-templates select="msg">
43:                     <xsl:sort select="from" order="ascending"/>
44:             </xsl:apply-templates>
45:     </xsl:template>
46:
47:     <xsl:template match="source | from">
48:         <TD>
49:             <xsl:apply-templates/>
50:         </TD>
51:     </xsl:template>
52: </xsl:stylesheet>
```


My Messages

e-mail	Greg Shepherd	I need some help with my homework
e-mail	Greg Shepherd	Meet me at the library
phone	Kathy Shepherd	Remember to buy milk on the way home from work
pager	Kathy Shepherd	Buy a bottle of wine to take to the party tonight
e-mail	Kathy Shepherd	Bob returned your phone call
pager	Kathy Shepherd	Pick up your shirts from the Dry Cleaners
e-mail	Kristen Shepherd	Please play Scrabble with me tonight
phone	Kristen Shepherd	Can you drive me to my friend's house?

Рис. 16.6. Применение стилей с помощью XSLT

АНАЛИЗ Строки 21–25 содержат набор атрибутов (`xsl:attribute-set`) с именем ссылки `message-attributes`. В строках 22, 23 и 24 определяются атрибуты для размера, цвета и начертания соответственно. Новый шаблон содержится в строках 27–33 и относится к элементу `message` в исходном документе XML. Ссылка `message-attributes` вызывается атрибутом `xsl:attribute-set` дескриптора `FONT`. Элемент `message` удаляется из элемента `xsl:template` в строке 47. Результат преобразований, представленный в окне Internet Explorer 5.0, показан на рис. 16.6.

Профессиональное использование преобразований XSLT

Как и в случае со многими другими языками программирования, XSLT предлагает элементы, используемые для проверки выполнения условий, циклического перебора узлов и переключения между наборами инструкций. Несмотря на то, что это выходит за рамки упражнений настоящего урока, сейчас мы рассмотрим несколько полезных элементов. Все эти элементы представлены в табл. 16.1, а подробные сведения о них вы найдете по адресу `http://www.w3.org/TR/xslt.html`.

Элемент xsl:for-each

Элемент `xsl:for-each` представляет простую форму циклического перебора узлов из заданного набора. Инструкция оказывается полезной для обработки повторяющихся узлов. Синтаксис элемента `xsl:for-each` приведен ниже:

```
<xsl:for-each select="узел">
    шаблон
</xsl:for-each>
```

Значение атрибута `select` может быть любым выражением XPath. Например, вы можете выбрать все элементы `msg` в документе `message01.xml` (см. листинг 16.3) с помощью элемента `xsl:for-each`, который выглядит следующим образом:

```
<xsl:for-each select="msg">
```

Предположим, вам необходимо разместить содержание элементов `source`, `from` и `message` из каждого родительского элемента `msg` в ячейки HTML-таблицы. Вы можете создать шаблон с помощью элемента `xsl:for-each`, который выглядит следующим образом:

```
<xsl:for-each select="msg">
    <TR>
        <TD>xsl:value-of select="source"/></TD>
        <TD>xsl:value-of select="from"/></TD>
        <TD>xsl:value-of select="message"/></TD>
    </TR>
</xsl:for-each>
```

Элемент `xsl:for-each` позволяет связать узлы исходного документа с узлами результирующего документа. Например, приведенный выше фрагмент кода для каждого из элементов `msg` в документе `message02.xml` создает строку таблицы, содержащую данные элементов `source`, `from` и `message`.

Элемент xsl:if

Элемент `xsl:if` в XSLT позволяет выполнять инструкцию только в том случае, если проверка условия приведет к логическому значению `true`. В отличие от инст-

рукции if в большинстве языков программирования, инструкция xsl:if не содержит оператор else. Если проверка условия приведет к логическому значению true, инструкция выполняется. Синтаксис элемента xsl:if приведен ниже:

```
<xsl:if test=выражение>
    шаблон
</xsl:if>
```

Например, если вам необходимо выбрать только те элементы msg из документа message02, дочерние элементы source которых содержат слово e-mail, ваше выражение может выглядеть следующим образом:

```
<xsl:for-each select="msg">
    <xsl:if test="source"='e-mail'">
        <xsl:value-of select="message"/><br/>
    </xsl:if>
</xsl:for-each>
```

Если вы добавите этот фрагмент кода в шаблон для элемента note в своем документе, будет возвращено содержание всех элементов message, дочерние элементы source содержат слово e-mail:

```
I need some help with my homework
Please play Scrabble with me tonight
Meet me at the library
Bob returned your phone call
```

Элементы xsl:choose, xsl:when и xsl:otherwise

Элемент xsl:choose в XSLT определяет условный шаблон, в котором выбирается одно из возможных действий. Элемент xsl:choose обеспечивает функциональные возможности, аналогичные инструкциям switch, if-then-else или select в других языках программирования. В элементе xsl:choose нет никаких атрибутов. Это элемент-контейнер, который содержит один или несколько дочерних элементов xsl:when, а также дочерний элемент xsl:otherwise. Элементов xsl:when может быть сколько угодно, но как минимум один из них в элементе xsl:choose должен быть. Элемент xsl:otherwise, в свою очередь, необязателен, но при наличии в элементе xsl:choose он должен указываться последним:

```
<xsl:choose>

    <xsl:when test=выражение>
        шаблон
    </xsl:when>

    <xsl:otherwise>
        шаблон
    </xsl:otherwise>

</xsl:choose>
```

Элемент xsl:when разрешает выполнение инструкции в том случае, если проверка одного из условий возвратит логическое значение true точно так же, как и в случае использования элемента xsl:if. Атрибут test может принимать значение, равное любому из выражений XPath. Поскольку элемент xsl:otherwise задействуется в том случае, если не выполняется условие ни одного из элементов xsl:when, проверка логического выражения не нужна.

Например, если вам необходимо обработать содержание каждого из элементов message тем или иным образом, в зависимости от содержания их элемента source, вы можете использовать элемент xsl:choose для проверки каждого из возможных значений элемента source. Возможно, вам придется использовать метки, которые определяют источник каждого из сообщений. Элементам msg, дочерние элементы source которых содержат слово e-mail, вы можете присвоить заголовок E-mail Message: и т. д. Код соответствующего элемента xsl:choose приведен ниже:

```
<xsl:choose>

    <xsl:when test="source='e-mail'">
        <P>E-Mail Message:
            <xsl:value of select="message"/><br/>
        </P>
    </xsl:when>

    <xsl:when test="source='phone'">
        <P>E-Phone Message:
            <xsl:value of select="message"/><br/>
        </P>
    </xsl:when>

    <xsl:otherwise>
        <P>Pager Message:
            <xsl:value of select="message"/><br/>
        </P>
    </xsl:otherwise>

</xsl:choose>
```

В случае документа message02.xml вы можете использовать выражения XPath для проверки двух из трех значений элемента source. Третье значение, которое остается после проверки двух других, можно выбрать с помощью элемента xsl:otherwise. Если вы добавите этот фрагмент кода в шаблон, который выбирает каждый дочерний элемент msg элемента note в вашем документе, вы получите следующий результат:

```
Phone Message: Remember to buy milk on the way home from work
E-mail Message: I need some help with my homework
E-mail Message: Please play Scrabble with me tonight
Pager Message: Buy a bottle of wine to take to the party tonight
Phone Message: Can you drive me to my friend's house?
E-mail Message: Meet me at the library
E-mail Message: Bob returned your phone call
Pager Message: Pick up your shirts from the Dry Cleaners
```

Поскольку таблица стилей XSL — это экземпляр документа XML, который следует общим синтаксическим правилам XML, вы можете при необходимости использовать вложенные элементы. Например, если вы хотите отсортировать полученный результат по различным элементам source, добавьте в документ XSLT элемент xsl:sort, который выглядит следующим образом:

```
<xsl:sort select="source" order="ascending"/>
```

Однако элемент xsl:sort не допускает дочерних элементов, отличных от xsl:when и xsl:otherwise; таким образом, вам необходимо вынести сортировку за пределы элемента xsl:choose.

Резюме

Сегодня вы использовали язык преобразований XSLT (XSL Transformation Language) для превращения XML в HTML. Вы увидели, что XSLT может функционировать независимо от XSL. Для выполнения подобных операций вам необходим обработчик XSLT. Вы использовали XT, бесплатный обработчик XSLT, написанный на Java, который работает из командной строки, а также обработчик, встроенный в XML-редактор XRay. XSLT использует концепцию совпадающих шаблонов для выполнения определенных правил. Для обнаружения узлов используются выражения XPath. XSLT — это мощная технология, которую лучше всего охарактеризовать как основанный на событиях и правилах язык программирования.

Вопросы и ответы

Какая связь между XPath и XSLT?

XPath используется в шаблонах XSLT для обнаружения элементов, которые должны быть обработаны с помощью правил шаблона. Изначально XPath —это компонентная технология. После этого XPath была отделена от XSLT для использования с другими технологиями XML.

Какими платформами и операционными системами ограничено использование XSLT?

Язык XSLT способен функционировать на различных платформах. Все, что необходимо, — это SAX-совместимый синтаксический анализатор, а также библиотеки вызовов процедур. Различные версии средств XSLT написаны на различных языках программирования для всех популярных платформ.

Каким образом такое средство XSLT, как Xalan, используется для преобразования XML в HTML?

Xalan — это обработчик XSLT, разработанный группой *Apache XML Project* для преобразования документов XML к HTML, текстовому формату и другим типам документов XML. Например, вы можете использовать Xalan вместо XT. Сначала вам необходимо решить, какую именно версию Xalan вы будете использовать. На момент написания настоящей книги существовали две версии Xalan — 1.2.2 и 2.1.0, обе написанные на Java. Вы всегда можете загрузить свежие версии Xalan с Web-узла `http://xml.apache.org/`. Подробные сведения об использовании той или иной версии программного обеспечения вы найдете в сопровождающей его документации. Как и в случае большинства Java-приложений, вам необходимо задать переменные окружения, а также определить путь `classpath` для всех приложений и вспомогательных средств. В данном случае вам необходимо включить файлы `Xerces.jar` для синтаксического анализатора Xerces или другие файлы, соответствующие выбранному вами синтаксическому анализатору. Вы можете вызвать Xalan из командной строки, воспользовавшись следующей командой:

```
java org.apache.xalan.xslt.Process -IN myfile.xml -XSL myfile.xsl -OUT
myfile.out
```

Однако вам необходима документация, чтобы ознакомиться с точным синтаксисом, соответствующим используемой вами версии программы, поскольку структура инструкции, которую следует указать в командной строке, значительно отличается от версии к версии. На момент написания настоящей книги в командной строке допускалось использование 22 параметров и аргументов, контролирующих разрывы линий, форматирование HTML и т.д. Подробные сведения вы найдете в соответствующей документации.

Упражнение

Проверьте свои знания, полученные в результате изучения настоящего урока. Ответ вы найдете в Приложении А, "Ответы на вопросы".

В XSLT существует специальный элемент `xsl:number`, который автоматически нумерует совпадающие элементы. Напишите код XSLT, использующий элемент `xsl:number` для индексации названий компакт-дисков в документе `CD.xml`. Если вам это не удастся, обратитесь к примерам использования элемента `xsl:number` на Web-узле консорциума *W3C*.

День 17-й

Связывание данных XML с элементами HTML

Используя приемы для связывания данных, вы сможете объединить данные XML с HTML-страницами для более гибкого представления данных. Вам доступно очень много различных подходов для решения подобной задачи, в частности создание областей данных с помощью элементов HTML, связанных с элементами XML.

Сегодня рассматриваются следующие темы.

- Создание простых областей данных.
- Связывание элементов HTML с определенными элементами XML.
- Контроль связывания данных с помощью JavaScript.
- Вызов XML Data Source Object в качестве Java-аплета для передачи данных XML HTML-странице.

Создание простых экземпляров связывания данных

Связывание данных бывает различным. В некоторых случаях речь идет о связывании сложной модели данных с собственной моделью данных компьютерной программы. Таким образом, например, можно связать данные XML с объектами, созданными с помощью Java или других языков программирования. Некоторые формы связывания данных позволяют преобразовать общие структуры к более специфичным структурам для последующей обработки приложениями, работающими с сильно структурированными данными. В самой простой форме, применимой к XML, связывание данных подразумевает извлечение текстовых данных из элементов с помощью программы для их сопоставления с элементами других языков разметки. Сегодня вы познакомитесь с созданием связей между данными XML и данными дескрипторов HTML и сможете оценить интеллектуальный подход XML к хранению данных, а также возможности HTML по их отображению.

Связывая простые данные, вы сможете обработать документы так, чтобы отобразить данные XML на обычной HTML-странице. Несмотря на то, что — это некая разновидность преобразования данных, вы поймете, что этот путь более ограничен, чем язык XSLT с его широкими возможностями, с которым вы познакомились при изучении урока дня 16-го. Вы также обнаружите, что многие из приемов, рассмотренных в настоящем уроке, работают только в броузерах Microsoft Internet Explorer вер-

сии 4.0 и выше. Создание простых областей данных, особенно при использовании сценариев на стороне клиента, позволяет эффективно и быстро создавать специализированные инструменты для работы с данными XML, устранения неполадок и быстрого анализа симметричных наборов данных. Освоив эти приемы, вы сэкономите свое время, когда дойдет дело до управления сложными коллекциями данных XML.

В этом уроке вы создадите набор данных XML, выступающий в роли источника данных. Вы свяжете этот набор с HTML-страницей, после чего свяжете элементы HTML с полями в наборе данных, например элементов в документе XML. Вы также познакомитесь с использованием элементов DSO (Data Source Object), к которым относятся непосредственная загрузка DSO и Java-аплеты XMLDSO. После этого вы научитесь комбинировать связывание данных и использование сценариев для контроля над потоком данных на HTML-странице. Для всего описанного необходимы средства *Microsoft*; у других поставщиков их нет. Учитывая, что эти инструменты распространяются совершенно бесплатно и доступны для различных платформ, они открывают замечательную перспективу исследовать связывание данных.

Структура документа XML

Для того чтобы продолжить работу со связыванием простых данных, создайте новый или измените один из документов XML, с которыми вы работали в предыдущих главах. Используйте документ с симметричной структурой для контроля над потоком данных на HTML-странице. Симметрия в структуре документа — это характеристика неструктурированной базы данных, когда записи содержат строго определенные поля в неизменном порядке. В системе обмена сообщениями, рассматриваемой нами в течение нескольких дней, каждый элемент `msg` представляет запись. Корневой элемент `note` содержит любое количество подобных записей в качестве дочерних элементов. Каждый из элементов `msg` содержит дочерние элементы `source`, `from` и `message`, выступающие в роли полей записи, которые, в свою очередь, содержат текстовые данные. Структура рассматриваемого нами документа отражена на рис. 17.1.

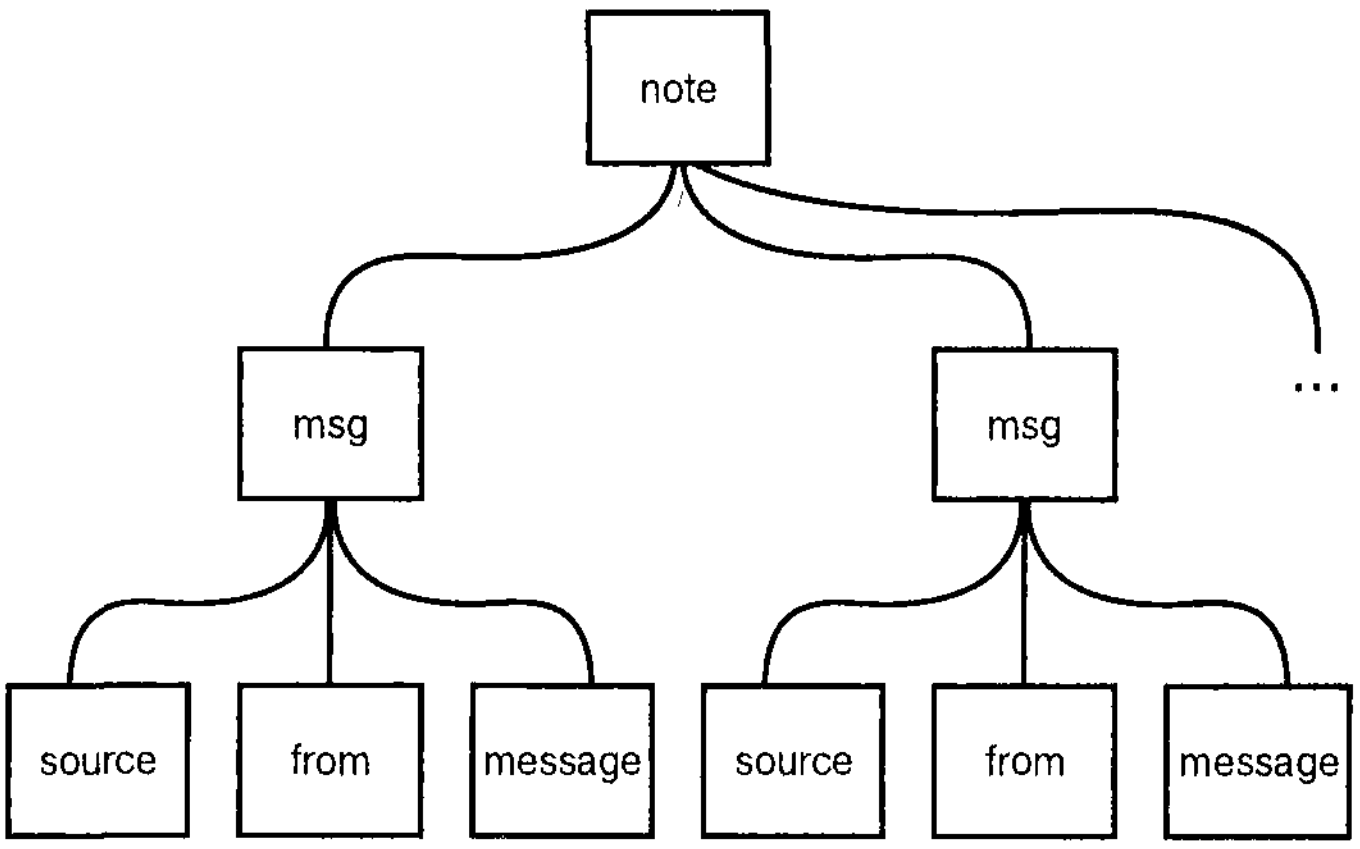


Рис. 17.1. Структура документа XML

Корневой элемент может содержать любое количество элементов `msg`, структура каждого из них повторяется. Используя эту структуру и убедившись в том, что все типы элементов обозначены подобным образом, создайте экземпляр документа XML и сохраните его под именем `message01.xml`. Если вы выполните все предложенные упражнения, вы сможете использовать этот экземпляр документа несколько раз. Вам

очень полезно представить в наборе данных несколько записей; для этого или создайте новые записи, или скопируйте существующие, чтобы получить не меньше шести дочерних элементов msg, каждый из которых содержит дочерние элементы source, from и message. Пример необходимого набора данных XML приведен в листинге 17.1. Ваши элементы могут содержать разные текстовые данные, но обязательно убедитесь в том, что все они названы одинаково, в частности в их именах используются знаки одного и того же регистра.

Листинг 17.1. Иерархический набор данных XML `message01.xml`

```
1: <?xml version="1.0"?>
2: <!-- листинг 17.1 - message01.xml -->
3:
4: <note>
5:     <msg>
6:       <source>phone</source>
7:       <from>Kathy Shepherd</from>
8:       <message>Remember to buy milk on the way home from
         work</message>
9:     </msg>
10:
11:    <msg>
12:      <source>e-mail</source>
13:      <from>Greg Shepherd</from>
14:      <message>I need some help with my homework</message>
15:    </msg>
16:
17:    <msg>
18:      <source>e-mail</source>
19:      <from>Kristen Shepherd</from>
20:      <message>Please play Scrabble with me tonight</message>
21:    </msg>
22:
23:    <msg>
24:      <source>pager</source>
25:      <from>Kathy Shepherd</from>
26:      <message>Buy a bottle of wine to take to the party
         tonight</message>
27:    </msg>
28:
29:    <msg>
30:      <source>phone</source>
31:      <from>Kristen Shepherd</from>
32:      <message>Can you drive me to my friend's house?</message>
33:    </msg>
34:
35:    <msg>
36:      <source>e-mail</source>
37:      <from>Greg Shepherd</from>
38:      <message>Meet me at the library</message>
39:    </msg>
40:
41:    <msg>
42:      <source>e-mail</source>
43:      <from>Kathy Shepherd</from>
44:      <message>Bob returned your phone call</message>
45:    </msg>
46:
```

```
47:      <msg>
48:        <source>pager</source>
49:        <from>Kathy Shepherd</from>
50:        <message>Pick up your shirts from the Dry Cleaners</message>
51:      </msg>
52:
53: </note>
```

Анализ Дочерние элементы `msg` корневого элемента `note` содержат только другие элементы. Каждый из этих элементов содержит дочерние элементы `source`, `from` и `message`. Элементы `source`, `from` и `message` включают в себя только текстовые данные.

Как видите, в документе `message01.xml` элементы `msg` напоминают записи или строки в обычной базе данных, а элементы `source`, `from` и `message` — их поля или столбцы. В следующем разделе вы сможете связать этот документ с HTML-страницей, после чего связать стандартные элементы HTML с полями в наборе данных. Потом вам потребуется настроить ссылки.

Связывание документа XML с HTML-страницей

Связывание документа XML с HTML-страницей требует установления ссылки. Для этого вы воспользуйтесь специальным элементом HTML — `<XML>`, который поддерживается броузером Microsoft Internet Explorer версии 4.0 и выше и представляет собой именно элемент HTML, а не элемент XML. Элемент `<XML>` может содержать несколько атрибутов, из которых для связывания данных вы будете использовать только два. Атрибут `src` (`source`) представляет действительный идентификатор URI для обнаружения связываемого источника данных XML. Атрибут `id` представляет именованную ссылку, которая используется на HTML-странице для привлечения данных XML. Синтаксис элемента `<XML>` приведен ниже:

```
<XML src="[URI]" id="[name]"/>
```

Что интересно, так это то, что некоторые версии броузера Microsoft Internet Explorer не поддерживают элемент `<XML>` в приведенной выше форме. Ранние версии Internet Explorer не обладают полной поддержкой стандартов HTML 4.01 или XHTML 1.0 и могут не воспринять краткую форму представления этого пустого элемента. Если вы убедитесь в том, что не в состоянии выполнить некоторые из сегодняшних упражнений, вполне возможно, что вам пора обновить версию броузера. Это справедливо даже по отношению к первым редакциям Microsoft Internet Explorer версии 5.0 для некоторых платформ. Компания *Microsoft* рекомендует регулярно обновлять версию Internet Explorer, чтобы обеспечить совместимость с развивающимися технологиями программирования, подобными тем, что рассматриваются в настоящем уроке. Кроме того, как и в случае любого другого программного обеспечения, ранние редакции могут содержать ошибки, которые исправляются в более поздних версиях.

Использование дескриптора `<XML>` приводит к созданию ссылки, которая называется островом данных. *Остров данных* — это документ XML, внедренный в HTML-страницу. Причем он внедряется как "виртуально", благодаря дескриптору `<XML>` с атрибутом `src`, представляющим идентификатор URI, который указывает на внешний документ XML, так и явно, когда его код просто добавляется к коду HTML-страницы. В последнем, менее практичном случае, документ XML заключается между дескрипторами `<XML>`.



Несмотря на то, что код документа XML просто добавить в код HTML-страницы, такой подход оказывается непрактичным и противоречит основным правилам XML-программирования, согласно которым разделение содержания и стилей считается ключевым условием. Во всех рассматриваемых в настоящем уроке упражнениях вы будете использовать внешние источник данных XML.

Например, внешняя ссылка на документ `message01.xml`, который вы создали раньше, выглядит следующим образом:

```
<XML id="myMsg" src="message01.xml"/>
```

Значение атрибута `id` (равное `myMsg`) представляет уникальное имя, назначенное вами; в документе HTML оно используется со знаком "решетки" (#), принятым в HTML. Другими словами, ссылка на этот остров данных указывается в виде `#myMsg`.

При необходимости можно задать значение атрибута `src`, равное полностью определенному идентификатору URI. Таким образом, источник данных можно расположить в любой точке Internet или, как в нашем случае, локально в домене или на сервере.

Размещение ссылки XML на HTML-странице

Теперь, когда вы познакомились с синтаксисом элемента для связывания острова данных XML, создайте HTML-страницу, на которой будет использоваться дескриптор <XML>. При выполнении настоящих упражнений предполагается, что вы уже знакомы с созданием HTML-страниц. Вам потребуются такие стандартные элементы HTML, как `HTML`, `HEAD`, `BODY` и т. д., а также дескриптор <XML> вместе с соответствующими атрибутами. Пример HTML-страницы, содержащей остров данных, приведен в листинге 17.2. Создайте подобную страницу и сохраните ее под именем `island01.xml`.

Листинг 17.2. HTML-страница, содержащая XML Data Island, island01.html

```
1:  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
2:   "http://www.w3.org/TR/html4/loose.dtd">
3:  <!-- Листинг 17.2 island01.html -->
4:
5:  <HTML>
6:  <HEAD>
7:  <TITLE>XML Data Island</TITLE>
8:  </HEAD>
9:
10: <BODY>
11: <XML id="myMsg" src="message01.xml"/>
12:
13: </BODY>
14: </HTML>
```



В строках 1 и 2 представлено стандартное объявление `DOCTYPE` документа HTML версии 4.01, с которым связана схема DTD. В строке 11 содержится элемент <XML> (<XML id="myMsg" src="message01.xml"/>), который связывает с документом HTML исходный документ XML, указанный с помощью ссылки `myMsg`.

Создав документ `island01.html`, загрузите его в Internet Explorer и просмотрите полученные результаты. Вам не кажется, что на этом этапе больше не нужны никакие действия?

Вы создали остров данных, который связывает документ `message01.xml` с HTML-страницей `island01.html`, но вы еще не связали данные из документа XML с элементами HTML, предназначенными для отображения данных в окне броузера. Следовательно, в окне броузера не будет никаких сведений. Вам необходимо использовать данные HTML для того, чтобы представить данные XML в окне броузера.

Связывание элементов HTML с элементами XML

Для представления данных XML в окне броузера вам необходимо связать элементы HTML с определенными элементами из набора данных XML. Возможность связывания поддерживается следующими элементами HTML.

- `A`
- `APPLET`
- `BUTTON`
- `DIV`
- `FRAME`
- `IMG`
- `INPUT`
- `LABEL`
- `MARQUE`
- `SELECT`
- `SPAN`
- `TEXTAREA`

Вы будете использовать элементы `DIV` и `SPAN` как контейнеры для данных XML, которые размещаются среди других элементов HTML для контроля над отображением данных в окне броузера. Когда вы связываете элемент HTML с элементом XML, броузер отображает текстовые данные элемента XML. Синтаксис элементов `DIV` и `SPAN` приведен ниже:

```
<DIV datasrc="#[ссылка]" datafld="[тип элемента XML]"></DIV>
```

или

```
<SPAN datasrc="#[ссылка]" datafld="[тип элемента XML]"></SPAN>
```

Атрибут `datasrc` указывает на источник данных, которые должны связываться с элементом HTML. Атрибут `datafld` задает конкретное поле, которое должно связываться. Например, для связывания элемента `SPAN` с элементом `from` из документа `message01.xml` можно использовать следующую строку:

```
<SPAN datasrc="#[ссылка]" datafld="[тип элемента XML]"></SPAN>
```

Элементы `DIV` и `SPAN` — это элементы-контейнеры, которые могут содержать данные, стили или другие элементы HTML, предназначенные для использования на HTML-странице, они отличаются между собой. Элемент `DIV` обеспечивает отображение данных отдельно от других элементов HTML-страницы, добавляя разрывы до и после них. `SPAN` — это "внутренний" элемент-контейнер, который часто используется для добавления стиля к существующему элементу. Элемент `SPAN` не приводит к появлению разрывов строк до и после его содержания, а внедряет его между другими элементами HTML-страницы. При добавлении данных из документа XML к существующей структуре HTML вы будете использовать преимущественно элемент SPAN.

Создание HTML-таблиц, содержащих данные XML

Предположим, вам необходимо использовать данные из документа `message01.xml` для представления в виде таблицы на HTML-странице. Вы можете использовать элемент `SPAN` в элементах ячеек таблицы `TD` (Table Data). Вам необходимо сделать это, поскольку элемент `TD` не поддерживает связывание элементов HTML. Несмотря ни на что, создание таблицы, содержащей связанные элементы, с использованием элементов `SPAN` — один из самых простых способов представления данных XML на HTML-странице, поскольку такой элемент HTML, как `TABLE`, поддерживает рекурсию при динамическом представлении данных. Другими словами, вы можете создать полную таблицу данных XML на HTML-странице, в которой каждая строка будет соответствовать каждому элементу из набора данных, написав всего одну строку кода для HTML-таблицы. Таблица будет динамически добавлять новые строки для представления всех данных, получаемых при обработке источника данных. Синтаксис элемента таблицы HTML приведен ниже:

```
<TABLE datasrc="#[ссылка]">
    <TR>
        <TD><SPAN datafld="имя элемента"/></SPAN></TD>
                дополнительные инструкции для каждого поля
    </TR>
</TABLE>
```

Если элемент `SPAN` содержится в рамках другого элемента HTML, используйте краткую форму представления пустого элемента:

```
<TD><SPAN datafld="имя элемента"/></SPAN></TD>
```

Однако, в связи с особенностями различных версий броузеров, вам все-таки следует отдавать предпочтение полной форме представления элемента.

В случае документа `message01.xml` элемент `TABLE` может выглядеть так:

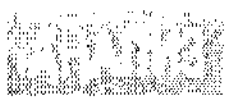
```
<TABLE datasrc="#myMsg">
    <TR>
        <TD><SPAN datafld="source"/></TD>
        <TD><SPAN datafld="from"/></TD>
        <TD><SPAN datafld="message"/></TD>
    </TR>
</TABLE>
```

Добавьте несколько типичных атрибутов таблицы, чтобы указать название таблицы и заголовков ее столбцов. Один из возможных вариантов приведен в листинге 17.3. Создайте подобный документ и сохраните его под именем `island02.xml`.

Листинг 17.3. Ячейки таблицы HTML , связанные с данными XML Data , island02.xml

```
1:  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
2:   "http://www.w3.org/TR/html4/loose.dtd">
3:  <!-- листинг 17.3 island02.html -->
4:
5:  <HTML>
6:  <HEAD>
7:  <TITLE>XML Data Island</TITLE>
8:  </HEAD>
9:
10:  <BODY>
11:  <XML id="myMsg" src="message01.xml"/>
12:  <H1>My Messages</H1>
13:    <TABLE id="table" border="6" width="100%"
14:           datasrc="#myMsg" summary="messages">
```

```
15:          <THEAD style="background-color: aqua">
16:                <TH>Source</TH>
17:                <TH>From</TH>
18:                <TH>Message</TH>
19:          </THEAD>
20:          <TR valign="top" align="center">
21:               <TD><SPAN datafld="source"/></TD>
22:               <TD><SPAN datafld="from"/></TD>
23:               <TD><SPAN datafld="message"/></TD>
24:          </TR>
25:    </TABLE>
26: </BODY>
27: </HTML>
```



В строке 11 определяется остров данных с помощью такого дескриптора HTML, как XML. Атрибут `id="myMsg"` содержит ссылку (`myMsg`), которая используется в строке 14 для указания источника данных для таблицы. Документ XML указывается с помощью значения атрибута `src`. Открывающий дескриптор таблицы (строки 13 и 14) содержит атрибут `datasrc` с именованной ссылкой (`#myMsg`) на остров данных. Открывающий дескриптор таблицы также содержит и другие атрибуты, используемые для идентификации — `id` и `summury` — или форматирования — `border` и `width` — элементов результирующей таблицы. Раздел заголовка таблицы (строки 15 и 16) добавляет для обеспечения заголовков всех столбцов таблицы. Строки 20–24 заняты содержанием элемента `TR`. Строки 21–23 связаны с элементами `source`, `from` и `message` из исходного документа XML, для чего используются элементы `SPAN`.

Загрузив этот документ в броузер Microsoft Internet Explorer версии 5.0 или выше, вы должны увидеть результат, подобный тому, что показан на рис. 17.2.

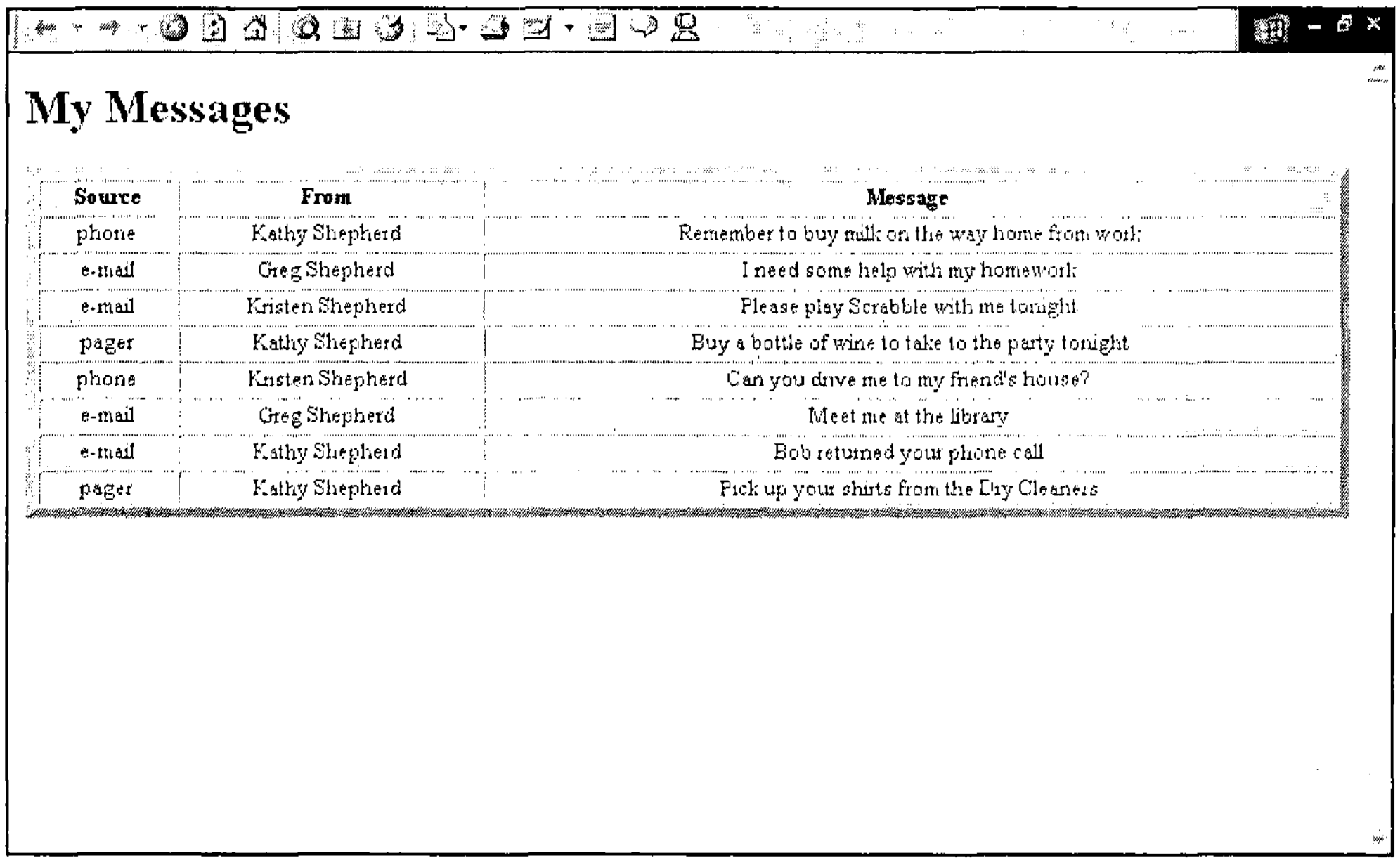


Рис. 17.2. Данные XML представлены в виде HTML-таблицы

Как видно из этого примера, на HTML-странице задается всего одна строка данных, а новые строки таблицы создаются динамически до тех пор, пока в таблице не поместятся все данные XML. В результате мы получаем простой, эффективный и полезный прием для загрузки данных XML на HTML-странице.

Добавление элементов JavaScript для управления протоком данных

Если вы используете эти приемы для отображения больших наборов данных XML, вам понадобится контроль за потоком данных с помощью "моментальных снимков" страницы. Для демонстрации подобного способа контроля присвойте атрибуту `datapagesize` элемента `TABLE` значение `3`. Вы можете использовать сколько угодно строк таблицы, отображаемых на экране одновременно. В нашем документе `island02.xml` новый атрибут в открывающем дескрипторе таблицы может выглядеть следующим образом:

```
<TABLE id="table" border="6" width="100%"
        datasrc="#myMsg" summary="messages"
        datapagesize="3">
```

JavaScript предоставляет возможность перемещения по многостраничным документам с помощью методов некоторых объектов. Для перемещения по записям в нашем примере можно вызвать стандартные методы JavaScript, такие как `firstPage()`, `lastPage()`, `nextPage()` и `previousPage()` объекта DSO. Вы просто добавляете соответствующие методы к значениям атрибута `id` открывающего дескриптора `TABLE` для создания полностью определенного вызова метода. Например, для отображения первой страницы в источнике данных `#myMsg` воспользуйтесь методом `firstPage()`, как показано ниже:

```
table.firstPage()
```

Значение атрибута `id` открывающего дескриптора `TABLE` равно `table`. Таким образом, полностью определенный вызов метода для загрузки первой страницы выглядит следующим образом: `table.firstPage()`. Кроме того, вы можете использовать и другие методы JavaScript для перемещения по страницам, такие как `lastPage()`, `nextPage()` и `previousPage()`.

Очень просто назначить вызов метода атрибуту `onClick` элемента HTML `BUTTON`. Например, вы можете использовать следующие строки для создания четырех кнопок:

```
<BUTTON onClick="table.firstPage()">&lt;&lt;
</BUTTON>

<BUTTON onClick="table.previousPage()">&lt;
</BUTTON>

<BUTTON onClick="table.nextPage()">&gt;
</BUTTON>

<BUTTON onClick="table.lastPage()">&gt;&gt;
</BUTTON>
```

Измените свою HTML-страницу, добавив атрибут `datapagesize` элемента `TABLE`, а также вызовы `BUTTON`. Сохраните измененный документ под именем `island03.xml`. Полный вариант HTML-страницы, в которую уже внесены все необходимые изменения, приведен в листинге 17.4.

Листинг 17.4. Элементы управления страницами JavaScript, добавленные к таблице HTML и связанные с данными XML Data, island03.xml

```
1:  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
2:   "http://www.w3.org/TR/html4/loose.dtd">
3:  <!-- листинг 17.4 island03.html -->
4:
5:  <HTML>
6:  <HEAD>
7:  <TITLE>XML Data Island</TITLE>
8:  </HEAD>
9:
10:  <BODY>
11:  <XML id="myMsg" src="message01.xml"/>
12:  <H1>My Messages</H1>
13:    <TABLE id="table" border="6" width="100%"
14:           datasrc="#myMsg" summary="messages"
15:                datapagesize="3">
16:         <THEAD style="background-color: aqua">
17:                <TH>Source</TH>
18:                <TH>From</TH>
19:                <TH>Message</TH>
20:         </THEAD>
21:         <TR valign="top" align="center">
22:              <TD><SPAN datafld="source"/></TD>
23:              <TD><SPAN datafld="from"/></TD>
24:              <TD><SPAN datafld="message"/></TD>
25:         </TR>
26:    </TABLE>
27:  <BR/>
28:  <BUTTON onClick="table.firstPage()">&lt;&lt;
29:  </BUTTON>
30:
31:  <BUTTON onClick="table.previousPage()">&lt;
32:  </BUTTON>
33:
34:  <BUTTON onClick="table.nextPage()">&gt;
35:  </BUTTON>
36:
37:  <BUTTON onClick="table.lastPage()">&gt;&gt;
38:  </BUTTON>
39:
40:  </BODY>
41:  </HTML>
```

Открывающий дескриптор элемента TABLE (строки 13–15) содержит атрибут datapagesize со значением, равным 3. Кнопки для вызова методов перемещения по страницам заданы в строках 28–38. Кнопка, заданная в строках 28 и 29, загружает первую страницу данных после активизации с помощью щелчка мышью. Текст кнопки состоит из двух знаков "меньше" (<<), заданных должным образом с помощью ссылок (&lt, &lt), что позволяет избежать проблем, связанных с обработкой знаков разметки. Кнопка, заданная в строках 31 и 32, загружает предыдущую страницу данных после активизации с помощью щелчка мышью. Если в таблице отображается самая первая страница данных, то щелчок на этой кнопке просто игнорируется. Кнопка, заданная в строках 34 и 35, загружает следующую страницу данных после активизации

с помощью щелчка мышью. Если в таблице отображается самая последняя страница данных, то щелчок на этой кнопке просто игнорируется. Кнопка, заданная в строках 37 и 38, загружает последнюю страницу данных после активизации с помощью щелчка мышью.

Создав и сохранив HTML-страницу, загрузите ее в броузер Internet Explorer. Вы должны получить результат, подобный тому, что показан на рис. 17.3.

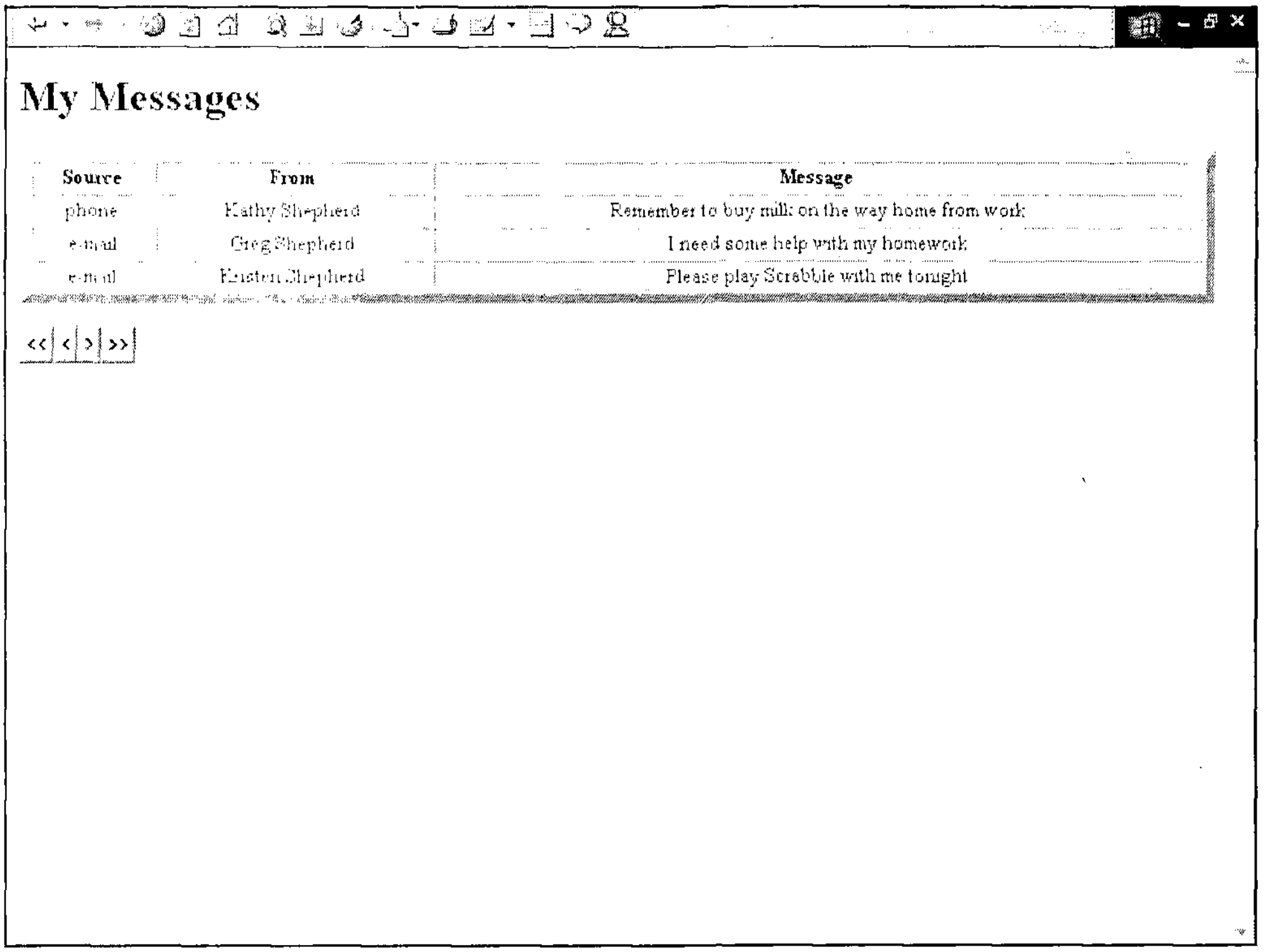


Рис. 17.3. Данные XML, представленные в HTML-таблице с помощью элементов управления переходами между страницами

Просмотр данных одной записи

Предположим, вам необходимо представить данные из набора данных XML по одной записи. Вы можете сгенерировать список полей, не привязывая его к таблице, после чего использовать элементы управления JavaScript для перемещения между данными. Данные XML представляют собой несколько записей, как вы уже знаете. Доступ к отдельным записям возможен с помощью объекта из набора записей `recordset` DSO. DSO представляет собой часть стандартной технологии доступа, которую компания *Microsoft* называет ADO (ActiveX Data Objects — объекты данных ActiveX). Подобные технологии существуют для многих форм данных, включая базы данных, простой текст и XML, но вы будете использовать методы DSO, относящиеся к технологии XMLDSO (XML Data Source Object).

Начните с определения списка полей, используя простые дескрипторы `SPAN` в разметке таблицы, созданной в предыдущем упражнении. Это может выглядеть следующим образом:

```
<BODY>
<XML id="myMsg" src="message01.xml"/>
<H1>My Messages</H1>
    <SPAN datasrc="#myMsg" datafld="source"></SPAN><BR/>
    <SPAN datasrc="#myMsg" datafld="from"></SPAN/><BR/>
    <SPAN datasrc="#myMsg" datafld="message"></SPAN/><BR/>
</BODY>
```

Затем добавьте кнопки HTML, необходимые для контроля за методами набора записей DSO. Все методы, которые вы можете использовать, перечислены в табл. 17.1.

Таблица 17.1. Методы набора записей DSO

Метод	*Получает доступ к*	*Синтаксис вызова*
`move`	Определенной записи в наборе данных, указанной по ее значению (#)	`myMsg.recordset.move([#])`
`moveFirst`	Первой записи в наборе данных	`myMsg.recordset.moveFirst()`
`moveLast`	Последней записи в наборе данных	`myMsg.recordset.moveLast()`
`moveNext`	Следующей записи в наборе данных	`myMsg.recordset.moveNext()`
`movePrevious`	Предыдущей записи в наборе данных	`myMsg.recordset.movePrevious()`

Например, для перехода к последней записи в наборе данных вы можете написать следующий сценарий, выполняемый после щелчка на кнопке:

```
<BUTTON onClick="myMsg.recordset.movelast()">
      &gt;&gt;
</BUTTON>
```

Эта кнопка, которая активизируется щелчком мышью, вызовет метод набора записей DSO для изменения текущей записи на последнюю запись в наборе данных, указанных с помощью ссылки `myMsg`. Методы JavaScript, с которыми вы имели дело раньше, были защищены от достижения начала или конца файла благодаря тому, что игнорировали вызовы записей, выходящих за рамки действия документа. Методы DSO не предоставляют подобной защиты, поэтому вам необходимо написать сценарий для обработки ошибок, которые могут возникнуть в том случае, если вы попытаетесь перейти к последней записи при уже выбранной последней записи или к первой записи при уже выбранной первой записи в наборе данных. Объекты наборов записей обладают свойствами `BOF` (Beginning Of File — метка начала файла) и `EOF` (End Of File — метка конца файла), которые пригодятся при решении подобных проблем. Вы можете поэкспериментировать с использованием свойств `BOF` и `EOF`, применив инструкцию `if` в сценарии кнопки. Например, кнопка `moveprevious` проверяет наличие в текущей записи из набора записей метки начала файла. Тогда сценарий будет вызывать метод `moveNext()`, а не метод `movePrevious()`. Сценарий выглядит следующим образом:

```
<BUTTON onClick="myMsg.recordset.moveprevious();
                 if (myMsg.recordset.BOF)
                     myMsg.recordset.movenext()">
```

Используя эти приемы, вы можете получить следующий фрагмент кода:

```
<BUTTON onClick="myMsg.recordset.movefirst()">
      &lt;&lt;
</BUTTON>
<BUTTON onClick="myMsg.recordset.moveprevious();
                 if (myMsg.recordset.BOF)
                     myMsg.recordset.movenext()">
      &lt;
</BUTTON>
```

```
<BUTTON onClick="myMsg.recordset.movenext();
                    if (myMsg.recordset.EOF)
                        myMsg.recordset.moveprevious()">
      &gt;
</BUTTON>
<BUTTON onClick="myMsg.recordset.movelast()">
      &gt;&gt;
</BUTTON>
```

Объедините это все и создайте HTML-страницу, которая будет повторять листинг 17.5.

Листинг 17.5. Обзор простой ссылки `island04.xml`

```
1:  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
2:   "http://www.w3.org/TR/html4/loose.dtd">
3:  <!-- листинг 17.5 island04.html -->
4:
5:  <HTML>
6:  <HEAD>
7:  <TITLE>XML Data Island</TITLE>
8:  </HEAD>
9:
10: <BODY>
11: <XML id="myMsg" src="message01.xml"/>
12: <H1>My Messages</H1>
13:  <SPAN datasrc="#myMsg" datafld="source"></SPAN><BR/>
14:  <SPAN datasrc="#myMsg" datafld="from"></SPAN/><BR/>
15:  <SPAN datasrc="#myMsg" datafld="message"></SPAN/><BR/>
16:
17:  <BR/>
18:
19:  <BUTTON onClick="myMsg.recordset.movefirst()">
20:       &lt;&lt;
21:  </BUTTON>
22:  <BUTTON onClick="myMsg.recordset.moveprevious();
23:                    if (myMsg.recordset.BOF)
24:                        myMsg.recordset.movenext()">
25:       &lt;
26:  </BUTTON>
27:  <BUTTON onClick="myMsg.recordset.movenext();
28:                    if (myMsg.recordset.EOF)
29:                        myMsg.recordset.moveprevious()">
30:       &gt;
31:  </BUTTON>
32:  <BUTTON onClick="myMsg.recordset.movelast()">
33:       &gt;&gt;
34:  </BUTTON>
35:
36: </BODY>
37: </HTML>
```

В строке 11 определяется остров данных с использованием дескриптора HTML XML. Строки 13–15 предназначены для представления одной записи; при этом используются элементы SPAN для связывания с такими элементами XML, как `source`, `from` и `message`. Сценарии кнопок, вызывающие методы DSO, рассмотренные нами раньше, содержатся в строках 19–34.

Вызов XMLDSO с помощью Java-аплета

При установке Internet Explorer на вашем компьютер также устанавливается и такая библиотека Java, как `com.ms.xml.dso`. Этот пакет включает в себя аплет XMLDSO для передачи данных XML на HTML-страницу совместно с приемами связывания данных, которые мы уже рассмотрели раньше. В следующем упражнении вызовите этот аплет для передачи данных XML в HTML-таблицу.

Аплеты DSO первоначально создавались в рамках общей стратегии обеспечения доступности различной структурированной информации HTML-страницам с помощью объектов. Первые подобные объекты предназначались для передачи на HTML-страницы записей из реляционных баз данных. Однако в дальнейшем были созданы объекты для передачи на HTML-страницы и источников данных XML.

Синтаксис вызова аплета XMLDSO на HTML-странице приведен ниже:

```
<APPLET code="com.ms.xml.dso.XMLDSO.class"
        width="100%" height="50" id="[ccskrf]">
   <PARAM NAME="url" VALUE="[URI]">
</APPLET>
```

Открывающий дескриптор `APPLET` содержит атрибут `code`. Значение атрибута `code` — это полностью определенное имя класса и идентификатор пакета для XMLDSO. Атрибуты `width` и `height` определяют необходимые размеры окна отчета, в котором отображаются сведения о возникших ошибках или индикаторы успешной загрузки. Атрибут `id` предоставляет ссылку, которая используется в любом другом месте HTML-страницы для связывания данных. Элемент `APPLET` содержит один обязательный дочерний элемент `PARAM`, который указывает адрес URL для экземпляра документа XML, выступающего в качестве набора данных.

Используйте элемент `APPLET` вместо элемента `XML` при выполнении следующего упражнения для связывания элементов HTML `TD` с элементами документа XML `message01.xml`. Вам необходимо изменить код HTML-страницы, чтобы она выглядела так, как показано в листинге 17.6. Внеся все необходимые изменения, сохраните полученный документ под именем `island05.xml`.

Листинг 17.6. Остров данных XML Data, созданный с помощью Java-аплета XMLDSO, island05.xml

```
1:  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
2:   "http://www.w3.org/TR/html4/loose.dtd">
3:  <!-- листинг 17.6 island05.html -->
4:
5:  <HTML>
6:  <HEAD>
7:  <TITLE>XML Data Island</TITLE>
8:  </HEAD>
9:
10: <BODY>
11: <applet code="com.ms.xml.dso.XMLDSO.class"
12:         width="100%" height="50" id="myMsg">
13:    <PARAM NAME="url" VALUE="message01.xml">
14: </applet>
15: <H1>My Messages</H1>
16:   <TABLE id="table" border="6" width="100%"
17:          datasrc="#myMsg" summary="messages">
18:        <THEAD style="background-color: aqua">
19:              <TH>Source</TH>
```

```
20:                 <TH>From</TH>
21:                 <TH>Message</TH>
22:           </THEAD>
23:           <TR valign="top" align="center">
24:                <TD><SPAN datafld="source"/></TD>
25:                <TD><SPAN datafld="from"/></TD>
26:                <TD><SPAN datafld="message"/></TD>
27:           </TR>
28:      </TABLE>
29: </BODY>
30: </HTML>
```

Строки 11–14 содержат аплет, который инициализирует XMLDSO. Созданный объект указывается с помощью с сылки `myMsg` (строка 12). Исходным документом XML является документ `message01.xml`. Параметры таблицы (строки 16–28) остались без изменений по сравнению с предыдущим примером.

Результаты загрузки страницы в броузер приведены на рис. 17.4.

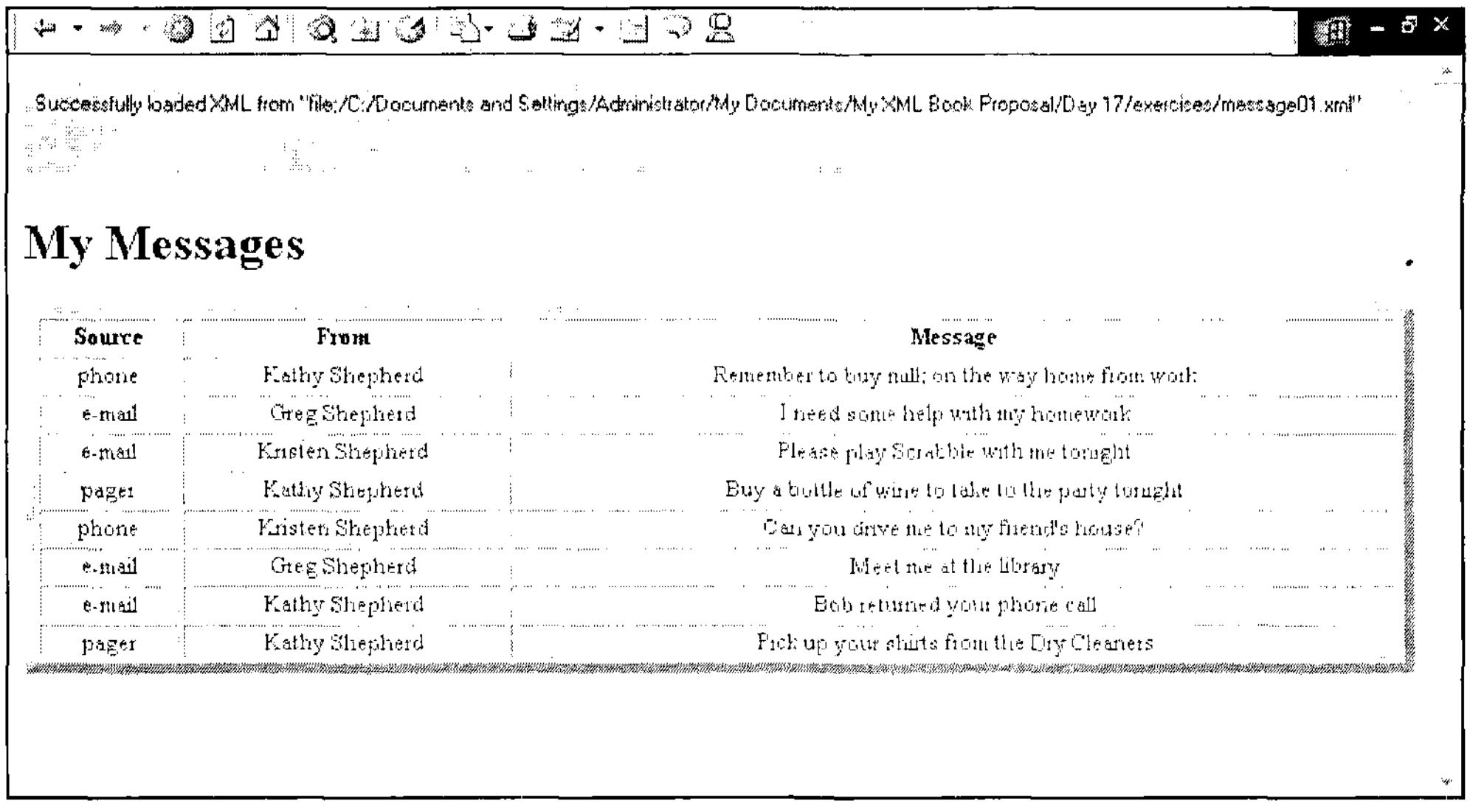


Рис. 17.4. Остров данных XML, созданный на HTML-странице с помощью Java-аплета XMLDSO

Сообщения об ошибках

Окно отчета, которое вы указали с помощью атрибутов `width` и `height`, используется для отображения сообщений, поступающих от аплета. Если вы точно выполнили все указания из предыдущего примера, вы должны увидеть небольшое окно зеленого цвета, которое сообщит вам об успешной загрузке страницы. Для того чтобы познакомиться с сообщениями об ошибке, измените идентификатор URI в элементе `PARAM` для загрузки несуществующей страницы. Вы должны увидеть сообщение об ошибке, отображенное на красном фоне. В подобных сообщениях часто приводятся рекомендации, как исправить возникшие ошибки. Поскольку при этом используется синтаксический анализатор XML, можно ожидать и соответствующие сообщения об ошибках. Очень распространены мелкие ошибки обработки документа, а также ошибки, связанные с указанием неправильного имени файла. Другими словами, если документ XML, с которым вы пытае-

тесь работать, плохо оформлен, связывание данных не сработает. Кроме того, если документ XML не найден по указанному вами адресу URL, тоже возникнет сообщение об ошибке. Перечень всех возможных ошибок при использовании XMLDSO вы найдете по адресу `http://msdn.microsoft.com/library/default.asp?url=/library/en-us/xmlsdk30/htm/xmlconusingxmldatasourceobject.asp`.

Резюме

Сегодня вы познакомились с несколькими подходами к созданию островов данных XML в HTML-страницах. Вы научились объединять интеллектуальное хранение данных, свойственное XML, с возможностями HTML по представлению данных. Вы познакомились с несколькими методами связывания элементов HTML с определенными элементами XML. Вы создали элементы управления JavaScript для управления потоком данных из исходного набора данных XML на HTML-страницу. И наконец, вы использовали Java-аплет XMLDSO для переноса данных из документа XML на HTML-страницу. Все эти приемы используются для построения инструментов поддержания и работы со структурированными документами XML.

Рассмотренные в настоящем уроке методы связывания данных очень интересны и просты для реализации, однако область их применения ограничивается доступными технологиями *Microsoft*. Поскольку эти инструменты бесплатны, они заслуживают всяческого изучения, особенно если вам необходимо манипулировать симметричными данными XML. Однако следует отметить, что требование симметричности данных оказывается достаточно серьезным ограничивающим фактором. Несмотря ни на что, связывание данных, рассмотренное в настоящем уроке, обеспечивает вас дополнительными инструментами.

Вопросы и ответы

Каким образом документ XML может выступать в качестве базы данных, когда дело доходит до связывания простых данных?

Экземпляры документов XML могут выступать в роли одноуровневых баз данных, если их структура симметрична. Дочерние элементы родительского элемента выступают в качестве аналогов записей в базе данных, а их дочерние элементы — в качестве аналогов отдельных полей.

В чем отличие элементов `DIV` и `SPAN` по отношению к связыванию данных?

Оба элемента, `DIV` и `SPAN`, представляют собой элементы-контейнеры для стилей или данных. Элемент `SPAN` используется в рамках других элементов, в то время как элемент `DIV` добавляет разрывы строк перед и после его содержания при просмотре в окне броузера.

Что произойдет, если пользователь попытается просмотреть остров данных XML с помощью Netscape Navigator?

Поскольку броузеры от компании *Netscape* не поддерживают объекты DSO, в окне броузера не будет никаких данных. Вы должны помнить, что во всех примерах в настоящем уроке были рассмотрены инструменты, разработанные компанией *Microsoft*.

Упражнение

Проверьте свои знания, полученные в результате изучения сегодняшнего урока. Ответ вы найдете в Приложении А, "Ответы на вопросы".

Используйте один из описанных в главе методов для создания связанной HTML-таблицы, содержащей данные из документа `CD.xml`.

День 18-й

Технологии XBase и XInclude

Несколько важных технологий нам обязательно следует рассмотреть, ориентируясь на будущие реализации .XML. Рекомендация XML Base (XBase) консорциума *W3C* описывает средство для определения идентификаторов URI для частей документов XML. Рекомендация XML Inclusions 1.0 (XInclude) обеспечивает спецификацию обработки для включения документов XML или их фрагментов в процесс слияния.

Сегодня рассмотрим следующие темы.

- Сведения о предложенном средстве XBase и синтаксисе, используемом для указания идентификаторов URI Base.
- Как спецификация XInclude поддерживает включение документов.
- Различия между XInclude и другими формами объединения документов XML.

XML Base

Предложение XML Base получило статус рекомендации консорциума W3C 27 июня 2001 года. XBase описывает один, касающийся пространств имен, атрибут (`xml:base`), предназначенный для указания базового идентификатора URI для разрешения полностью определенного относительного идентификатора URI в документе XML. HTML-программисты признают подобие между XBase и таким элементом HTML, как `base`, который используется для определения идентификаторов URI в ссылках HTML на изображения, таблицы стилей, аплеты и т.д.

Рекомендация XBase определяет атрибут, который вы в идеале можете разместить в элементе XLink. Однако, в будущем она может использоваться с другими технологиями XML, которым потребуется разрешение полностью определенных идентификаторов URI. На момент написания настоящей книги атрибут `xml:base` не мог похвастаться поддержкой на уровне броузеров и других приложений, хотя Netscape 6 поддерживает ряд средств, необходимых для разрешения полностью определенных идентификаторов URI. Так как в атрибуте `xml:base` присутствует префикс `xml:`, он автоматически связывается с идентификатором URI `http://www.w3.org/XML/1998/namespace`.

Атрибут xml:base

Документ содержит базовый идентификатор URI, который определяется сервером, на котором он размещен. Другими словами, когда вы определяете относительный

идентификатор URI, обработчик может разрешить и полностью определенный идентификатор URI, предполагая, что параметры сервера известны. Например, предположим, вы разместили простой элемент XLink в элементе `message` документа XML, как вы уже поступали в уроке дня 10-го. Ссылка выглядит следующим образом:

```
1:  <message xlink:type="simple"
2:       xlink:href="ks.html">
3:   Remember to buy milk on the way home from work
4:  </message>
```

В этом примере ссылка `simple` установлена на документ `ks.html` (строки 1 и 2). При этом предполагается, что ресурс, на который указывает ссылка, доступен в том же домене или на том же сервере, что и документ XML.

Вы можете переписать ссылку XLink, используя атрибут `xml:base` для полного определения расположения ссылки. Например, вам необходимо установить базовый идентификатор URI `http://www.arhitag.com/devan/` для ссылки. Синтаксис атрибута `xml:base` следующий:

```
xml:base="{URI}"
```

Указанный вами идентификатор URI будет добавлен к используемому относительному идентификатору URI (в данном случае, XLink) для получения полностью определенного URI. Другими словами, такая часть идентификатора URI, как `http://www.arhitag.com/devan/`, добавляется в начале идентификатора URI для ресурса `ks.html`. После этого полностью определенный идентификатор URI для ссылки будет разрешен как `http://www.arhitag.com/devan/ks.html`.

Вы можете задать это, добавив атрибут `xml:base` к элементу `message`, как показано ниже:

```
1:  <message xml:base="http://www.arhitag.com/devan/"
2:       xlink:type="simple"
3:       xlink:href="ks.html">
4:   Remember to buy milk on the way home from work
5:  </message>
```



Атрибут `xml:base` элемента `message` (строка 1) определяет базовый идентификатор URI как `http://www.arhitag.com/devan/`. Значение атрибута `xlink:href` равно `ks.html`. Таким образом, полностью определенный идентификатор URI для ссылки будет разрешен как `http://www.arhitag.com/devan/ks.html`.

Одно из преимуществ похода XBase заключается в том, что вы можете указывать только базовый идентификатор URI, который будет оставаться вместе с документом при перемещении последнего на другой сервер. Вы также можете использовать подход XBase для изменения относительных идентификаторов URI в больших документах XML, просто изменив базовый идентификатор URI, не изменяя все идентификаторы URI в документе.

Несколько атрибутов XBase

Поскольку атрибут `xml:base` размещается в элементе, его можно использовать в документе любое количество раз. Областью действия атрибута `xml:base` является элемент, в котором он содержится, а также все дочерние элементы этого элемента.

Таким образом базовым для элемента будет идентификатор URI, определяемый его атрибутом `xml:base`. Если для элемента не указан явно базовый идентификатор URI, он наследует его от родительского элемента, если такой, конечно же, существует. В противном случае базовым для элемента будет идентификатор URI документа, в котором он содержится. Например, изучите приведенный ниже фрагмент документа XML:

```
1:  <message xml:base="http://www.arhitag.com/devan/"
2:        xlink:type="simple"
3:        xlink:href="ks.html">
4:        Remember to buy milk on the way home from work
5:        <Reference xml:base="http://www.arhitag.com/"
6:                 xlink:type="simple"
7:                 xlink:href="index.html">
8:                 home page
9:        </Reference>
```



Элемент `Reference` связан с базовым идентификатором URI `http://www.arhitag.com`. Таким образом, гипертекстовая ссылка разрешается как полностью определенный идентификатор URI `http://www.arhitag.com/index.html`. Если атрибут `xml:base` в строке 5 опустить, то гипертекстовая ссылка `index.html` связывается с базовым идентификатором URI родительского элемента `message`, а значит, разрешается как полностью определенный идентификатор URI `http://www.arhitag.com/devan/index.html`, что является совершенно другим адресом. Следовательно, если атрибут `xml:base` опущен в строках 1 и 5, гипертекстовые ссылки в строках 3 и 7 разрешаются как базовый идентификатор URI экземпляра документа XML. Как правило, в результате идентификатор URI будет совпадать с адресом сервера, на котором хранится документ.

XML Include

Спецификация XInclude 1.0 — это еще рабочий черновик консорциума *W3C*, однако уже практически законченный. Рекомендация включает в себя модель обработки, а также предварительный синтаксис включения различных документов XML в один экземпляр. В этом вы можете увидеть идею повторного использования компонентов для создания новой программы, унаследованную от объектно-ориентированного программирования. Такие языки программирования, как C++ и Java, позволяют вам реализовать классы методов и импортировать их как отдельные компоненты. XInclude обеспечит тот модульный подход к построению сложных экземпляров документов XML. Используя XInclude, вы можете создать новый документ, указав только идентификаторы URI образующих его документов.

В процессе создания нового документа происходят определенные обработка и преобразования. С преобразованиями XSLT вы познакомились в уроке дня 16-го. Мы говорим о гораздо более простой форме преобразований, которая позволяет объединять документы в один общий. Однако полученный документ представляет новую древовидную структуру, образованную данными, добавленными во время слияния. На языке консорциума *W3C* результирующий документ еще до начала процесса слияния называется исходным набором данных (source infoset). После слияния документ называется результирующим набором данных (result infoset).

Каждый документ, который добавляется к результирующему набору данных, указывается с помощью соответствующего идентификатора URI. Таким образом, результирующий набор данных — это исходный набор данных после замены элементов XInclude соответствующими данными. Обязательный атрибут `href` элемента XInclude объявляет идентификатор URI документа XML, который будет добавляться в результирующий набор данных. Элемент `include` принадлежит пространству имен XInclude (`http://www.w3.org/1999/XML/xinclude`) и обычно указывается с помощью префикса `xinclude:` для пространства имен. Простейший синтаксис элемента XInclude приведен ниже:

```
<xinclude:include href="[URL]">
```

Например, предположим, вы работаете над руководством по правилам работы в организации, а каждое из ее подразделений предоставляет свою часть руководства. Каждое из подразделений предоставляет документ XML, содержащий определенные правила, которые должны добавляться в финальный документ. Возможно, департамент с персоналом предоставит сведения о правилах, касающихся отпуска и охраны здоровья (`HR.xml`). Финансовый департамент может предоставить правила, касающиеся совершения покупок (`Finance.xml`). Департамент управления изложит основные цели, которые ставит перед собой организация (`Admin.xml`). Вы можете создать главный документ (`policy.xml`), который будет содержать полное руководство и *включать* в себя все документы, предоставленные отдельными департаментами. Соответствующая схема представлена на рис. 18.1.

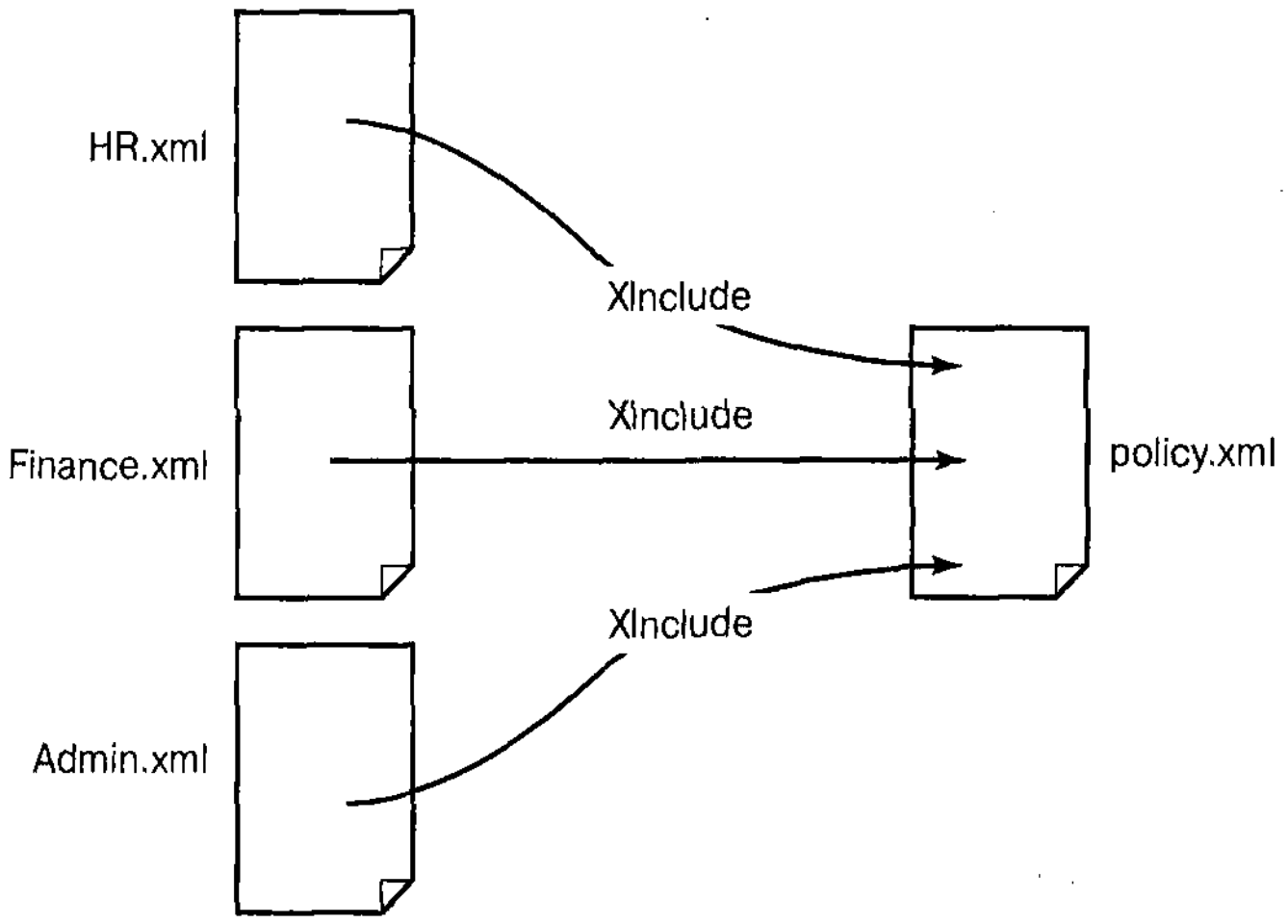


Рис. 18.1. Слияние документов с помощью XInclude

Ваш документ `policy.xml` содержит элементы XInclude с атрибутами `href`, соответствующими документам `HR.xml`, `Finance.xml` и `Admin.xml`. Возможный вариант документа `policy.xml` приведен в листинге 18.1.

Листинг 18.1. Исходный набор сведений с элементами XInclude `policy.xml`

```
1: <?xml version="1.0"?>
2: <!-- листинг 18.1 - policy.xml -->
3:
4: <manual xmlns:xinclude="http://www.w3.org/1999/XML/xinclude">
5:    <prolog>Corporate Procedure Manual</prolog>
6:    <section><xinclude:include href="HR.xml"/></section>
7:    <section><xinclude:include href="Finance.xml"/></section>
8:    <section><xinclude:include href="Admin.xml"/></section>
9: </manual>
```

АНАЛИЗ Корневой элемент `manual` содержит атрибут `xmlns:xinclude`, который объявляет пространство имен XInclude (`"http://www.w3.org/1999/XML/xinclude"`). Каждый дочерний элемент `section` корневого элемента (строки 6–9) содержит атрибут `xmlns:xinclude` со значением `href`, соответствующим одному из документов (`HR.xml`, `Finance.xml` и `Admin.xml`), которые объединяются в документе `policy.xml`.

Обработка XML во время включения

Необязательный атрибут `xinclude:parse` можно включить в элемент XInclude для объявления того, должен соответствующий документ считаться документом XML или обычным текстовым документом. Если этому атрибуту присвоить значение `xml`, документ считается документом XML, что приводит к выполнению перед слиянием подстановки строк, проверки выражений XInclude и других задач. Другое возможное значение атрибута `xinclude:parse`, равное `text`, приводит к тому, что документ считается обычным текстовым документом, поэтому его обработка не выполняется.

Выражения XPointer

Атрибут `href` содержит выражения XPointer для включения фрагмента из указанного документа, а не целого документа. С выражениями XPointer вы познакомились в уроке дня 11-го. Предположим, вам известна структура файла `HR.xml`, объявленного раньше, и он содержит определенное количество дочерних элементов `paragraph` в каждом, если вы хотите включить дочерние элементы `paragraph` из раздела `introduction`, используйте XInclude совместно с XPointer. Соответствующая часть исходного набора данных может выглядеть следующим образом.

```
<section>
<xinclude:include href="HR.xml#pointer(introduction/paragraph)"/>
</section>
```

Поддержка XInclude

На момент написания настоящей книги спецификация XInclude еще не получила широкой поддержки на уровне приложений. Броузеры не поддерживают преобразования XInclude; вам необходимо создавать программный код для выполнения соответствующих операций вручную. Для этого можно использовать, например, язык Java. XInclide получит поддержку на уровне броузеров не раньше, чем станет полноценной рекомендацией консорциума *W3C*.

У вас может создаться впечатление, что технология XInclude очень похожа на XLink, если атрибуту `show` присвоить значение `embed`, но на самом деле это не так. В результате использования XLink и значения `embed` атрибута `show` новая древовидная структура не создается. Использование же XInclude совместно с преобразованиями приводит к появлению совершенно нового результирующего набора данных. Другими словами, XLink с `show="embed"` не приводит к созданию новой древовидной структуры XML.

Резюме

В настоящем, достаточно небольшом уроке, вы познакомились с двумя новыми и активно развивающимися технологиями — XBase и XInclude. Вы узнали, что XBase позволяет указать базовые идентификаторы URI для относительных ссылок в документе XML. Это очень напоминает использование элемента `BASE` HTML. Вы также познакомились с технологией XInclude, которая позволяет объединять данные из нескольких документов в виде одного результирующего набора данных. Если обработчик, поддерживающий технологию XInclude, и обнаружит в исходном документе элементы `xinclude`, будет подстановка данных вместо указанных идентификаторов URI. Кроме того, можно проверить полученный документ на

соответствие всем стандартам XML. Хотя XML еще не обрела поддержки на уровне броузеров, это обязательно произойдет, как только эта технология получит статус полноценной рекомендации *W3C*.

Вопросы и ответы

Каким образом базовый идентификатор URI изменяется при переносе документа XML на новое место?

Базовый идентификатор URI отражает новое месторасположение документа. Однако, если вы указали идентификатор URI для использования относительными ссылками в документе, они остаются неизменными даже при перемещении документа.

Существует ли ограничение на количество документов, объединяемых с помощью технологии XInclude?

Количество документов, которые могут объединяться с помощью технологии XML Inclusions, не ограничено. Именно благодаря этому и появляется возможность создания очень больших и сложных документов.

Какие ограничения технологии XInclude существуют?

Единственное огромное ограничение технологии XInclude заключается в отсутствии поддержки со стороны броузеров. Как только это произойдет, построение модульных документов XML станет реальностью.

Существуют ли в настоящее время обработчики, поддерживающие технологию XInclude?

На момент написания настоящей книги такой программный продукт, как Cocoon, обеспечивает поддержку XInclude. Это приложение представляет собой набор программных средств, созданных на Java. Cocoon использует ряд ключевых технологий *W3C*, таких как XML, XSL, DOM и XInclude для представления Web-содержимого.

День 19-й

Интеграция XML с корпоративными бизнес-моделями

В традиционных бизнес-приложениях все чаще и чаще начинают использовать технологии XML. Целый ряд корпоративных систем предназначен для управления распространением информации, предоставлением данных и интеграцией приложений, а также для максимизации управления знаниями и рабочим потоком, что значительно упрощается при использовании XML. XML для улучшения бизнес-процессов и получения новых возможностей становится стратегической целью многих организаций.

Сегодня рассматриваются следующие темы.

- Основы анализа документов.
- Роль XML в корпоративных приложений.
- Значение трехуровневой архитектуры для Web-приложений.
- Несколько сценариев промежуточного уровня для перемещения информации из корпоративных баз данных на динамические Web-страницы с использованием XML.

Создание бизнес-моделей с использованием технологий XML

В настоящей книге я рассказал вам о различных отдельных технологиях, образующих XML, а также о том, каким образом их использовать для организации обмена структурированными данными. Вы уже познакомились со стандартами и предварительными рекомендациями, относящимися XML. Вы исследовали механизмы, которые выходят за рамки XML и используются для манипуляции данными XML и их отображения. Сегодня вы познакомитесь с несколькими способами использования XML для построения интегрированных бизнес-приложений. Вам предлагается несколько сценариев построения приложения, их анализ с точки зрения бизнеса, преимущества использования XML, рассмотрение различных вариантов архитектуры приложений, а также рассмотрение общих вопросов разработки приложений на базе технологий XML. После этого вам предложат создать работающее решения для обмена

данными с использованием сценариев на стороне сервера (Active Server Pages) для извлечения данных из базы и их представления в виде набора данных XML. После этого вы напишете сценарии для перемещения данных из базы в виртуальную структуру XML, которая может связываться с элементами таблицы на HTML-странице для дальнейшего отображения в окне броузера.

Основные положения анализа документов

Один из типичных шагов в создании XML-приложений — анализ документов для выявления наилучшего способа их разметки. Вы можете преобразовать документ на бумаге или, что вероятнее, целый класс документов, таких как описания процедур, меморандумы, отчеты, контракты или меню в ресторане, в экземпляры XML. Иногда преобразование приводит к использованию существующей схемы, определяющей структуру документа, а в других случаях вам необходимо начинать с нуля. В любом случае вам необходимо внимательно изучить примеры каждого класса документов, который вам нужно создать, чтобы понять его структуру и состав.

Анализ документа включает исследование частей, целей и требований одного или нескольких классов документов для получения подробного описания каждого из них. На практике это описание выражается в виде отчета об анализе документа (Document Analysis) и в соответствующем наборе инструкций схем, предназначенных для определения структуры документа. Применение для определения структуры документа коммерческой схемы также возможно, однако анализ все равно необходим для того, что все действительно было "в ажуре". Сама природа схем XML позволяет вам реализовать все необходимые элементы структуры.

Самое важное при получении точного и полезного описания класса документа — сбор точных и полезных сведений обо всех документах, образующих класс. Причем лучшим источником таких сведений оказывается не просмотр документов, а общение с людьми, которые отвечали за их создание и поддержку. Не располагая подобными сведениями, вы никогда не сможете создать действительно эффективную структуру документа.

Попросту говоря, только авторы документов действительно знают, что полезное и важное в них содержится. Если вам необходимо выполнить анализ документа, получите необходимые сведения от людей, а не от документов — представителей класса. Использование только документов оказывается неполным. Кроме того, анализ только документов редко позволяет получить все необходимые сведения. Очень важно, чтобы вы понимали назначение отдельных компонентов данных или их источников, их роль в функционировании целой системы.

Прибегаете вы к помощи создателей документов или нет, анализ означает идентификацию и определение компонентов классов документов посредством применения наборов дескрипторов для данных, которые вам необходимо представить. На самом деле, выбранный вам уровень "точности" — одна из самых сложных задач, стоящих перед вами на ранних этапах анализа документов. Небольшое невнимание может привести к значительным последствиям, а повышенное внимание оказывается излишним. Как и в случае любой другой разработки программного обеспечения, анализ оказывается очень важным шагом на пути к успеху, экономии времени и затрат.

Результатом анализа может оказаться принятие или создание словаря разметки для описания целого класса документов. Поскольку "авторские" типы элементов являются одной из основных характеристик XML, любые совместные действия при создании словаря разметки должны тщательно планироваться. Например, предположим, вы работаете над языком разметки для корпоративной бухгалтерской системы, в составе целой команды. Если каждый из членов команды создаст собственный дескриптор для номеров заказов (например `Ponum`, `Purchase_order_number`, `PO_number`, `PON` или `P_O_number`), а после создаст соответствующие имена для элементов этого типа,

применять ко всей системе один словарь разметки станет невозможно. Таким образом, полный анализ нужно выполнить до того, как начнется работа над разметкой, а словарь разметки надо разослать всем членам команды, прежде чем начинать работы над отдельными документами.

Для обеспечения эффективной разметки вам необходимо хорошо понимать, какие именно цели должна решать создаваемая вами система. Если, например, основной целью является извлечение, повторное использование и обмен информацией, вы должны определить имена элементов, типы данных, а также ошибки проверки для каждого из них.

Вы можете прийти к выводу, что групповой подход станет эффективным в значительной части проектов по анализу документов XML, однако не забывайте о том, что стоит избегать лишних действий и не создавать ненужные, дублирующиеся типы данных. Очень часто вам потребуется создать словарь разметки для обмена данными в областях, контролировать которые вы не в состоянии. В некоторых случаях вы можете использовать уже готовые схемы, которые можно найти в Internet. К наиболее популярным источникам схем относятся следующие.

- Biztalk.org (свыше 300 схем для 11 категорий, созданных 75 организациями).
- Xml.org (свыше 200 схем для более чем 80 категорий, созданных различными организациями).
- Schema.net (очень много схем для основных категорий, созданных специальными группами и организациями).

Если вы не хотите использовать готовую схему, лучше создать новую схему, базирующуюся на структуре документа. Вы должны, конечно же, знать необходимый набор дескрипторов, информацию, которая содержится в элементах, отношения между ними, порядок следования элементов и т.д.

Шаги анализа XML

К вашим услугам несколько подходов к анализу и несколько готовых систем, способных решать подобные задачи. Значительная часть подходов направлена на решения одних и тех же задач. Анализ позволяет определить набор связанных и полезных данных для моделирования. Создав схему или отчет об анализе, вы создаете базовую инфраструктуру для иерархии компонентов данных и самой возможности обмена данными.

Вполне вероятно, что вам нужны чрезвычайно подробные метаданные, которые помогут улучшить контекстный и точный поиск информации в документах. Размещение информации в структурах — таких как заключения, списки ингредиентов, процедур, авторов и т. д. — позволяет вам создавать контекстные ссылки на данные.

Определение требований

Определение требований — это один из самых важных шагов при проведении анализа. Требования влияют на решения по поводу архитектуры, анализа и проектирования систем. Четкое понимание требований позволит вам определить необходимую точность определения элементов. Требования позволяют вам выбирать необходимые компромиссные решения между проектированием и разработкой при возникновении каких-либо конфликтов. Определяя требования, вспомните следующее.

- Задачи приложения: зачем вы должны (или хотите) использовать XML и кто использует сохраненную, созданную или обработанную информацию? Каковы требования к обмену, поиску, безопасности и повторному использованию данных? Какие задачи, которые вы не можете решить в данный момент, вам обязательно необходимо решить?

- Задачи, которые не относятся к приложению: каковы реальные ограничения, которые вы хотите преодолеть? Что будет опущено при проектировании приложения и почему?
- Типичные (желаемые) результат работы системы: должны результаты представляться в форме электронных книг, Web-страницы, общих данных в сети предприятия, "твердой копии", данных для службы поиска, мультимедийных или других форматов?
- Требования организации: каковы особенности маркетингового подхода организации? Оправдаются ли затраченные усилия?
- Существующие стандарты документов: каковы принятые в организации стандарты ведения личных данных и данных общего доступа?
- Существующие стандарты приложений: какие схемы используются в организации?

Определение области действия

Такое же важное как и определение требований, выделение области действия позволяет обозначить границы разметки. Во многих случаях целью проекта оказывается увеличение или уменьшение существующей области действия. Вам необходимо понимать, каково влияние тех или иных вносимых изменений. Например, можно ли класс маркетинговых документов расширить так, чтобы включить новые разработанные продукты? Если да, то каким образом новый бизнес-план повлияет на ваше приложение? Обеспечили ли вы необходимые возможности расширения?

Идентификация набора элементов

Определять набор элемента достаточно сложно. Вам нужно рассмотреть, например, подходят ли существующие типы документов для нового приложения. Очень часто бумажные документы создаются из соображений удобства и простоты, а не обеспечения оптимальной структуры данных. Возможно, вы будете иметь дело с экземпляром XML, содержащим сведения из нескольких бумажных документов и передающим данные определенным пользователям.

Определив необходимый набор сведений, выберите названия элементов, используя в качестве критериев значимость, возможность повторного использования, возможность поиска и т.д. Подобное решение очень часто также оказывается достаточно проблематичным. Как вам убедиться в том, что достигнута необходимая "точность" определения набора элементов? Некоторые проблемы разметки просто не могут иметь идеального решения; поэтому сохранение и поддержание данных, которые вам никогда не потребуются, ненужное и дорогостоящее занятие.

Идентификация элементов требует от вас избирательности и чувствительности к данным и их структуре. Например, документ может содержать разделы, заголовки, абзацы, слова и знаки препинания. Тот же документ можно описать с помощью таких компонентов, как To (кому), From (От), Subject (Тема), Date (Дата), Regarding (касательно) и Body (основная часть).

Определение отношений между сведениями

Анализ включает в себя и понимание отношение между различными компонентами сведений. Например, вам необходимо обеспечить определенные иерархические отношения между элементами. Элементы — это контейнеры, поэтому вам необходимо точно знать, что должен содержать каждый из них. Например, предположим, вы создаете файл XML для хранения данных, которые передаются приложению управления бухгалтерскими записями. Элемент-контейнер может содержать такие сведения, как

номер журнала, одно или несколько действий. В свою очередь, к действиям могут относиться такие сведения, как номер учетной записи, сумма и т.д.

Из полученного опыта использования схем, вы помните что порядок элементов, перечисление значений атрибутов, определение обязательных элементов и атрибутов — это очень важные характеристики. Во время анализа важно определить и различные зависимости элементов. Например, в системе управления транзакциями сведениям о кредитной информации должны соответствовать сведения о дебитной информации. Знание необходимых отношений между элементами поможет вам лучше спроектировать систему.

Другие формы анализа информации

Вы познакомились с некоторыми приемами, полезными для анализа статических данных, например бумажной документации. Если вы создаете документ XML на основе существующей базы данных, не пытайтесь использовать словарь данных в качестве источника сведений о необходимой структуре. В следующих разделах вы познакомитесь с несколькими типичными бизнес-проблемами, которые решаются благодаря использованию технологий XML. В большинстве случаев в подобных системах есть одна или несколько баз данных, содержащих данные, обмен которыми реализуется в виде обмена документами XML.

Бизнес-приложения

Использование технологий XML в корпоративных приложениях набирает все большую популярность. Применение подобных технологий часто повышает эффективность деловых процессов, улучшает выполнение транзакций, а также способствует выработке новых парадигм ведения бизнеса. Приняв во внимание подобные цели, переходя к технологиям XML становится стратегической задачей для многих предприятий. В следующем разделе вы рассмотрите преимущества использования XML в нескольких ситуациях. Это будут примеры приложений, реализованных с использованием XML. Внимательно изучите приведенные примеры и попробуйте определить, *каким образом XML и сопутствующие технологии добавляются к описанным классам* приложений. Затем представьте себе, каким образом вам использовать подобные подходы для решения задач, которые стоят перед вашей организацией. Для каждого из рассмотренных нами сценариев вам предлагаются следующие сведения.

- Краткий обзор проблем, которые должны решаться.
- Потенциальные преимущества использования XML.
- Проблемы, связанные с разработкой решений.

Приложения для распространения и сбора информации

Предположим, вы хотите использовать технологии обмена данными для управления и координирования работы организации. Возможно, вы заняты созданием нового товара или маркетинговым продвижением новой услуги. Возможно, вам необходимо организовать обмен инженерными данными между различных подразделениями компании для принятия решения, касающихся производства, маркетинга и т.д. В зависимости от размеров вашего проекта вам может потребоваться система для управления, контроля и поддержания потока данных. В работе над системой могут участвовать как ваши сотрудники, так и приглашенные специалисты, поэтому очень важно обеспечить эффективный обмен данными. Вам необходимо обеспечить надежную обработку всех данных.

Сегодня вы уже познакомились с шагами, которые обычно используются при анализе документов, которые необходимо преобразовать в структуры XML. Может оказаться так, что информация, которая содержится в документах, имеет небольшое значение, поэтому тратить усилия на разработку соответствующих структур XML не стоит. Для все большего числа организаций информация становится тем товаром, который производится и продается потребителям. Распространение информации очень часто оказывается важной составляющей всего процесса доставки товара. Эффективная, быстрая и точная доставка информации ставит организацию в очень выгодное положение в современной экономике. Очень часто в качестве канала распространения информации выбирается Web, которая обеспечивает эффективность и возможность глобального доступа. Web может использоваться для обмена сведения между деловыми партерами, клиентами и сотрудниками, которые находятся в одном здании, на одной улице или различных частях мира.

Преимущества XML

Как уже отмечалось раньше, в уроках 1 и 2, HTML, несмотря на все свои преимущества, например, на платформно-независимость, характеризуется целым рядом недостатков. Там, где необходимо работать со структурированными данными, а особенно там, где необходимо организовать динамический доступ к данными, HTML вряд ли можно считать лучшим вариантом выбора. Построив приложения, базирующиеся на различных технологиях XML, с которыми вы познакомились на страницах настоящей книги, вы сможете преодолеть все эти ограничения.

Разделение стиля и содержания в XML — это один из факторов, приводящий к успешному представлению динамического содержания на Web-страницах. При использовании HTML вам требовалось внедрять в стандартные дескрипторы для разметки текстовых данных дополнительные элементы, которые определяют форматирование отдельных компонентов страницы. HTML не обеспечивал решение всех этих задач без привлечения дополнительных Web-технологий. В противоположность этому, XML и XSL предоставляют в ваше распоряжение все необходимые средства. Кроме того, как вы узнали из урока дня 16-го, вы можете преобразовать любой документ XML к формату HTML для представления данных с использованием различных агентов пользователя, обладающих поддержкой HTML.

Локальные пользователи выбирают таблицы стилей для применения к данным, сохраненным в документе XML, а приложения создаются для доступа к данным без использования каких-либо таблиц стилей. Дальше, в этом уроке, вы создадите сценарий, выполняемый на стороне сервера, который будет помещать данные XML в HTML-таблицу, отображаемая с помощью любого Web-броузера. Подобные преимущества позволяют охарактеризовать XML как выдающуюся технологию для распространения информации, а не только как технологию публикации страниц.

Вопросы, связанные с архитектурой и разработкой

Распространение и сбор информации идеально подходят для архитектуры, которая базируется на использовании Web-технологий. В типичной ситуации пользователи информации отправляют свои запросы серверу, который извлекает необходимые сведения и конструирует ответ. С результирующим документом можно связать таблицу стилей, его можно преобразовать, а затем доставить агенту пользователя. Поскольку универсальная поддержка XML на уровне броузеров до сих пор отсутствует, значительная часть систем сбора и распространения информации полагается на предварительную обработку данных на стороне сервера перед их доставкой.

Вы уже убедились в том, что при этом используются как статические, так и динамические данные. Статическое содержание, как правило, получается из заранее соз-

данных документов. Динамическое содержание конструируется с помощью специального сервера сбора данных. В любом случае, XML предлагает значительные преимущества перед решениями, основанными на HTML и собственных механизмах конструирования данных, например, СУБД. Я совсем не хочу сказать, что вам вообще нужно отказаться от использования HTML и баз данных. Обе технологии играют исключительно важную роль в определенных ситуациях. Используйте эти технологии там, где работают они лучше всего — отображение страниц в окне броузера или эффективное сохранение структурированных данных.

При создании подобных приложений вы обычно принимаете решение о природе и классах документов и данных, которые должны распространяться, после чего создаете шаблоны для их доставки. На практике такой шаблон состоит из схемы и одной или нескольких таблиц стилей. В случае статических документов авторы могут использовать автоматические приложения, которые позволяют создать базу данных и доставить данные в электронной форме. Кроме того, деловые партнеры могут предоставить часть данных в электронной форме или через Web. Как вы уже видели, расширяемость и взаимодействие технологий XML позволяет вам совместно использовать схемы, созданные компаниями-разработчиками, вашими деловыми партнерами и вами лично.

Решения для интеграции приложений

Во многих организациях пользовательские приложения образуют целую программную среду, с которой имеют дело все сотрудники. Отдельные подразделения организации могут использовать приложения, которые даже работают на различных платформах. И это вовсе не результат неправильного проектирования или бизнес-планирования; это следствие самой природы бизнес-приложений. Бизнес-приложение — это специализированное приложение, предназначенное для решения строго определенных задач. Для достижения своих целей организация может использовать целый ряд программных пакетов. Проблема возникает в тот момент, когда появляется необходимость обмена данными, полученными с помощью различных приложений. Традиционные методы интеграции данных дороги, медленны и требуют больших трудозатрат, а часто и обращения к сторонним консультантам, которые специализируются на интеграции бизнес-приложений.

Преимущества XML

Поскольку XML обращается со сохраненными данными как с текстом, эта технология предоставляет возможности интеграции, легко реализуемые с помощью сценариев и находящие массу различных применений. Сама природа интеллектуального хранения обеспечивает, что метаданные представляют собой часть инфраструктуры передачи данных. Несколько специализированных приложений XML предназначено исключительно для обмена данными между приложениями. Например, серверные решения от компаний *WebMethods* (`http://www.webmethods.com`) и *BizTalk* (`http://www.biztalk.org`) выполняют преобразование бизнес-логики различных приложений. Несмотря на то, что это выходит за рамки рассмотрения настоящего урока, вам следует знать, что все эти основанные на XML решения выполняют интеграцию приложений даже в том случае, если эти приложения находятся в Internet или в корпоративных сетях различных предприятий. Завтра, в следующем уроке, вы познакомитесь с протоколом SOAP (Simple Object Access Protocol — простой протокол доступа к объектам), а также узнаете, каким образом полностью платформно-независимое решение обеспечивает использование удаленных объектов с помощью вызовов методов, заключенных в контейнерах XML.

Вопросы, связанные с архитектурой и разработкой

Архитектура интеграции приложения подразумевает передачу данных XML из одной программы в другую по локальной сети, а при необходимости и через Internet. Дальше в настоящем уроке вы создадите HTML-приложение, которое хранится на Web-сервере. После загрузки приложения оно будет запускать отдельный сценарий на стороне сервера, написанный на VBScript. Этот сценарий будет обращаться к базе данных Microsoft Access, используя для этого выражения SQL (Structured Query Language — язык структурированных запросов), а результаты будет возвращать в виде потока данных XML. HTML-приложение будет связывать элементы XML, используя простое связывание данных, после чего отображать таблицу в виде Web-страницы. Это всего один пример интеграции приложений с использованием технологии XML.

Для приложения, которое вы создадите, процесс разработки разделяется на несколько этапов. Прежде всего, вам необходимо разработать драйвер протокола или использовать один из существующих, который сможет определить инструкцию запроса. При этом должна обеспечиваться установка соединения с Web-сервером для выполнения приложения, которое находится на сервере. Это приложение должно выполнить необходимые действия, после чего сгенерировать ответ, который будет отправлен обратно. После этого вы создадите связывающий уровень, который преобразует данные, содержащиеся в ответе, в структуры, которые будут обработаны локальным приложением.

Приложения интеграции данных

Иногда данные, которые вам необходимы для работы, содержатся не в одной базе данных, а в нескольких базах, которые, в свою очередь, могут находиться в сетях нескольких предприятий. Целый класс приложений, известный как приложения интеграции данных, способен удовлетворить требования к сбору данных и унифицированному выводу результатов. Подобная интеграция способна сохранить время и силы сотрудников, которым необходимо консолидировать данные из нескольких источников для выполнения сложных бизнес-транзакций. Стоимость запуска всех отдельных приложений больше суммы выполнения всех отдельных операций, так как надо учитывать действия вручную, необходимые для объединения результатов или получения единого отчета. Автоматизация подобных процессов имеет большое значение.

Преимущества XML

Централизованный синтез сложных бизнес-данных — это основное назначение приложений интеграции данных. Подобное приложение должно получить многонаправленный доступ к различным компонентам; именно здесь и проявляется целый ряд преимуществ XML. Другими словами, результирующий синтез данных потребуется для обратной связи с приложениями, участвующими в интеграции данных. Возможности технологий XML при обмене данными в некоторых ситуациях превращают формат XML в стандарт де-факто, используемый при выполнении бизнес-транзакций. Все это еще достаточно слабо используется на стороне клиента, где поддержка технологий XML еще недостаточна, однако в данном направлении уже ведутся интенсивные разработки. Вплоть до настоящего времени синтез данных можно выполнять только на сервере, и только после этого данные передаются агентам пользователя в том или ином формате.

Вопросы, связанные с архитектурой и разработкой

Как правило, решения интеграции данных построены на архитектуре, которая характеризуется сервером интеграции, расположенным между различными клиент-приложениями, с одной стороны, и несколькими базами данных и другими источни-

ками — с другой. Источники данных представляют собой записи баз данных, файлы в файловой системе или приложения. Единственное требование состоит в том, что к источникам данных должен быть удаленный доступ. Сервер интеграции получает данные от источников, преобразует их в потоки данных XML, после чего передает их клиент-приложениям в приемлемой форме. Сервер также запрашивает обновленные данные у клиент-приложений, разделяет их на составные части и сохраняет должным образом. Доступ к источникам данных осуществляется собственными механизмами XML, с помощью транспортных протоколов баз данных и файловых систем или же с помощь специальных прикладных интерфейсов API, таких как JDBC (Java DataBase Connectivity — Java-интерфейс доступа к базам данных) и ODBC (Open DataBase Connectivity — открытый интерфейс доступа к базам данных).

Трехуровневая Web-архитектура

В настоящей главе вы познакомитесь с архитектурами, которые можно охарактеризовать, как *трехуровневая модель "клиент/сервер"*. Это специальная архитектура "клиент/сервер", состоящая из трех четко определенных и отдельных процессов, каждый из которых выполняется на отдельной *платформе*.

Уровень клиента

Интерфейс пользователя, который выполняется на компьютере пользователя (который и является клиентом), очень часто характеризуется броузером. Любой другой агент пользователя на стороне клиента, такой как беспроводное устройство с поддержкой Web-технологий, сотовый телефон или персональный цифровой помощник PDA, также может предетавлять этот уровень. Это первый из трех уровней, часто называемый уровнем клиента, или уровнем пользователя.

Промежуточный уровень

Промежуточный уровень содержит функциональные модули, которые отвечают за обработку данных, и выполняется на одном или нескольких серверах, один из которых является сервером приложений. Еще один класс серверов — Web-серверы, также относится к промежуточному уровню и выступает в качестве "порталов" для других приложений промежуточного уровня. Приложение распространения и сбора информации, с которым вы познакомились раньше в этом уроке, — это пример приложения, которое должно выполняться на промежуточном уровне. Сценарии, предназначенные для выполнения на сервере, также используются и на серверах промежуточного уровня.

Предположим, вы посетили электронный книжный магазин, а Web-узел приветствует вас по имени, а также предлагает список новых поступлений, созданный на основе ваших предыдущих покупок. Возможно, вы уже приобретали книги, подобные этой, посвященные тем или иным компьютерным технологиями. Приложение персонализации также выполняется на промежуточном уровне. Оно чаще всего считывает данные из сохраненных записей базы данных на уровне данных, после чего определяет ваши предпочтения, обрабатывает результаты на промежуточном уровне, формирует новые списки, соответствующие вашим предпочтениям, после чего предлагает вам именно те книги, которые вы захотите приобрести. Это классический пример трехуровневого Web-приложения.

Уровень данных

Как уже отмечалось, третий уровень, известный как уровень данных, чаще всего включает систему управления базами данных (СУБД), которая сохраняет данные, необходимые на промежуточном уровне. Этот уровень функционирует на втором сервере — сервере баз данных. Профили подписчиков и всевозможные каталоги, подобные тем, что используются рассмотренным нами электронным магазином, обычно сохраняются в базах данных на уровне данных.

Вы придете к выводу, что XML может использоваться на любом из этих уровней. Различные приложения, с которыми вы познакомились раньше в этом уроке, являются примерами того, каким образом XML может использоваться на всех уровнях такой популярной архитектуры. В следующих упражнениях вы построите приложения на базе XML, которые будут взаимодействовать и выполняться на всех трех уровнях. Эта наиболее популярная модель разработки соответствует подходу, который продвигает консорциум *W3C* среди Web-разработчиков.

Использование XML на различных уровнях

В последующих упражнениях вы будете сегодня использовать программирование на промежуточном уровне для доступа к данным, которые хранятся в базе данных. После этого вы передадите данные с помощью специально для этого созданных структур XML броузеру для отображения на стороне клиента. В этих упражнениях мы будем использовать следующие технологии.

- XML (Extensible Markup Language — расширяемый язык разметки) версии 1.0 — для передачи данных между уровнями.
- HTML (HyperText Markup Language — язык разметки гипертекста) — для отображения данных в окне броузера на стороне клиента.
- Microsoft Access 2000 — для сохранения записей баз данных и представления уровня данных.
- Microsoft Internet Information Services (IIS) версии 5.0 — для предоставления локального Web-сервера для проверки и выполнения приложений.
- Операционная система Microsoft Windows 2000.
- ODBC Data Source Administrator версии 3.5 — для определения имени источника данных DSN (Data Source Name) и того, каким образом осуществляется подключение к базе данных.
- Язык SQL (Structured Query Language — язык структурированных запросов) — для извлечения данных из базы данных под управлением сценария, который выполняется на промежуточном уровне.
- Язык VBScript для создания Active Server Pages — для выполнения сценариев на стороне сервера.

Для того чтобы выполнить дальнейшие действия и создать приложение, вы можете использовать практически любые указанные компонентные технологии. Допустима использовать другие, более известные вами компоненты — например, другую СУБД или язык для создания сценариев на промежуточном уровне. Однако все, что не касается применения XML, выходит за рамки рассмотрения настоящей книги, поэтому вам придется самостоятельно разбираться в использовании тех или иных технологий. Я предлагаю вам только краткое описание каждой из них, а более подробный материал, касающийся применения SQL, IIS, VBScript, HTML и ODBC API или Microsoft Access, придется искать самим.

ПРИМЕЧАНИЕ

Вы также должны знать, что продукты компании *Microsoft* используются в учебных целях. Вы можете спокойно использовать другие продукты в качестве Web-серверов и т.д. Например, вместо описанного набора программных компонентов вы можете установить Apache, Tomcat и MySQL.

Уровень данных

Продолжая наш пример системы управления сообщениями и напоминаниями, основанной на XML, создайте базу данных, в которой в качестве отдельных записей будут выступать сообщения. Поля записи будут соответствовать таким уже известным по предыдущим урокам компонентам, как `id`, `from`, `source` и `message`. Поле `id` в базе данных будет относиться к типу данных `autonumber`, а поля `from`, `source` и `message` — к типу данных `text`. Вы должны помнить, что, несмотря на использование в примере Microsoft Access, вы можете использовать любую другую ODBC-совместимую СУБД, которая поддерживает SQL.

Для создания базы данных с помощью Access, запустите приложение и выберите переключатель Blank Access database в группе Create a new database using диалогового окна Microsoft Access, как показано на рис. 19.1.

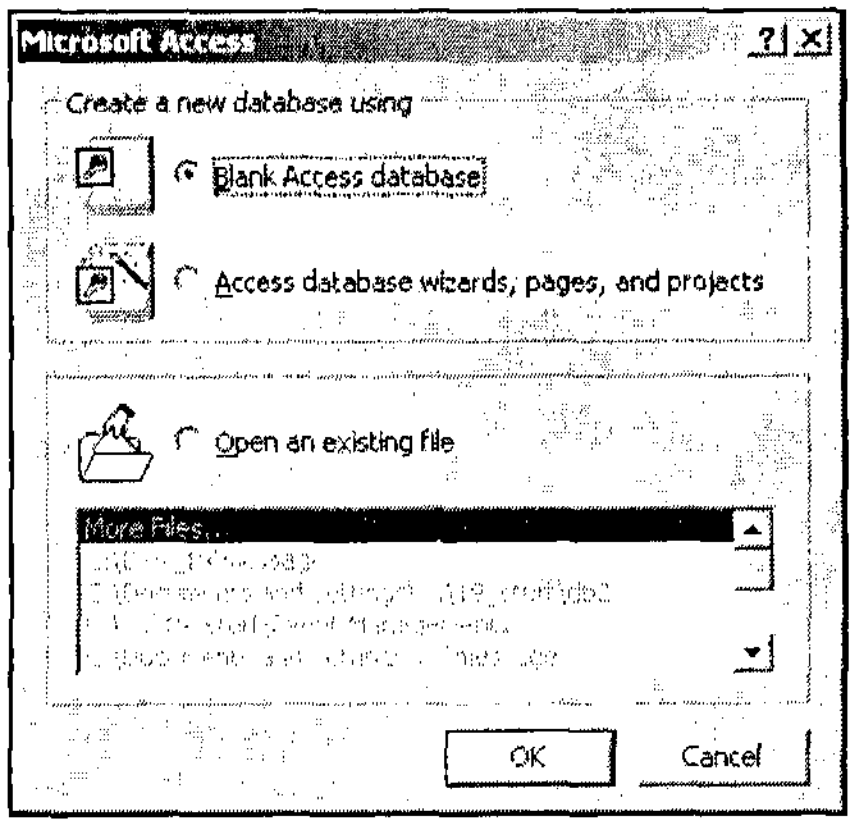


Рис. 19.1. Создание новой базы данных Access

На экране появится диалоговое окно, в котором вас попросят сохранить новую базу данных. Для выполнения упражнений настоящей главы создайте папку `c:\day_19` и сохраните в ней новую базу данных `message.mdb`. Конечно, вы можете выбрать любую другую папку, но в этом случае вам придется должным образом изменять все указанные пути. Если вы создали папку `c:\day_19`, то полный путь к базе данных будет иметь следующий вид:

```
c:\day_19\message.mdb
```

Сохранив новую базу данных, вы увидите главное окно конструктора Access. Выберите вариант Create table in Design view (Создание таблицы в режиме конструктора) для объекта Table (Таблица). Этот вариант уже должен быть выделен (рис. 19.2).

В каждой из строк появившейся таблицы вам предстоит ввести сведения о каждом поле. В первой строке введите `id` в столбце Field Name (Имя поля), после чего нажмите клавишу <Tab> для переходу к следующему столбцу, Data Type (Тип данных). Из раскрывающегося списка выберите вариант AutoNumber (Счетчик). Прежде чем переходить к следующему полю, сделайте поле `id`, ключевым, щелкнув на кнопке Primary

Key (Ключевое поле) на панели инструментов программы или выбрав команду Primary Key из меню Edit (Правка). Щелкните во второй строке в столбце Field Name (Имя поля) и введите from, после чего выберите из раскрывающегося списка вариант Text (Текстовый) в столбце Data Type (Тип данных). Повторите эти действия для полей to и message, для которых также следует указать текстовый тип данных. После всего этого окно конструктора должно принять такой вид, как показано на рис. 19.3.

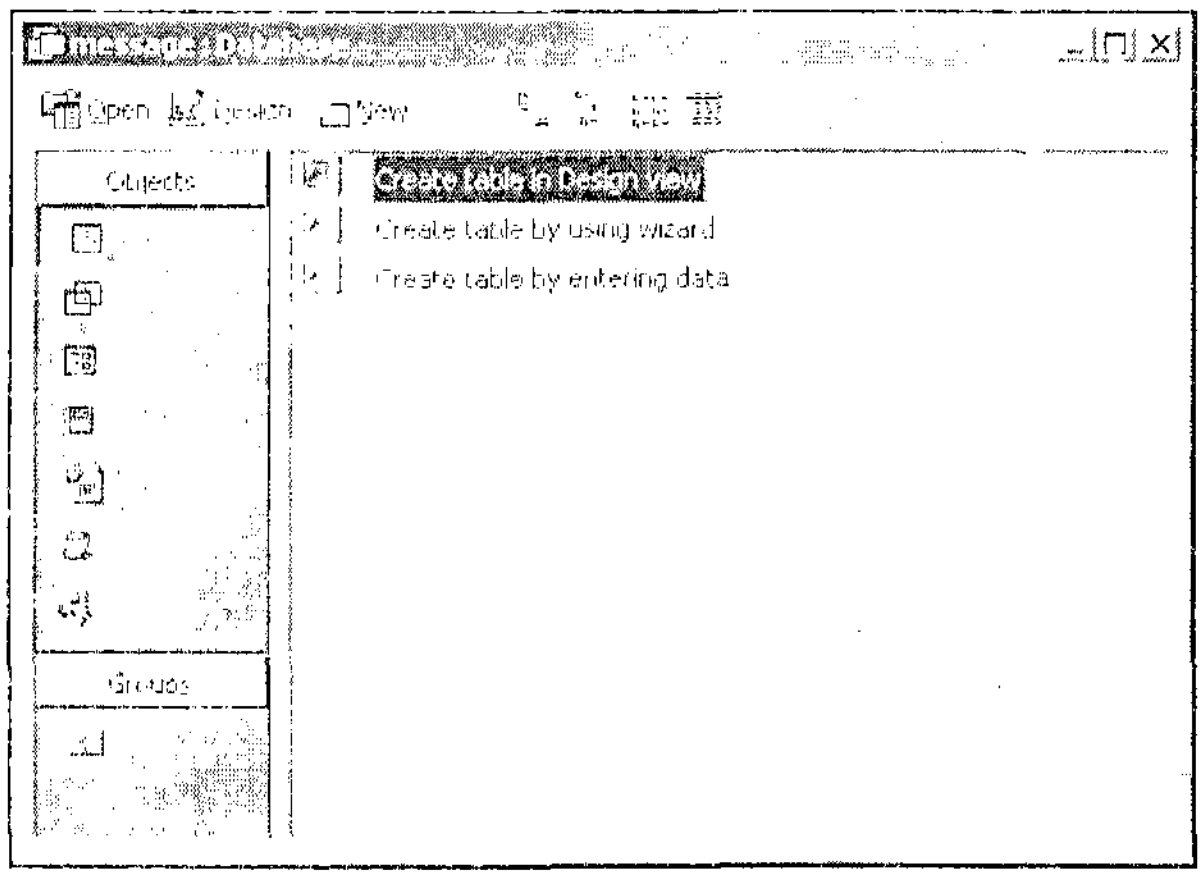


Рис. 19.2. Выбор режима конструктора в Microsoft Access

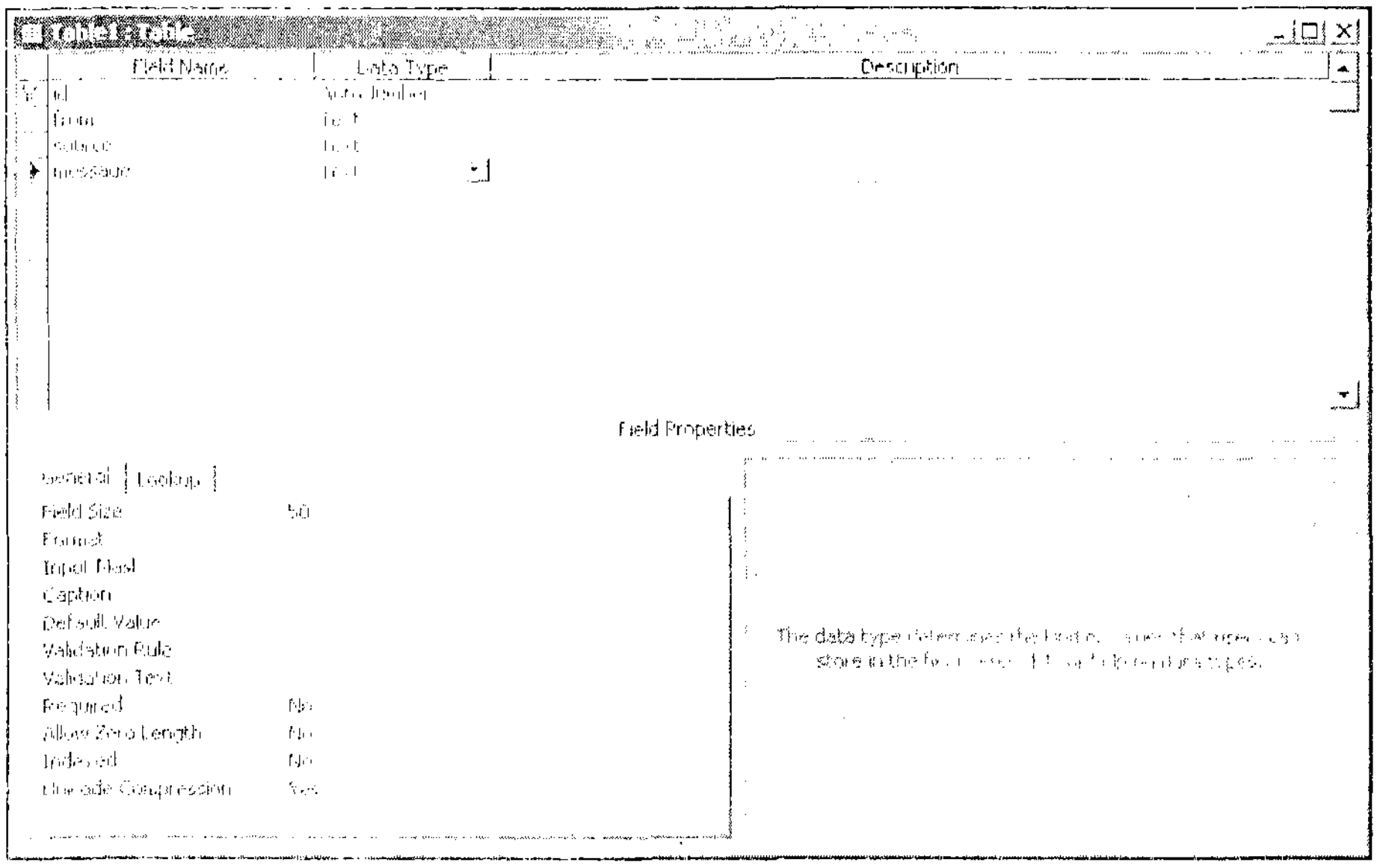


Рис. 19.3. Создание полей базы данных в режиме конструктора

Из выпадающего меню File (Файл) выберите команду Save As (Сохранить как), чтобы сохранить таблицу под именем msg. Из выпадающего меню View (Вид) выберите команду Datasheet View (Режим таблицы), чтобы приступить к вводу данных в базу данных. Нельзя вводить данные в ячейке id, поскольку функция AutoNumber (Счетчик) в Access выполняет все necessary действия за вас. Используйте клавишу <Tab> для перехода к ячейке from

и ввода имени. Продолжайте вводить данные в каждом из полей для создания нескольких записей. Вам необязательно использовать точно такие же данные, как показано на рис. 19.4; также не следует ограничиваться всего тремя записями.

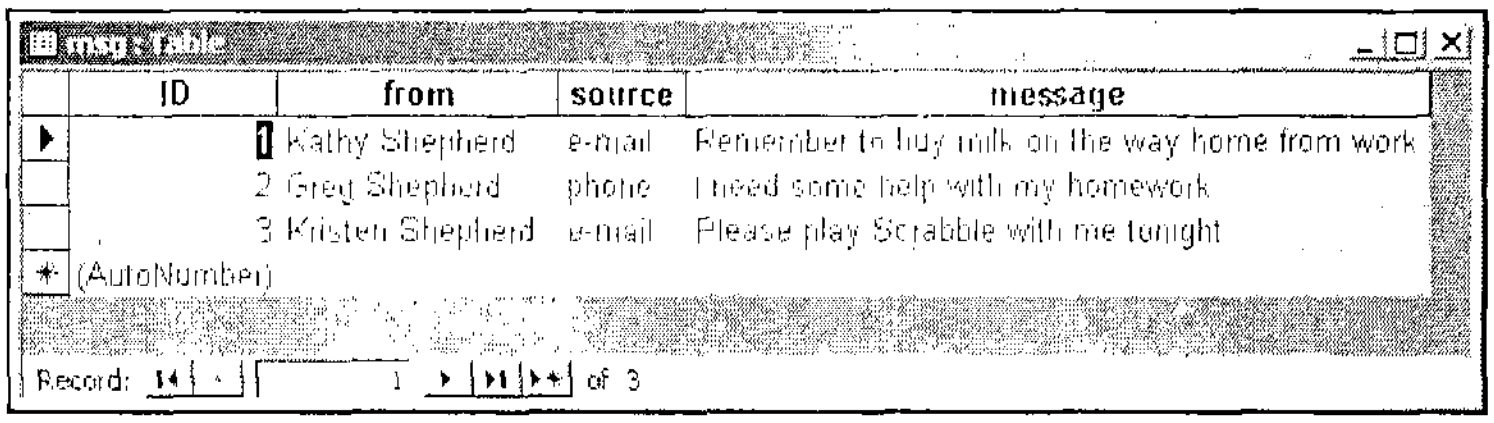


Рис. 19.4. Таблица msg теперь заполнена данными

Теперь можно закрыть базу данных. Вы создали базу данных с сообщениями, доступ к которой осуществляется с помощью сценариев. База данных содержит записи, соответствующие отдельным сообщениям или напоминаниям точно так же, как вы представляли их в виде XML в предыдущих уроках.

Как вы уже знаете, база данных обеспечивает удобные и эффективные средства хранения, индексирования, безопасности и ряда других технологий, на которые полагаются при сохранении важных данных. Почти все организации сохраняют свои данные в базах данных. Очень часто это относится к наследуемым данным, которые вам потребуются при создании динамического содержания для Web или для взаимодействия приложений. Например, вам понадобится переместить данные из базы данных в поток данных XML, который может использоваться другими приложениями или должным образом оформляться при передаче клиенту.

Определение имени источника данных ODBC

Средства доступа ODBC, точно так же как и JDBC, предназначены для обеспечения доступа к базам данных и реализуются с помощью специальных клиентских и серверных программных средств. Средство доступа ODBC предоставляет в ваше распоряжение прикладной интерфейс API, который обеспечивает извлечение данных из базы данных.

Вы будете использовать ODBC, создав именованный источник данных с помощью специального административного инструмента. Тогда вы обеспечите взаимодействие между приложением и базой данных, которое активизируется с помощью вызова методов соответствующих объектов сценариев. Имя источника данных DSN — это ссылка на драйвер базы данных (в нашем случае, базы данных Microsoft Access), который связан с определенной базой данных. Задав такой параметр, как DSN, вы укажете путь к базе данных в файловой системе операционной системы. Другими словами, когда вы создаете объект, взаимодействующий с DSN, этот объект будет использовать ODBC для доступа к базе данных, а тогда вы сможете использовать инструкции SQL при работе с данными. Объект базы данных обеспечивает соединение с базой данных посредством собственного набора методов. Соединение будет оставаться активным до тех пор, пока вы не закроете его с помощью метода `Close` соответствующего прикладного интерфейса API.

Для создания DSN вам необходимо воспользоваться административным инструментом Data Sources (ODBC) (Источники данных (ODBC)). В нашем упражнении предполагается, что вы используете операционную систему Windows 2000. Откройте окно Control Panel (Панель управления) Windows 2000, щелкнув на кнопке Start (Пуск) панели задач Windows 2000, после чего выберите команду Settings⇨Control Panel (Настройка⇨Панель управления). В окне Control Panel раскройте папку Administrative Tools (Инструменты администрирования), после чего запустите инструмент Data Sources (ODBC). После того как окно инструмента Data Sources (ODBC) появится на экране, перейдите на вкладку System DSN. Обязательно выберите именно вкладку System DSN, а не вкладку User DSN или File DSN.

СОВЕТ

Вы можете настроить System DSN или User DSN. Однако User DSN позволяет только создателю DSN получать доступ к базе данных через источник данных. Если необходимо предоставлять доступ к определенной базе данных нескольким пользователям компьютера, вам придется настроить несколько System DSN или User DSN. System DSN позволяет всем пользователям компьютера получать доступ к базе данных через указанный источник данных.

Вы увидите окно, в котором представлены все существующие DSN, если таковые существуют. Щелкните на кнопке Add (Добавить), чтобы открыть еще одно окно и увидеть список всех драйверов баз данных, установленных на вашем компьютере. Если Microsoft Access 2000 была должным образом установлена в составе пакета Microsoft Office 2000, вы должны увидеть нужный драйвер в представленном списке. Выберите вариант Microsoft Access Driver (*.mdb) и щелкните на кнопке Finish (Готово). На экране отобразится диалоговое окно ODBC Microsoft Access Setup. В текстовом поле Data Source Name (Имя источника данных) введите `messageDB`, уделив должное внимание регистру вводимых символов. Вам необходимо, чтобы имя DSN было идентично приведенному в наших примерах, чтобы сценарии, которые вы создадите при выполнении последующих шагов, работали должным образом. Щелкните на кнопке Select (Выбрать), чтобы воспользоваться диалоговым окном Select Database File (Выбор файла базы данных) для выбора файла `c:\day_19\message.mdb`. В этом момент на экране вашего монитора должны быть окна, приведенные на рис. 19.5.

Закройте все открытые диалоговые окна, щелкая на кнопках OK в каждом из них.

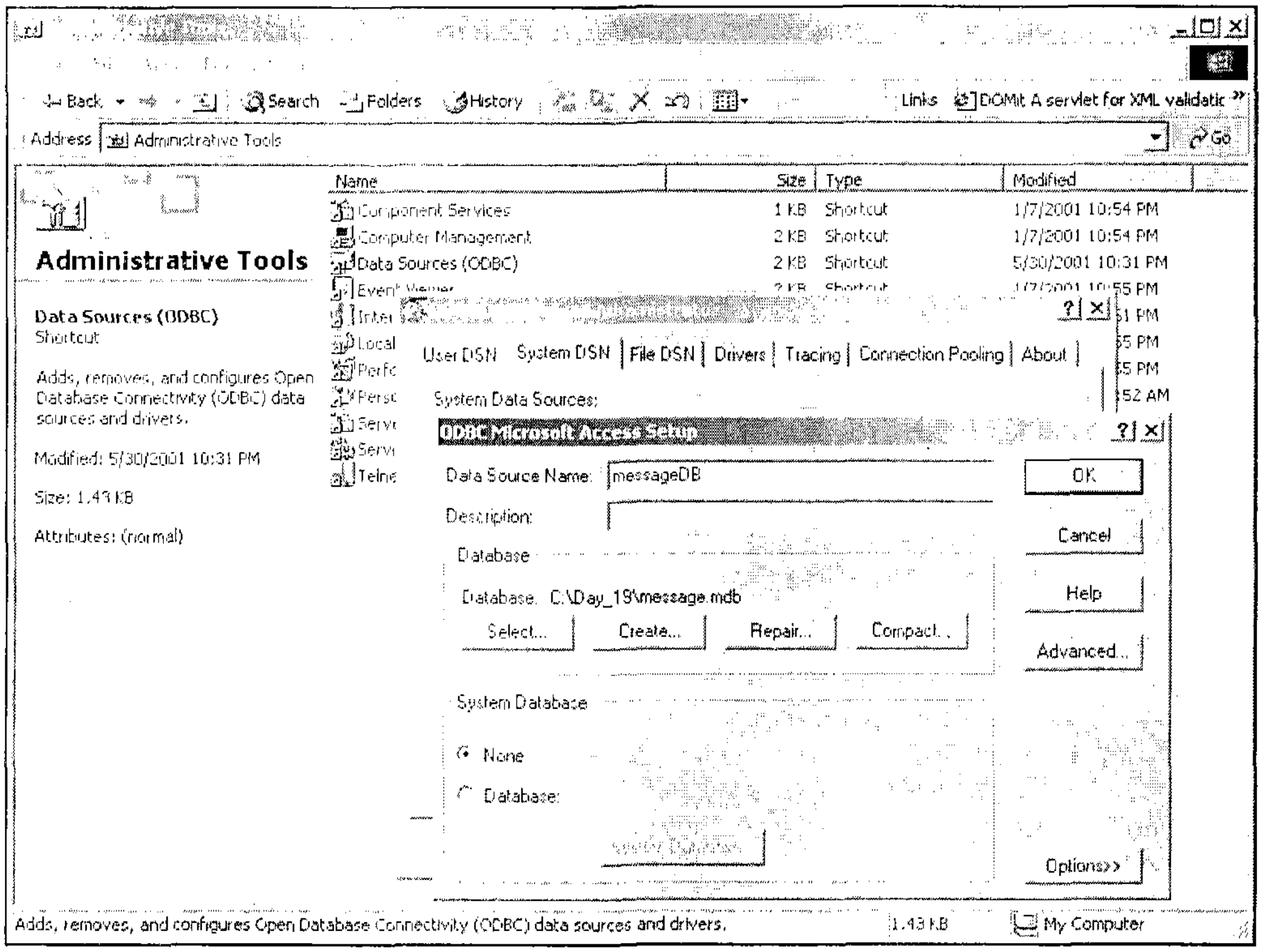


Рис. 19.5. Создание имени источника данных с помощью инструмента Data Sources (ODBC)

Настройка Web-сервера

В следующих упражнениях вы будете использовать в качестве Web-сервера, доступ к которому реализуется с помощью Web-броузера, средство IIS. IIS предоставляет замечательные возможности для проверки сценариев, которые должны выполняться на стороне сервера, на локальном компьютере. Очень часто программисты разрабатывают приложения на локальном компьютере, проверяют их с помощью IIS, после чего пересылают их на удаленный сервер в Web. Вы можете выполнить это, используя другие инструменты, однако IIS обеспечивает самый простой способ решения подобных задач.

Для того чтобы использовать Microsoft Internet Information Services (IIS) в качестве Web-сервера при выполнении последующих действий, вам необходимо зарегистрировать папку `c:\day_19` как каталог общего доступа в Web. Отобразив папку в окне программы Проводник, щелкните на ней правой кнопкой мыши и выберите команду Sharing (Доступ) из появившегося контекстного меню. Перейдите на вкладку Web Sharing, не выбрав по ошибке вкладку Sharing (Доступ). Установите переключатель Share this folder на вкладке Web Sharing. На экране отобразится еще одно диалоговое окно, Edit Alias. В этом окне вам не нужно изменять никаких заданных по умолчанию данных; их достаточно для того, чтобы вы смогли выполнить все оставшиеся в настоящем уроке упражнения. Обратите внимание, что система автоматически назначила имя `day_19` общему Web-ресурсу. Вы будете использовать эту часть при указании полного адреса URL в адресной строке броузера при получении доступа к приложениям, сохраненным в общем ресурсе на Web-сервере. Синтаксис команды, необходимой для загрузки программы, например HTML-страницы или сценария, сохраненной в общем ресурсе на Web-сервере, приведен ниже:

```
http://localhost/day_19/myfile.asp
```

Часть `localhost` этого адреса URL заменяет стандартное выражение `www.domain.ext`, которое вы указываете в адресной строке броузера при просмотре Web-узлов в Internet с использованием протокола HTTP. Адрес `localhost` указывает на локальный Web-сервер, поддерживаемый IIS, а такая часть, как `day_19`, указывает на общий Web-ресурс, созданный вами сегодня, *`myfile.asp`* — это имя файла документа, HTML-приложения или сценария ASP, который вы хотите запустите на сервере.

Доступ к базе данных с помощью сценария на стороне сервера

В следующем упражнении вы создадите сценарий ASP, который будет генерировать код HTML-страницы, предназначенной для отображения данных, полученных с помощью инструкций SQL. Другими словами, теперь, когда у вас есть база данных, доступ к которой осуществляется с помощью ODBC API, вы можете создать объект соединения, позволяющий вам передавать инструкции SQL базе данных, которая будет возвращать результаты сценарию, выполняемому на стороне сервера. Заключая полученные в результате обработки запроса данные в инструкции HTML, вы можете отобразить результаты (например, записи базы данных) в окне броузера. Например, предположим, вы хотите использовать сценарий для отправки запроса базе данных, а также для представления всех полученных полей каждой записи в виде маркированного списка. Цель этого упражнения — получить результат, подобный тому, который приведен на рис. 19.6.

Вы можете запустить свой любимый текстовый редактор и приступить к созданию сценария, изучая материал этого урока. Сценарий ASP содержит элементы как VBScript, так и HTML. Инструкции VBScript спокойно воспринимаются Web-броузерами. Вы приступите к созданию сценария ASP с инструкции, которая определяет используемый язык сценариев. Синтаксис инструкции приведен ниже:

```
<%@ LANGUAGE = "VBScript"%>
```

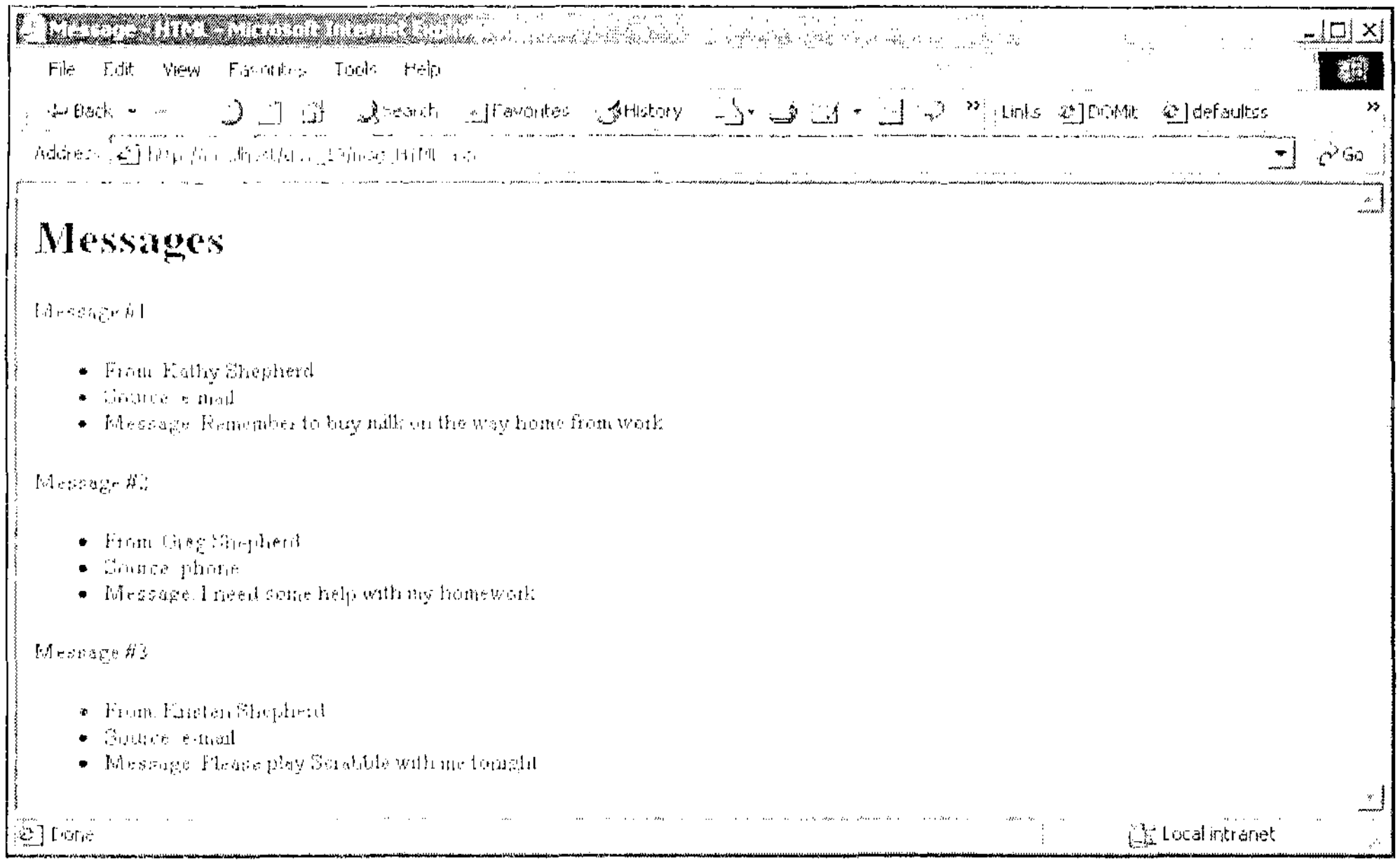


Рис. 19.6. Результат, представленный в виде HTML-страницы и полученный с помощью сценария ASP, обращающегося к базе данных

После этого вы можете создать начало HTML-страницы, включив в нее такие разделы, как <!DOCTYPE...>, <HTML>, <HEAD> и <BODY>. Эти разделы выполняют точно такие же функции, как и на любой другой HTML-странице. Однако в раздел <BODY> вам придется включить логику, необходимую для открытия соединения ODBC, передачи инструкций SQL базе данных, а также представления полученных результатов в виде инструкций HTML.

Для подключения к базе данных, которую вы указали раньше, создайте объект сервера, который инициализирует объект данных ActiveX (ADO) `Connection`, а также метод `open`, которому в качестве аргумента вы передаете имя источника данных DSN. Синтаксис вызова объекта и связанного с ним метода приведен ниже:

```
set oMyObject = Server.CreateObject("ADODB.Connection")
      oMsg.open("myDSN")
```

В частности, вы введете следующий код:

```
set oMsg = Server.CreateObject("ADODB.Connection")
      oMsg.open("messageDB")
```

Новый объект называется `oMsg`, а DSN `messageDB` передается в качестве аргумента метода `open`. В результате создается соединение между сценарием ASP и вашей базой данных `message.mdb`. Это соединение остается открытым до тех пор, пока вы не закроете его, воспользовавшись свойством `Close` объекта `Connection`:

```
oMsg.Close
```

Свойство `Close` указывается после любой другой логики, которую вы захотите выполнить при открытом соединении ODBC, например ряд инструкций SQL и форматирование полученных результатов. Для выполнения инструкций SQL вы используете метод `Execute` объекта `Connection`. Получается объект `RecordSet`, который возвращается методом `Execute`. Синтаксис инструкции, передающей инструкции SQL базе данных, приведен ниже:

```
set myRecordSetVariable = oMsg.Execute("Инструкция SQL")
```

Для выбора всех записей (*) в таблице msg вашей базы данных message.mdb можно использовать достаточно простую инструкцию SQL, например, следующую:

```
SELECT * FROM msg;
```

Если вы присвоите объекту RecordSet имя msgRS, полный вызов объекта может иметь следующий вид:

```
set msgRS = oMsg.Execute("SELECT * FROM msg;")
```

Если вы хотите возвратить только первую запись из базы данных, можно вывести на стандартное устройство вывода или экран все необходимые поля. Однако, поскольку в подобном случае вам потребуются сведения обо всех записях в базе данных, вам необходимо средство, которое будет перебирать все записи и возвращать сведения о них до тех пор, пока не достигнут конец базы данных. Проще всего создать цикл, который позволит считывать сведения о записи, формировать результат, после чего переходить к следующей записи для дальнейшей обработки. Для этого замечательно подойдут циклы VBScript. Начните с инструкции, которая прикажет интерпретатору выполнять остальные инструкции до тех пор, пока не будет достигнут конец базы данных, на что укажет маркер EOF (End-Of-File — конец файла). В рамках цикла укажите поля, сведения о которых должны сообщаться для каждой записи, после чего будет переход к следующей записи. Синтаксис цикла приведен ниже:

```
do until myRecordSetVariable.EOF
        Response.Write (myRecordSetVariable("field_name"))
    ...
    myRecordSetVariable.MoveNext
Loop
```

В нашем примере вам потребуется следующий цикл:

```
do until msgRS.EOF
            Response.Write ("<P>Message #" & _
                msgRS("ID") & "<UL>")
            Response.Write ( _
                "<LI>From: " & msgRS("from") & "</LI>" & _
                "<LI>Source: " & msgRS("source") & "</LI>"& _
                                    "<LI>Message: " & msgRS("message") &
"</LI>" )
            Response.Write ("</UL>")
            msgRS.MoveNext
Loop
```

После выполнения этого цикла вы закроете соединение с помощью свойства oMsg.Close, как уже отмечалось раньше. После этого вы можете удалить объект, присвоив переменной oMsg значение null, воспользовавшись следующей инструкцией:

```
set oMsg = nothing
```

Все, что вам остается сделать, — это закрыть элементы BODY и HTML. Полный вариант приложения приведен в листинге 19.1.

Листинг 19.1. Сценарий ASP для извлечения записей из базы данных msg_HTML.asp

```
1: <%@ LANGUAGE = "VBScript"%>
2:  <%' листинг 19.1 - msg_HTML.asp
3:  %>
4:
5:  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
6:   "http://www.w3.org/TR/html4/loose.dtd">
7:  <!-- HTML file generated by ASP -->
8:
9:  <HTML>
```

```
10:      <HEAD>
11:          <TITLE>Message - HTML</TITLE>
12:      </HEAD>
13:      <BODY>
14:          <H1>Messages</H1>
15:          <%
16:          set oMsg = Server.CreateObject("ADODB.Connection")
17:          oMsg.open("messageDB")
18:
19:          set msgRS = oMsg.Execute("SELECT * FROM msg;")
20:
21:          do until msgRS.EOF
22:              Response.Write ("<P>Message #" & _
23:                  msgRS("ID") & "<UL>")
24:              Response.Write (_
25:                  "<LI>From: " & msgRS("from") & "</LI>" &
26:                  "<LI>Source: " & msgRS("source") & "</LI>"& _
27:                                  "<LI>Message: " & msgRS("message") &
                                    "</LI>" )
28:              Response.Write ("</UL>")
29:              msgRS.MoveNext
30:          Loop
31:          oMsg.Close
32:          set oMsg = nothing
33:          %>
34:      </BODY>
35: </HTML>
```



В строке 1 указывается, что для создания сценария Active Server Pages используется язык VBScript. В строках 5 и 6 содержится объявление `DOCTYPE` для HTML-страницы, которая получается в результате выполнения сценария. Строки 9–14 содержат типичные дескрипторы HTML для новой страницы. Строки 15–33 содержат сам сценарий. Объект ADO создается в строке 16, и ему присваивается имя `oMsg`. Объект `Connection` создается со значением DSN, равным `messageDB`, с помощью метода `open` (строка 17). В строке 19 передается запрос SQL базе данных с помощью метода `Execute` объекта `Connection`. Строки 21–30 содержат цикл, который выводит результаты выполнения запроса на экран. Цикл выполняется до тех пор, пока не будет достигнет конец файла базы данных (строка 21). Каждое из полей первой записи передается на стандартное устройство вывода (`Response.Write`) и заключается в соответствующие дескрипторы HTML. В строке 29 свойство `MoveNext` объекта `RecordSet` вызывает переход к следующей записи в базе данных. Строка 30 указывает сценарию на необходимость перехода к строке 21. Если последняя запись еще не была обработана, цикл выполняется до тех пор, пока этого не произойдет. Строка 31 приводит к закрытию соединения, а в строке 32 переменной `oMsg` присваивается значение `null`. Сценарий завершается в строке 33, а в оставшихся двух строках закрываются дескрипторы HTML.

Для выполнения сценария и получения результатов, представленных на рис. 19.1, вам необходимо загрузить в броузер соответствующий сценарий ASP. Для того чтобы это сделать, введите в адресной строке Microsoft Internet Explorer следующее выражение:

```
http://localhost/day_19/msg_HTML.asp
```

Создание документа XML с помощью сценария

В предыдущем примере мы получили результат в формате HTML для отображения в окне броузера. Внимательно изучив приведенный код, вы увидите, что результат представляет собой правильный документ XML, но выраженный с помощью словаря HTML. Внеся незначительные изменения в сценарий, вы можете заставить его представлять результат в формате XML, что позволит ему выглядеть так же, как примеры из предыдущих уроков.

В следующем примере вы будете использовать те же вызовы методов объектов и ту же самую инструкцию SQL, но полученные данные будут заключаться в дескрипторы XML. Другими словами, вместо разделов <!DOCUMENT...>, <HTML>, <HEAD> и <BODY> вы будете использовать стандартное объявление XML и необходимый корневой элемент:

```
<?xml version="1.0"?>
<note>
    ...код VBScript
</note>
```

Код цикла будет немного отличаться, поскольку вместо дескрипторов HTML используются дескрипторы XML. Теперь цикл выглядит следующим образом:

```
do until msgRS.EOF
        Response.Write ("<msg id='" & _
           msgRS("ID") & "'>")
        Response.Write (_
            "<from>" & msgRS("from") & "</from>" & _
            "<source>" & msgRS("source") & "</source>" & _
                         "<message>" & msgRS("message") & "</message>"
)
        Response.Write ("</msg>")
        msgRS.MoveNext
Loop
```

Обратите внимание, что элементы списка HTML теперь заменены уже привычными элементами XML `from`, `source` и `message`. Также обратите внимание, что значение поля `id` базы данных используется в качестве значения атрибута в результирующем документе XML.

Полный код программы приведен в листинге 19.2. Внесите необходимые изменения в первый сценарий и сохраните его под именем `msg_XML.asp`.

Листинг 19.2. Сценарий ASP для извлечения записей из базы данных и представления в виде правильного XML msg_XML.asp

```
1: <%@ LANGUAGE = "VBScript"%>
2:  <%' листинг 19.2 - msg_XML.asp
3:  %>
4:
5:  <%Response.ContentType="text/xml"%>
6:  <?xml version="1.0"?>
7:  <!-- XML file generated by ASP -->
8:
9:  <note>
10:      <%
11:      set oMsg = Server.CreateObject("ADODB.Connection")
12:      oMsg.open("messageDB")
13:
14:      set msgRS = oMsg.Execute("SELECT * FROM msg;")
15:
16:      do until msgRS.EOF
```

```
17:          Response.Write ("<msg id='" & _
18:             msgRS("ID") & "'>")
19:          Response.Write ( _
20:              "<from>" & msgRS("from") & "</from>" & _
21:              "<source>" & msgRS("source") & "</source>" & _
22:                      "<message>" & msgRS("message") & "</message>"
)
23:          Response.Write ("</msg>")
24:          msgRS.MoveNext
25:      Loop
26:      oMsg.Close
27:      set oMsg = nothing
28:      %>
29: </note>
```

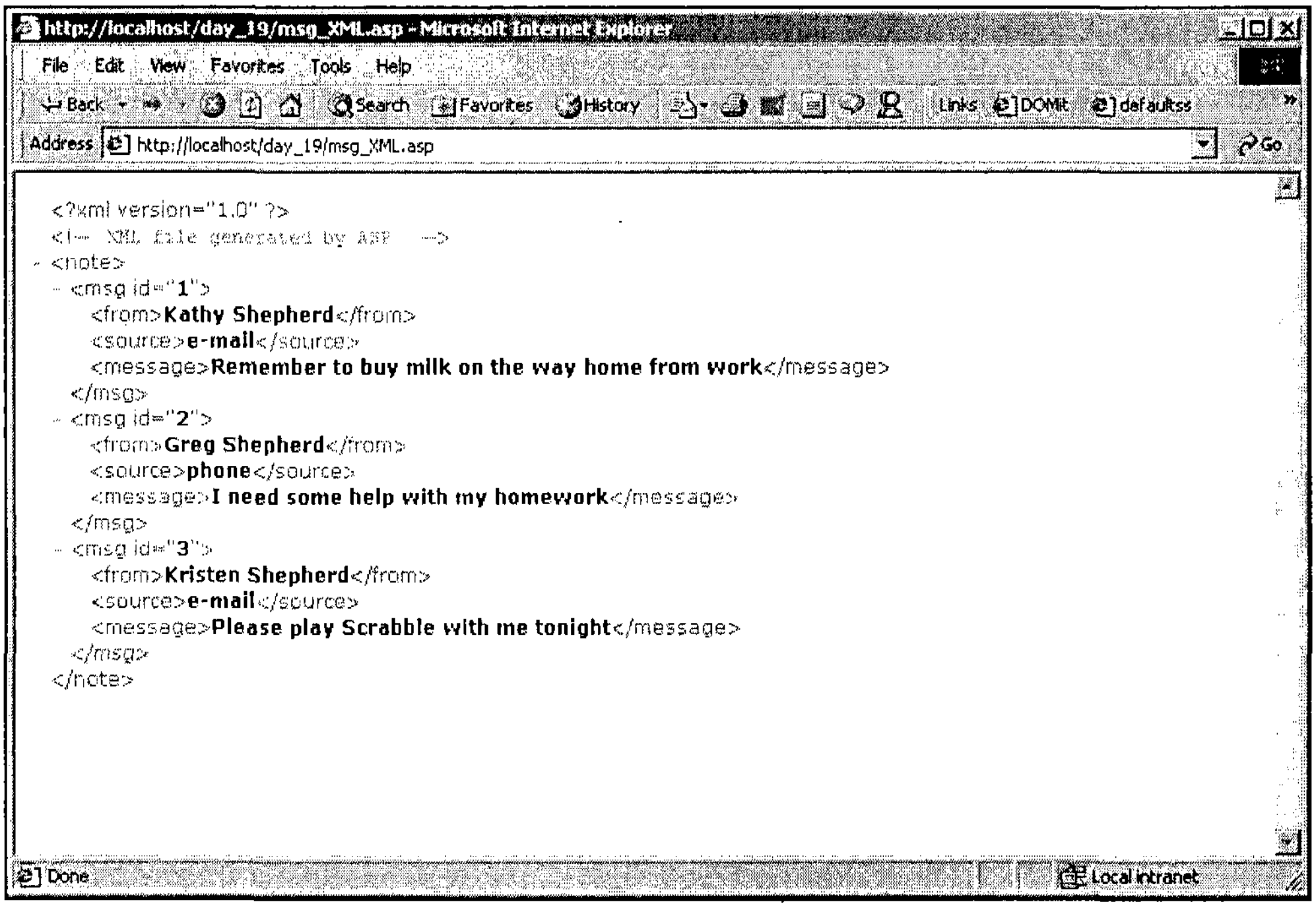


Рис. 19.7. XML, полученный после выполнения сценария, обращающегося к базе данных



В строке 1 указывается, что для создания сценария Active Server Pages используется язык VBScript. В строке 6 содержится объявление документа XML, который получается в результате выполнения сценария. В строке 9 содержится открывающий дескриптор корневого элемента `note`. Строки 10–28 содержат сам сценарий. Объект ADO создается в строке 11, и ему присваивается имя `oMsg`. Объект `Connection` создается со значением DSN, равным `messageDB`, с помощью метода `open` (строка 12). В строке 14 передается запрос SQL базе данных с помощью метода `Execute` объекта `Connection`. Строки 16–25 содержат цикл, который выводит результаты выполнения запроса на экран. Цикл выполняется до тех пор, пока не будет достигнут конец файла базы данных (строка 16). Каждое из полей первой записи передается на стандартное устройство вывода (`Response.Write`) и

заключается в соответствующие дескрипторы XML (строки 17–23). В строке 24 свойство `MoveNext` объекта `RecordSet` вызывает переход к следующей записи в базе данных. Строка 25 указывает сценарию на необходимость перехода к строке 16. Если последняя запись еще не была обработана, цикл выполняется до тех пор, пока этого не произойдет. Строка 26 приводит к закрытию соединения, а в строке 32 переменной `oMsg` присваивается значение `null`. Сценарий завершается в строке 28, а в строке 29 содержится закрывающий дескриптор корневого элемента `note`.

Вы можете выполнить этот сценарий, введя в адресной строке Internet Explorer следующий адрес URL:

```
http://localhost/day_19/msg_XML.asp
```

Выполнение сценария должно привести к результату, приведенному на рис. 19.7.

Дальнейшая обработка

Предыдущий пример позволил получить результат в формате XML, но при этом в окне броузера данные, полученные из базы данных, не были представлены. Вы можете создать таблицу стилей и связать ее с документом, чтобы обеспечить стилевую разметку документа. Однако, вместо этого, вы создадите HTML-страницу, которая определяет остров данных XML и свяжет элементы HTML с соответствующими элементами данных XML. Вы уже использовали подобный прием в уроке дня 17-го. Различие состоит в том, что вместо получения доступа к статическому документу XML, как вы делали раньше, ваш остров данных будет вызывать сценарий ASP `msg_XML.asp` для получения того, что называется *виртуальным* XML. В данном случае данные XML оказываются временными; они хранятся в памяти и никогда не сохраняются на диске.

Если вы сохранили копию страницы `island02.html`, созданной в уроке дня 17-го, вы можете немного изменить ее сейчас. Она очень похожа на ту страницу, которая потребуется вам при выполнении упражнения. Основное отличие состоит в том, что остров данных в текущем документе создается после выполнения сценария ASP, а не получается из статического документа XML. В нашем упражнении дескриптор HTML `<XML>` будет выглядеть следующим образом:

```
<XML SRC="msg_XML.asp" ID="myMsg"/>
```

Обратите внимание, что источник острова данных — сценарий ASP. Поэтому вы сможете только выполнить страницу, обработав ее с помощью Web-сервера IIS. Другими словами, вы не сможете просмотреть ее с помощью броузера, как обычные страницы. Для выполнения кода ASP его нужно сначала обработать с помощью Web-сервера.

Оставшаяся часть страницы содержит стандартные элементы разметки HTML, а также данные, необходимые для создания таблицы. Код для создания таблицы выглядит так:

```
<TABLE id="table" border="6" width="100%"
        datasrc="#myMsg" summary="messages">
        <THEAD style="background-color: aqua">
            <TH>From</TH>
            <TH>Source</TH>
            <TH>Message</TH>
        </THEAD>
        <TR valign="top" align="center">
            <TD><SPAN DATAFLD="from"/></TD>
            <TD><SPAN DATAFLD="source"/></TD>
            <TD><SPAN DATAFLD="message"/></TD>
        </TR>
</TABLE>
```

Значение атрибута источника данных в дескрипторе таблицы соответствует значению атрибута id, которое указано в дескрипторах острова данных (<XML SRC="msg_XML.asp" ID="myMsg"/>). Каждая из ячеек <TD> содержит элемент SPAN, связанный с одним из элементов from, source или message в экземпляре документа XML. Полный код программы приведен в листинге 19.3.

Листинг 19.3. Область данных ASP, полученная с помощью XML Data из базы данных, ASP_isle.html

```
1: <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
2:   "http://www.w3.org/TR/html4/loose.dtd">
3:  <!-- листинг 19.3 - ASP_isle.html -->
4:
5:  <HTML>
6:      <HEAD>
7:          <TITLE>XML Data Binding</TITLE>
8:      </HEAD>
9:      <BODY>
10:          <H1>Messages</H1>
11:    <XML SRC="msg_XML.asp" ID="myMsg"/>
12:    <TABLE id="table" border="6" width="100%"
13:           datasrc="#myMsg" summary="messages">
14:         <THEAD style="background-color: aqua">
15:                <TH>From</TH>
16:                <TH>Source</TH>
17:                <TH>Message</TH>
18:         </THEAD>
19:         <TR valign="top" align="center">
20:                <TD><SPAN DATAFLD="from"/></TD>
21:                <TD><SPAN DATAFLD="source"/></TD>
22:                <TD><SPAN DATAFLD="message"/></TD>
23:         </TR>
24:          </TABLE>
25:      </BODY>
26: </HTML>
```



В строке 11 определяется остров данных, получаемый после выполнения сценария msg_XML.asp. Атрибут ID="myMsg" содержит ссылку, которая используется в строке 13 для указания на источник данных для таблицы. Соответствующий документ XML указан с помощью значения атрибута src. Открывающий дескриптор таблицы (строки 12 и 13) содержит атрибут datasrc, указывающий на ссылку #myMsg острова данных. Раздел заголовка таблицы (строки 14—18) добавлен для того, чтобы на странице отображались заголовки столбцов таблицы. Элемент TR определен в строках 19 —23. В строках 20—22 используются элементы SPAN для связывания с элементами XML from, source и message соответственно.

Выполните эту страницу, указав следующий адрес URL:

http://localhost/day_19/ASP_isle.html

Результат должен выглядеть так, как показано на рис. 19.8.

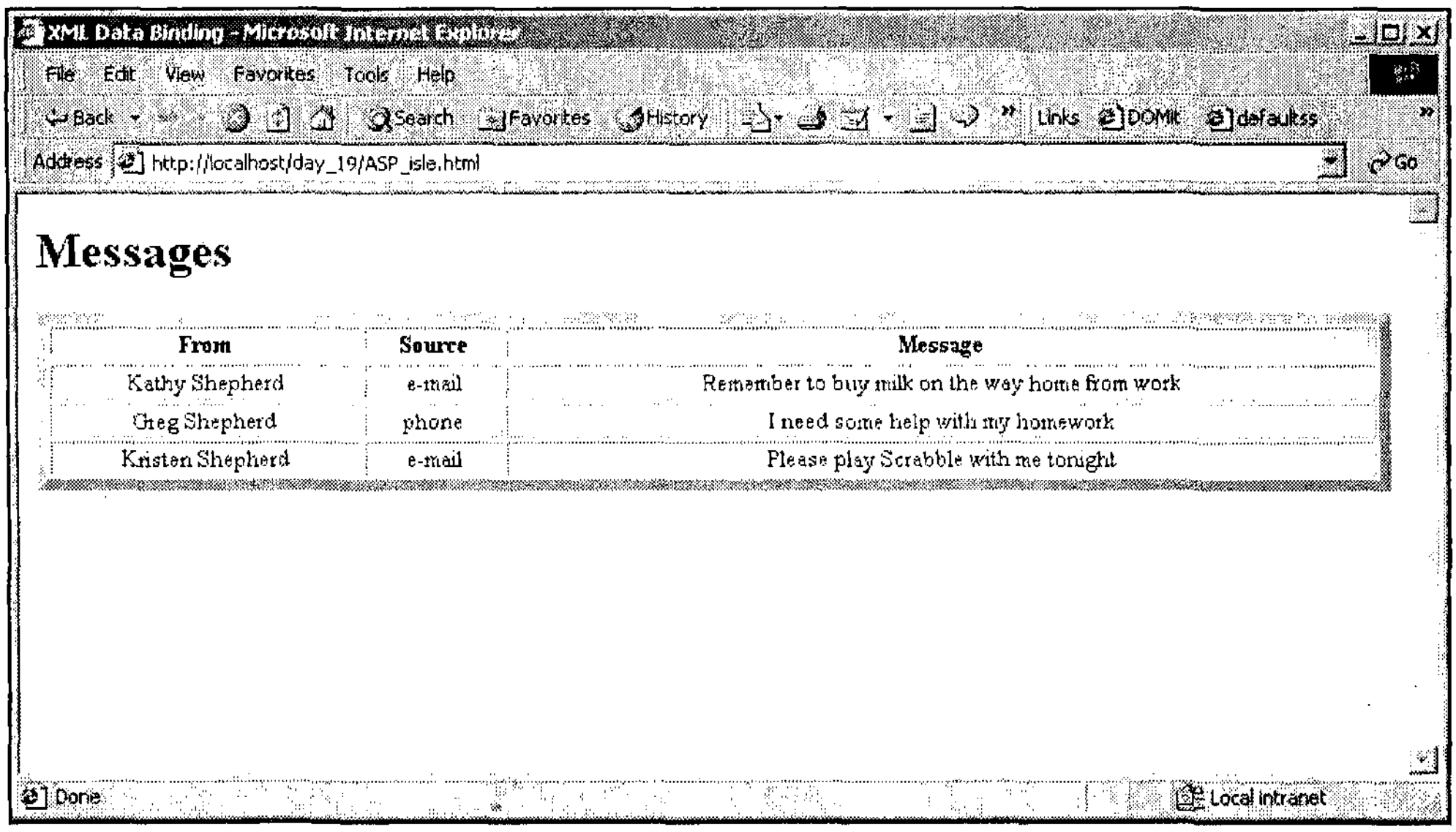


Рис. 19.8. Результат выполнения Web-приложения с использованием сценария на промежуточном уровне

Резюме

Сегодня вы познакомились с интеграцией XML в бизнес-модели при использовании в корпоративных приложениях. Основная часть рассмотренных нами моделей каким-нибудь образом связана с базами данных. В трехуровневой архитектуре Web-приложений базы данных находятся на уровне данных. Сценарии, которые выполняются на промежуточном уровне, получают доступ к данным, сохраненным в базе данных, обрабатывают их, после чего передают клиенту. Затем происходит дальнейшая обработка данных для представления результатов пользователю. Вы использовали целый ряд технологий для выполнения таких задач, как интеграция данных и приложений, но в их основе лежат возможности XML по обмену данными.

Вопросы и ответы

Что такое анализ документа?

Анализ документа — это процесс исследования частей, назначения и требований для одного или нескольких классов документов при проектировании подробного описания каждого из них.

Что такое трехуровневая архитектура Web?

Это специальный вид архитектуры "клиент/сервер", состоящий из трех четко определенных процессов, каждый из которых выполняется на отдельной платформе. Броузеры и другие агенты пользователя, которые функционируют на уровне клиента, получают доступ к приложениям через Web-сервер, который функционирует на промежуточном уровне. Данные сохраняются на уровне данных, как правило, в базах данных, файловых системах или других структурах.

День 20-й

Электронная коммерция и XML

Электронный бизнес открывает огромные возможности перед компаниями, которые стремятся улучшить управление операциями, автоматизируя обмен данными в B2C (business-to-consumer) и B2B (business-to-business). Целый ряд протоколов разработан для совместного использования данных, удаленного вызова объектов и обмена данных с использованием XML, но все они еще далеки от завершения. Окончательные стандарты до сих пор не утверждены, однако значительная часть разработки основных методологий на настоящий момент уже выполнена.

Сегодня мы рассмотрим следующие темы.

- Преимущества приложений электронного бизнеса на основе XML.
- Использование XML в приложениях B2C (business-to-consumer) и B2B (business-to-business).
- Протоколы XML для обмена структурированными сообщениями.

Применение XML в бизнесе

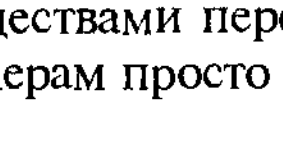
Вчера вы познакомились с приложениями, чрезвычайно полезными при использовании в корпоративной среде. Каждый из рассмотренных сценариев описывал систему, которая выигрывала от применения XML, что приводило к выигрышу для предприятий. Сейчас вы познакомитесь с преимуществами использования технологий XML компаниями, которые продают товары и предоставляют услуги через Web. В наше время приставка *e-* (e-business — электронный бизнес; e-commerce — электронная коммерция) еще очень многих вводит в замешательство. На самом деле, сложного в этом ничего нет. Представьте себе, что e-business или e-commerce — это всего лишь обычная финансовая транзакция между компанией и ее клиентом (B2C, business-to-consumer) или двумя компаниями (B2B, business-to-business). Существуют и отношения, называемые B2G (business-to-government), а также некоторые другие. Сегодня основное внимание мы уделим использованию именно B2C-приложений, а после этого мы рассмотрим несколько технологий, ключевых для рынка B2B.

Преимущества ведения электронного бизнеса

В соответствии с данными компании *IBM* (источник: `http://www-7.ibm.com/nz/e-business/overview.html`) в 2001 году потребители потратили свыше 130 мил-

лиардов долларов при выполнении транзакций в режиме online. Другие компании предсказывают еще больший рост в секторе B2B. К 2003 году рынок электронной коммерции в сфере B2B должен вырасти до 1,3 триллиона долларов, в десять раз превысив рынок электронной коммерции в сфере B2C. Это около 9% общего уровня доходов в США, превышает валовый национальный доход Великобритании, в два раза превышает валовый национальный доход Канады и составляет одну треть валового национального дохода Японии (источник: `http://www.oecd.org/std/gdp.htm`). В 2004 году рынок электронной коммерции должен вырасти до 7,3 триллионов долларов, о чем сообщила известная аналитическая компания *Garthner Group* (`http://content.techweb.com/wire/story/TWB20000217S0002`). Для того чтобы все эти колоссальные капиталовложения оправдались, требуются просто совершенные технические решения. Поэтому не удивительно, что при построении подобных решений XML играет чрезвычайно важную роль.

Электронная коммерция и электронный бизнес больше не считаются технологиями будущего. Многие компании вкладывают немало средств в тесную интеграцию своих деловых операций и Web уже *сейчас*. Продажи через Web начались с рядовых потребительских товаров и очень быстро распространились на управление цепями поставки, электронную торговлю и предоставление всевозможных услуг. Все эти течения приводят к дальнейшему развитию сетевых и других технологий, стимулируют развитие экономик, причем даже в недостаточно развитых странах. Согласно исследованиям, проведенным компанией *PricewaterhouseCoopers* (PWC) (`http://www.pwcmoneytree.com/`), те компании, которые используют технологии электронного бизнеса, обладают значительными преимуществами перед своими конкурентами, поэтому как современным, так и будущим менеджерам просто необходимо знать, каким образом они добьются подобных результатов.



Электронный бизнес — это интеграция основных бизнес-систем с Web-технологиями. Говоря простым языком, электронный бизнес означаете бизнес с использованием сетей intranet, extranet или Internet. Электронная коммерция — это купля-продажа через Web.

Основной компонент любой стратегии электронного бизнеса — интеграция цепочки поставки: B2B (business-to-business). Скоро вы познакомитесь с этим подробнее, рассмотрев несколько базирующихся на XML технических решений, которые представляют ключевые платформы для сферы мелкого и среднего бизнеса. B2B — это тот сегмент рынка, на котором влияние электронного бизнеса на протяжении ближайших нескольких лет окажется наиболее значительным.

B2C-приложения

Модель B2B включает в себя автоматическое выполнение связанных с транзакциями приложений через Internet. Многие из таких приложений замечательно подходят для использования технологий XML, которые обеспечивают дополнительные функции и улучшенную производительность. В следующем разделе вы познакомитесь с двумя примерами того, как технологии XML используются в подобных приложениях. Продолжая подход, предложенный в прошлом уроке, я расскажу вам о следующем.

- Краткий обзор проблем, которые требуют решения.
- Потенциальные преимущества использования XML.
- Проблемы, связанные с разработкой решений.

Приложения для персонализации в Internet

Возможно, вы посещали Web-портал, на котором предоставляли такие сведения о себе, как адрес, почтовый индекс и другую информацию, чтобы получать прогноз погоды именно для местности, где вы проживает. Получая подобные сведения, вы, помимо всего прочего, получаете еще и сведения о предложениях местных магазинов, и рекламу. На самом деле, получение пресловутых рекламных сообщений как раз и оказывается платой за получение "бесплатного" прогноза погоды. В конце концов, какой смысл поставщику сведений для портала снабжать вам необходимыми сведениями совершенно бесплатно, не получая ничего взамен? Как правило, вам не приходится платить за просмотр таких сведений, однако рекламодатель может получить необходимую информацию, определив, на каких именно ссылках вы щелкали.

Подобная персонализация — это только начало того, что можно совершать в Web, используя XML совместно с другими технологиями. Например, целый ряд электронных книжных магазинов предлагает персонофицированные списки рекомендованных книг, базирующиеся на том, какие книги вы приобретали до этого. Для определения ваших предпочтений используются cookie-файлы, которые сохраняются на жестком диске вашего компьютера. При правильном подходе к его созданию cookie-файл — это просто сведения, которые помещают в базу данных на сервере. Это могут быть такие сведения, как ваше имя, перечень совершенных вами раньше покупок, а также частота ваших посещений. Сохраненный в виде документа XML соответствующий профиль может выглядеть так, как показано в листинге 20.1.

Листинг 20.1. Разметка персонализированного профиля `profile.xml`

```
<?xml version="1.0"?>
 <!-- figure 20.1 - profile.xml -->

 <profile>
  <cookie id="6233265454"/>
  <first_name>Devan</first_name>
  <last_name>Shepherd</last_name>
  <last_purchase date="07-01-01" frequency="6"/>
  <interests>
      <category>technical</category>
      <sub_category>computer</sub_category>
      <topic>XML</topic>
      <topic>Web Development</topic>
      <topic>C#</topic>
      <topic>E-Commerce</topic>
 </interests>
 </profile>
```

Предприятия, которые занимаются иной деятельностью, также могут представлять другие виды персонофицированных данных. Именно такой подход стал популярным при компоновке рекламного материала, распространяемого по электронной почте различными информационнымй агентствами. Однако нельзя забывать и о том, что настройка содержания под отдельных пользователей приведет к увеличению потока данных.

Возможности персонализации, базирующиеся на HTML, достаточно ограничены и хорошо подходят для представления статических страниц или динамического содержания, которое формируется в соответствии со строго определенной моделью. По мере усложнения управления узлом возрастает и сложность обеспечения доставки сведений с использованием HTML-инструментов.

Преимущества XML

XML обеспечивает более четкую интеграцию различного содержания и повышенную гибкость его доставки индивидуальным пользователям. Это связано с наследуемыми возможностями XML при представлении текстовых данных в структурированном виде. При необходимости можно получить отдельные структуры. Как правило, эти структуры базируются на наследуемых источниках данных с использованием технологий, подобных представленным в предыдущем уроке.

Вопросы, связанные с архитектурой и разработкой

Как правило, Web-узел, который использует настраиваемую доставку содержания, идентифицирует посетителя и передает результаты в базу данных. База данных предоставляет свои записи, затем их анализирует средство настройки, которое определяет шаблоны и выбирает сведения, чтобы представить их посетителю. Средство настройки может выбрать схему для определения структур данных и связывания таблицы стилей XSL для преобразования данных в соответствии с поставленными требованиям к их представлению. Схема помогает определить, какие именно элементы данных нужно получить из репозитария. Стили XSL позволяют преобразовать результирующие данные в соответствии с потребностями пользователя. Эти шаги могут быть динамическими, поэтому каждое представление данных уникально. Таким образом, данные, собранные в соответствии с предпочтениями отдельного пользователя, доставляются при каждом запуске приложения.

Проектирование профиля для подобных систем — это одно из самых важных требований, обеспечивающих успех всей операции. Анализ требований пользователя по отношению к содержанию позволяет получить наибольшую отдачу от усилий, направленных на сохранение и доставку информации.

Подобные порталы способны сохранять коллекции профилей, которые используются и другими приложениями, например системами доставки данных в сфере B2C, которые включают компоненты, характеризующие транзакции B2B. Чем большими сведениями вы будете располагать о посетителе вашего Web-узла, тем выше вероятность того, что вы сумеете предложить наиболее ценную и интересную информацию с его точки зрения. Обмениваясь сведениями о профилях посетителей с другими поставщиками информационного содержания, вы можете еше больше настроить интерфейс и содержание своего Web-узла. Если ваши партнеры используют один и тот же словарь XML при создании профилей посетителей, задача значительно упрощается, а ее решение оказывается гораздо эффективнее.

Некоторые интересные моменты связаны с неизменностью документов, содержащихся на Web-узлах. Существуют специальные средства, которые помогают отличить Web-узлы, предлагающие статические данные, от Web-узлов, содержание которых динамически изменяется. Например, документы, содержащие сведения о профилях, нужно сохранить так, чтобы эти сведения были доступными при первой необходимости. С другой стороны, содержание, созданное средством настройки, каждый раз разное. Содержание неизменно до тех пор, пока посетитель узла просматривает его. Статические Web-узлы предлагают сведения, которые практически не изменяются и не создаются динамически. Требования посетителей к узлам всегда состоят в динамическом создании данных. После просмотра динамически полученных данных они удаляются. При необходимости их всегда можно собрать снова.

Еще одна причина удаления динамических данных состоит в том, что обычно подобные данные постоянно обновляются. Если, например, ваш Web-узел предлагает такие сведения, как прогноз погоды или котировки акций, вам необходимо постоянно их обновлять. После того как данные использованы, они устаревают, и их можно удалить. Новый вызов котировок акций или прогноза погоды приводит к получению более свежих результатов.

Узлы сбора данных

Приложения для сбора данных в Internet идеально подходят для использования XML. Эти приложения консолидируют сведения из нескольких источников. Точно так же, как торговцы имеют дело с материальными товарами и различными платежными поручениями, подобные приложения занимаются подобными действиями в Web. Иногда консолидация осуществляется посредством перемещения материальных товаров через электронные каналы распространения. В других случаях консолидация касается цифровой информации.

Предположим, вы занимаетесь продажами систем "домашний кинотеатр" через Web-узел, который помогает своим посетителям выбрать отдельные компоненты благодаря предоставлению таких возможностей, как сравнение различных моделей, получение сведений о технических характеристиках и о цене. Сбор данных о различных компонентах также предлагается в виде услуги. Посетители вряд ли захотят прочитывать полные спецификации каждого из предложенных компонентов, и скорость изменения ассортимента доступных компонентов наверняка высока. Приложение сбора данных также должно обеспечить, чтобы все возможные комбинации компонентов представляли собой законченные и совместимые системы. При этом обязательно учитывать и такой фактор, как цены. Другими словами, при заданной сумме покупки оптимальное решение зависит от того, что предпочитает пользователь: качество звука или качество изображения, воспроизведение обычных музыкальных компакт-дисков, дисков с записями в формате MP3 или видеодисков. Дистрибьютор также предлагает такие сведения, как список супермаркетов, в которых можно приобрести соответствующие товары, магазины, распространяющие аудио- и видеопродукцию, и т.д.

Преимущества XML

Сбор информации в больших масштабах оказывается очень сложной задачей, что связано с существованием различных несовместимых систем. Например, спецификации, подробные сведения и данные заказов, предоставляемые производителями компонентов для "домашних кинотеатров", создаются на различных платформах и сохраняются с использованием тех или иных форматов файлов. Извлечение всех этих сведений и получение понятной для компьютерного приложения структуры оказывается чрезвычайно сложной задачей. А получение структуры данных, понятной не только компьютеру, но и человеку, — задача еще посложнее. Собственные форматы данных, используемые каждым из поставщиков, значительно усложняют сбор данных. После выполнения подобной операции обязательно необходимо убедиться в том, что полученные данные представлены в форме, понятной для посетителей Web-узла. Возможно, потребуется обеспечить несколько форматов представления данных, полученных от выполнения одной операции по сбору данных.

Использование XML для сбора данных из различных источников, объединение их в одном документе, а также предоставление результатов в нескольких форматах явно имеет смысл. Как вы уже знаете, язык XML понятен не только компьютеру, но и человеку. Собранные данные легко представить с помощью соответствующих элементов разметки. После этого, используя язык преобразований XSLT совместно со схемами, вы можете представить данные в различных форматах, не нарушая при этом целостности данных и не создавая нескольких источников данных. На самом деле схемы очень часто используются при сборе данных и, когда дело касается различных маркетинговых данных, могут обеспечить еще целый ряд преимуществ. Сведения о требованиях заказчиков, поставщиках решений, причем даже поставщиках только стереосистем, оказываются выгодными для организации, поскольку позволяют предложить варианты, вызывающие наибольший интерес клиентов.

Вопросы, связанные с архитектурой и разработкой

Получая доступ к данным, которые находятся в различных источниках, можно создать целый набор различных комбинаций информации. В некоторых случаях данные можно получить от поставщиков, которые способны поставлять сведения в формате XML. В подобных ситуациях поставщик использует схему, которую получает от вас или стороннего производителя, что позволяет предоставлять сведения строго определенной структуры. Другие поставщики данных могут использовать иные форматы, с которыми тоже придется иметь дело. Сбор информации и ее обработка всегда приводят хотя бы к минимальному, но выигрышу. После создания репозитания данных электронный запрос через Web может заставить приложение обратиться к необходимым документам, получить динамический набор данных, после чего применить к ним таблицу стилей XSL в соответствии с требованиями заказчика.

В качестве других примеров сбора сведений приведем следующие.

- Службы, занятые сбором данных о путешествиях в различных источниках, которые предоставляют потребителям сведения о минимальных ценах на авиабилеты, гостиничные номера, аренду автомобилей, услуги ресторана в указанных городах.
- Web-узлы, которые предлагают услуги по поиску минимальных цен на электронные и потребительские товары и т.д. Подобные узлы часто называют торговыми роботами. Они постоянно собирают сведения о различных товарах, в частности о фотокамерах, компьютерах, спортивных товарах, драгоценностях и т.д., а также всевозможные сравнения различных товарных групп.

Во всех этих случаях ключ к успеху спрятан в построении надежного и в высокой степени автоматизированного потока данных от поставщика информации через соответствующее приложение к потребителю информации. На каждом этапе этого процесса вам необходимо определять ценность тех или иных сведений. Сначала получают сведения, соответствующие определенным требованиям заказчика. После отсеивания всевозможных рекламных сведений и прочих маркетинговых уловок данные представляются в наиболее приемлемом для заказчика формате.

Разработка приложений подобного типа требует от вас выполнения и связывания данных, поступающих от поставщиков информации, с потребностями конкретных заказчиков. Определив, какие именно сведения доступны и каким образом вы предложите их заказчику, вы можете создать черновик схемы, чтобы узнать структуру сведений, участвуюших в публикации и получаемых от тех или иных поставщиков. Например, если речь идет о торговле через Web, производители товаров могут подписаться на службу, автоматически поставляющую необходимые сведения. Иногда организации самостоятельно заключают договоры о сборе сведений. Например, в июне 2001 года подробное соглашение подписано между авиакомпаниями *United*, *Delta*, *Continental*, *Nothwest* и *American Airlines*; оно касалось создания Web-узла `http://www.orbitz.com`, предназначенного для продаж авиабилетов потребителям, что явилось прямой конкуренцией Web-узлам различных туристических агентств. По данным *CNN*, на разработку соответствуюшей распределенной архитектуру авиакомпании выделили 50 миллионов долларов.

Интеграция звеньев поставок

К сбору данных относится и такое новое, базирующее на использовании XML направление, как интеграция звеньев поставки. Электронные системы значительно упрощают выполнение коммерческих транзакций между деловыми партнерами. Предположим, вы продаете компьютеры крупным супермаркетам, небольшим фирмам и другим организациям. Вы можете заключить договоры с целым рядом звеньев поставки, от которых получаете материнские платы, мониторы, клавиатуры, принтеры, сис-

темные корпуса и другие компоненты. Подобная цепочка поставки и является вашим рынком. Вы также поставляете компьютеры на рынок розничных торговцев. В подобной ситуации сведения об изменениях цен на те или иные комплектующие должны распространяться от поставщиков до розничных торговцев, причем каждый торговец может заказать комплектующие по новым ценам. XML позволяет разработчикам создать правильные документы, описывающие подобные транзакции.

Например, вам необходимо приобрести модули памяти для собираемых вами компьютеров. Вы можете обратиться к соответствующему звену поставки и заказать необходимое количество модулей памяти. После этого вы можете иметь дело только с тем поставщиком, который удовлетворит все ваши требования к количеству, качеству и цене.

Автоматизация значительно помогает при сборе данных, при обмене документами в соответствующих форматах между поставщиками и розничными торговцами. Созданные решения должны очень точно контролировать отправление и получение любых транзакций. В идеале получение всех сообщений должно подтверждаться, что позволит избежать проблем, связанных с возможными неполадками в соединениях с Internet. Автоматизированные системы должны передавать важные транзакции до тех пор, пока эти операции не будут выполнены успешно. Более того, решения должны обеспечить проверку подлинности и целостность данных на всех этапах обмена информацией.

От систематизации к обмену данных

Когда разработчики впервые рассмотрели вопросы, связанные с совместным использованием бизнес-данных через Web для ведения электронного бизнеса, внимание в основном обращали на словари и диалекты разметки. Анализ, как правило, заключался в том, обмен какой именно информацией должен произойти; кроме того, определялись используемые при обмене формат данных и схема. Многие из этих первых попыток построения систем сбора данных, например, следовали именно такой модели. Она работала при доставке информации во многих решениях; однако недостаток состоял в том, что для решения каждой конкретной задачи требовалось "настраивать" много различных моментов. Если структура бизнес-данных со временем изменялась, транзакции должны были создаваться заново, чтобы всегда полностью соответствовать схеме. Основная проблема состояла в том, чтобы перейти от сбора данных к решению фундаментального вопроса: каким именно образом должен проходить обмен данными XML между деловыми партнерами. В этом разделе вы познакомитесь с несколькими основными протоколами, используемыми при обмене данными XML, а также проблемами, которые каждый из них позволяет решить. При этом предполагается, что вы, как программист, знакомы с языками сценариев, основами HTTP и других используемых в Web-протоколов. Таким образом, мы остановимся именно на рассмотрении проблем, связанных с использованием того или иного протокола, а не на его основных конструкциях.

Консорциум *W3C* предоставил обзор целого ряда протоколов для обмена данными XML, который вы найдете по адресу `http://www.w3.org/2000/03/29-XML-protocol-matrix`. Несмотря на то, что консорциум предложил несколько десятков протоколов, наиболее явными претендентами на использование в качестве транспортных протоколов для передачи данных XML оказываются такие протоколы, как XML-PRC, SOAP, WDDX и ebXML.

Вызовы объекта HTTP в XML

Очень многие из бизнес-транзакций, используемых в электронном бизнесе, передаются через Internet по протоколу HTTP. Сымитировать подобные транзакции достаточно просто, воспользовавшись таким объектом XML, как `HTTP`, который позволяет извлекать

данные из отдаленных источников XML. Предположим, например, что вам необходимо очень быстро создать Web-страницу, способную извлекать любой документ XML с Web-сервера по указанному адресу URL. Вы можете использовать такой предложенный компанией *Microsoft* элемент XML, как `HTTP`, совместно с написанным на JavaScript небольшим программным кодом для выполнения подобных задач. На практике вы можете и не использовать технологии *Microsoft*, если только уверены в том, что они не нужны определенной части посетителей вашего узла. Мы будем рассматривать использование технологий *Microsoft* как один из подходов к решению поставленной перед нами проблемы.

С использованием JavaScript объект можно инициализировать как новый объект ActiveX точно так же, как вы создавали объект DOM `Microsoft.XMLDOM` в уроке дня 12-го. Синтаксис соответствующей команды приведен ниже:

```
var myVariable = new ActiveXObject("Microsoft.XMLHTTP")
```

После того как объект инициализирован, вы можете использовать вызовы методов для выполнения стандартных методов HTTP, таких как `GET` или `POST`. Метод `GET` можно использовать для получения документа XML с отдаленного сервера или любых других данных; метод `POST` предназначен для отправки документа XML с целью его обработки с помощью сценария на стороне сервера. Метод `GET` может выглядеть следующим образом:

```
myVariable.open("GET", (URL), false)
myVariable.send()
myResponseVariable =xmlHttp.responseText
```

Метод `open` устанавливает метод HTTP `GET` для указанного вами адреса URL. Параметр `false` указывает, что этот метод синхронный; таким образом, он должен завершить выполнение, прежде чем начнется выполнение другого метода. Метод `send()` передает метод `GET` по протоколу HTTP, а ответ, полученный в результате вызова, сохраняется в переменной *myResponseVariable*.

После того, как вы получили ответ (в нашем случае это полный текст файла XML), вы можете вывести его на экран, отправив его на стандартное устройство вывода. Для завершения этого простого примера вам необходимо заключить полученные данные в дескрипторы HTML `<XMP>`, которые позволят предотвратить обработку элементов разметки. Соответствующая инструкция приведена ниже:

```
document.write("<XMP>" + myResponseVariable + "</XMP>");
```

Полный вариант HTML-программы приведен в листинге 20.2. Создайте подобный документ и сохраните его под именем `httpGetXML.xml`.

Листинг 20.2. Программа получения файлов с использованием объекта XMLHTTP — `httpGetXML.xml`

```
<!DOCTYPE XHTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
 "http://www.w3.org/TR/html4/loose.dtd">
<!-- Листинг 20.2 httpGetXML.html -->

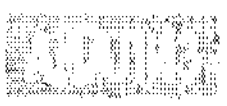
<HTML>
<BODY>
<SCRIPT language="JavaScript">

        var URL=prompt("Please enter the URL of an XML file to re
        trieve", "Enter XML File URL");

        var xmlHttp = new ActiveXObject("Microsoft.XMLHTTP")
        xmlHttp.open("GET", (URL), false)
        xmlHttp.send()
        xmlDoc=xmlHttp.responseText

```

```
16:          document.write("<XMP>" + xmlDoc + "</XMP>");
17: </SCRIPT>
18: </BODY>
19: </HTML>
```



В строках 9–10 создается стандартное окно JavaScript `prompt`, в котором пользователю предлагают ввести адрес URL интересующей его страницы. В строках 12–15 инициализируется объект HTTP (строка 12) и указывается синхронный метод `GET` (строка 13), который передается по протоколу HTTP. В строке 15 полученный результат сохраняется в переменной `xmlDoc`. Содержание этой переменной выводится на экране (строка 17).

Попробуйте загрузить эту страницу в броузер и введите адрес URL для страницы XML, которая хранится на отдаленном Web-сервере. Например, вы можете получить документ XML Schema, который определяет пространства имен *W3C* и находится по адресу `http://www.w3.org/XML/1998/namespace`. Если вы используете на своем компьютере Web-сервер IIS, вы также можете получить документ, созданный раньше и сохраненный на локальном компьютере.

XML-RPC и SOAP

XML-RPC — это протокол удаленного вызова процедур (remote procedure call), чьи данные, передаваемые через TCP-порт 80 (HTTP), задаются в формате XML. Протоколы XML-RPC написаны на многих языках программирования для многих платформ. Каждое приложение XML-RPC состоит из двух частей. Первая часть отвечает за запрос XML-RPC к определенной Web-службе и называется *вызовом клиента*. Если ваш сценарий отвечает на запросы XML-RPC, он называется *слушателем*. Сегодня мы будем использовать несколько реализаций XML-RPC. Эти реализации, которые базируются на таких различных технологиях, как PERL, Tcl, Python, PHP и AppleScript, обеспечивают независимость соответствующего механизма от используемых платформ и языков. Для решения подобных задач также подходят и такие языки, как Java, C++, C# и Visual Basic.

Прямым потомком протокола XML-RPC XML-RPC является протокол SOAP (Simple Object Access Protocol — простой протокол доступа к объектам), который также позволяет вызывать отдаленный объект посредством передачи простых параметров по протоколу HTTP. SOAP, как и XML-RPC, — это абстрактное представление взаимодействий распределенных объектов "клиент/сервер" в Web. Полная рекомендация SOAP 1.1 в настоящее время находится в стадии ноты консорциума *W3C*, поэтому ее дальнейшее рассмотрение и обсуждение продолжается. Текст ноты вы найдете по адресу `http://www.w3.org/TR/SOAP/`. SOAP — это диалект XML, т.е. он представляет собой облегченный протокол для обмена данными, требующий вызовов отдаленных объектов и возвращения результатов вызовов запрашивающей стороне. В ноте *W3C* SOAP определяется как протокол на основе XML, который состоит из трех частей: конверта (envelope), который определяет основу для описания того, что же содержится в сообщении и каким образом его необходимо обрабатывать, набора правил для выражения специфических для приложения типов данных и соглашений, касающихся представления отдаленных вызовов процедур и ответов.



> Протокол SOAP очень хорошо подходит для создания сложных приложений для вызовов удаленных объектов. Поддержка со стороны ведущих компаний отрасли обеспечивает разработку платформно-независимых решений. На момент написания книги технология еще недостаточно окрепла, а соответствующие средства разработки еще не получили серьезного распространения.

SOAP предоставляет вам возможность совершать удаленные вызовы методов объектов или функций. В традиционных XML-приложениях приложения, выполняемые на стороне клиента и на стороне сервера, должны точно знать, какой именно формат данных используется при обмене сообщениями. SOAP предлагает вам следующие возможности.

- Стандартный механизм для представления интерфейса вызова процедуры.
- Механизм запросов, позволяющий определить доступные функциональные возможности.
- Синтаксис вызовов.

Таким образом, SOAP способен заменить явные выражения XML, используемые в различных реализациях XML. Для того чтобы сделать этот процесс действительно прозрачным, должны быть доступны службы или инструменты, которые позволили бы использовать XML SOAP при выполнении обмена данными.

SOAP выполняет подобные операции по передачи параметров вызова объекта, заключая вызовы в стандартизированные структуры электронных конвертов. Представьте себе, что конверты SOAP полностью аналогичны обычным почтовым конвертам. Вы размещаете сведения, которые хотите передать объекту, в конверте, и отправляете их через Internet по протоколу HTTP. Это называется документом запроса SOAP. На другой стороне сервер открывает конверт, удаляет его содержание и передает его удаленному объекту в форме вызова. Ответ, полученный от объекта, снова помещается в конверт и возвращается через Internet по протоколу HTTP обратно клиенту. Это так называемый документ ответа SOAP. Документ запроса SOAP — это экземпляр документа XML, который содержит конверт SOAP, необязательный заголовок SOAP, а также обязательное "тело" SOAP. Тело документа SOAP содержит параметры вызова удаленного объекта. Сообщение SOAP должно соответствовать следующим правилам.

- Оно должно быть оформлено с использованием XML.
- Оно должно содержать конверт SOAP.
- Оно должно содержать заголовок SOAP.
- Оно должно использовать пространства имен конвертов SOAP.
- Оно должно использовать пространство имен SOAP.
- Оно должно содержать ссылку DTD.
- Оно не должно содержать инструкций обработки XML.

Синтаксис документа запроса SOAP приведен ниже:

```
<?xml version="1.0"?>
<SOAP-ENV:Envelope
        xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
        xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"

    <SOAP-ENV:Header>
     необязательные сведения о заголовке
        </SOAP-ENV:Header>

    <SOAP-ENV:Body>
     параметры вызова объекта
        </SOAP-ENV:Body>

</SOAP-ENV:Envelope>
```

Этот документ запроса SOAP содержит корневой элемент SOAP `ENV:Envelope`, который связан с пространством имен `http://schemas.xmlsoap.org/soap/envelope/`. Элемент `SOAP-ENV:Header` необязательный. Обязательный элемент `SOAP-ENV:Body` содержит все параметры, которые передаются удаленному объекту.

Синтаксис документа ответа SOAP приведен ниже:

```
<?xml version="1.0"?>
<SOAP-ENV:Envelope
        xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
        xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
    <SOAP-ENV:Header>
     параметры вызова объекта
    </SOAP-ENV:Header>

    <SOAP:Body>
     параметры вызова объекта
    </SOAP:Body>

</SOAP-ENV:Envelope>
```

Этот документ ответа SOAP содержит корневой элемент `SOAP-ENV:Envelope`, связанный с пространством имен `http://schemas.xmlsoap.org/soap/envelope/`. Элемент `SOAP-ENV:Header` необязательный. Обязательный элемент `SOAP-ENV:Body` содержит все параметры, которые передаются удаленному объекту.

Представьте себе, что вам необходим доступ к удаленному объекту, который возвращает текущее значение котировок акций. Вы можете передать объекту символ акции. После этого объект возвращает все необходимые сведения. В листинге 20.3 приведен полный вариант документа запроса SOAP.

Листинг 20.3. Документ запроса с использованием SOAP `SOAP_req.xml`

```
1: <?xml version="1.0"?>
2:  <!-- Листинг 20.3 SOAP_req.xml -->
3:
4:  <SOAP-ENV:Envelope
5:       xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
6:       xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
7:
8:     <SOAP-ENV:Header>
9:       <SOAPsrvr>123</SOAPsrvr>
10:     </SOAP-ENV:Header>
11:
12:     <SOAP-ENV:Body>
13:       <q:getQuote xmlns:q="urn:Devan's-delayed-quotes">
14:        <symbol>ssefx</symbol>
15:       </q:getQuote>
16:     </SOAP-ENV:Body>
17:
18: </SOAP-ENV:Envelope>
```



В этом примере корневой элемент `SOAP-ENV:Envelope` (строки 4–18) содержит элементы `SOAP-ENV:Header` (строки 8–10) и `SOAP-ENV:Body` (строки 12–16). Необходимые пространства имен для конверта и любых инструкций SOAP указаны в строках 5 и 6. Заголовок этого документа SOAP включает в себя элемент `SOAPsrvr`, который содержит данные, используемые сервером SOAP для управления передаваемым пакетом. Тело содержит элемент `q:getQuote`, связанный с пространством имен `urn:Devan's-delayed-quotes`. Элемент `symbol` содержит имя, в нашем случае, фонда `ssefx`, для которого сценарий возвращает последние котировки акций.

В листинге 20.4 приведен полный вариант документа ответа SOAP, полученного от сервера SOAP после выполнения вызова объекта.

Листинг 20.4. Документ ответа с использованием SOAP SOAP_resp.xml

```
1: <?xml version="1.0"?>
2:  <!-- Листинг 20.4 SOAP_resp.xml -->
3:
4:  <SOAP-ENV:Envelope
5:       xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
6:       xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
7:
8:     <SOAP-ENV:Header>
9:       <ProcessID mustUnderstand="0">0</ProcessID>
10:     </SOAP-ENV:Header>
11:
12:     <SOAP:Body>
13:       <qres:getQuoteResponse xmlns:qres='urn:Devan's-delayed-
          quotes'> <return>14.59</return>
14:     </qres:getQuoteResponse>
15:   </SOAP:Body>
16:
17: </SOAP-ENV:Envelope>
```

В этом примере корневой элемент `SOAP-ENV:Envelope` (строки 4—18) содержит элементы `SOAP-ENV:Header` (строки 8—10) и `SOAP-ENV:Body` (строки 12—16). Необходимые пространства имен для конверта и любых инструкций SOAP указаны в строках 5 и 6. Заголовок этого документа SOAP включает элемент `SOAPsrvr`, который содержит данные, используемые сервером SOAP для управления передаваемым пакетом. Тело содержит элемент `q:getQuote`, связанный с пространством имен `urn:Devan's-delayed-quotes`. Элемент `return` содержит сведения о последних котировках акций фонда `ssefx`.

Для того чтобы эти сообщения SOAP функционировали должным образом, вам необходимо создать клиент-приложение SOAP для отправки и получения подобных сообщений. Сервер SOAP интерпретирует сообщения и передает параметры удаленному объекту. Кроме того, сервер SOAP получает результаты от объекта и размещает их в документе ответа SOAP, который возвращается клиентскому приложению. На рис. 20.1 показан пример очень простого клиента SOAP, созданного с использованием среды разработки IBM Sash Weblications. Этот клиент представляет собой несколько небольших приложений, которые используют сообщения SOAP для вызовов удаленных объектов.

ПРИМЕЧАНИЕ

IBM Sash Weblications — это среда разработки, позволяющая очень тонкую настройку параметров работы, используется для создания различных сетевых приложений. Некоторые разработчики используют эту среду для создания приложений электронного бизнеса и связывают их с серверами HTTP, используя существующие ресурсы и инфраструктуру. Вы можете использовать IBM Sash Weblications для превращения Web-страниц в полнофункциональные приложения Windows, содержащие данные, загруженные из Web.

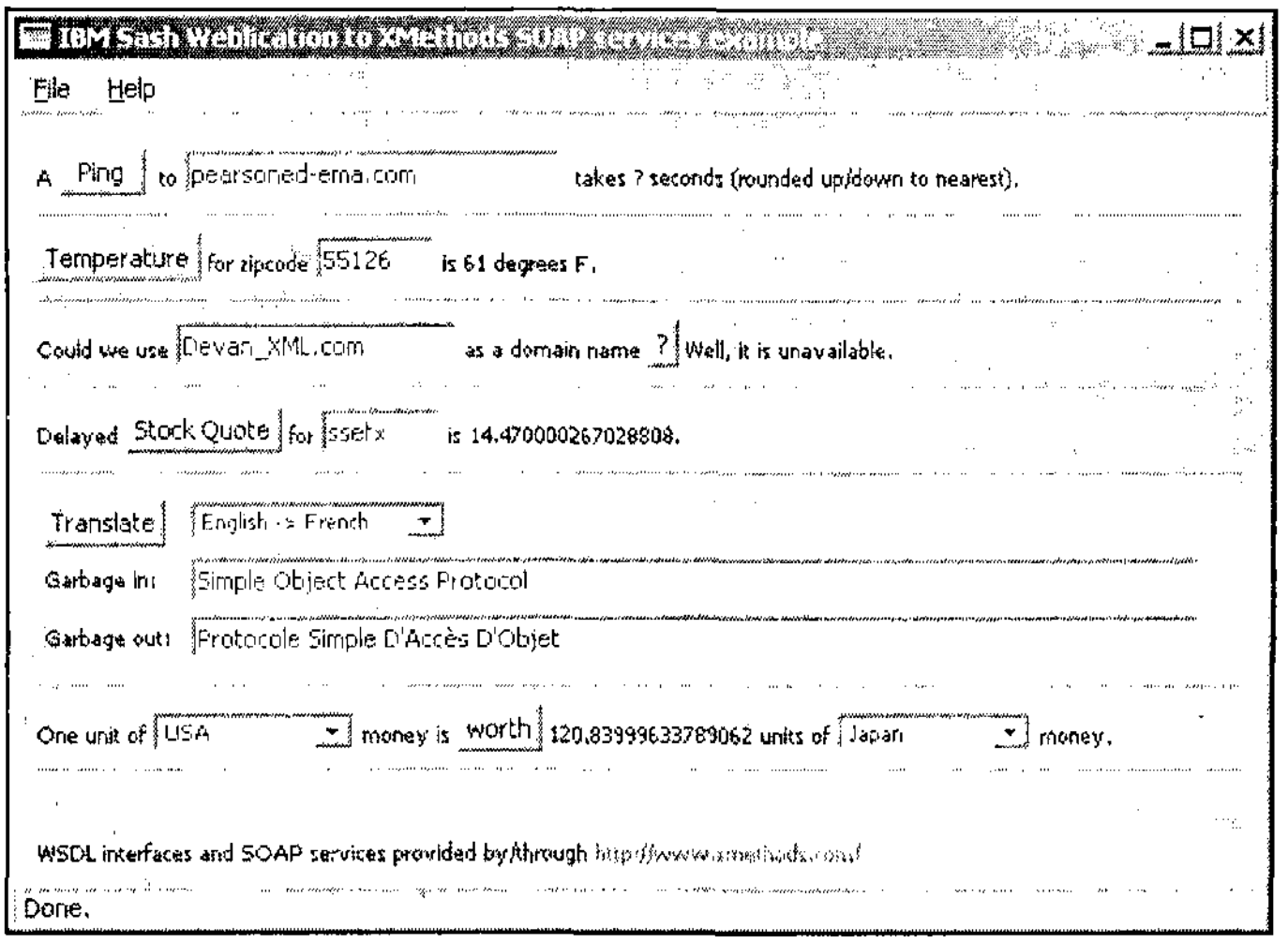


Рис. 20.1. Пример типичного клиента SOAP

Вы можете создать свой собственный клиент SOAP и серверные программы для доступа к объектам с использованием одного из существующих средств разработки SOAP. В эти пакеты входят средства для создания как клиентских, так и серверных приложений, а также целый ряд полезных примеров. К подобным средствам относятся следующие.

- IBM SOAP for Java (теперь часть Apache XML Project): `http://xml.apache.org/soap/`
- Lucin SOAP Toolkit: `http://www.soaptoolset.com/`
- Microsoft SOAP Toolkit: `http://msdn.microsoft.com/xml/general/soapland2.asp`

SOAP поддерживает все типы данных, перечисленные в части "Built-in Datatypes" рабочего черновика XML Schema Part 2: Datatypes. Он также поддерживает различные структуры и базирующиеся на массивах составные типы (`http://www.w3.org/TR/2001/REC-xmlschema-2-20010502/`).

Web Distributed Data Exchange (WDDX)

Протокол WDDX, разработанный компанией *Allaire* (`http://www.allaire.com/handlers/index.cfm?ID=5624&Method=full`), предоставляет механизм обмена сложными структурами данных по протоколу HTTP, альтернативен протоколу SOAP. В соответствии с документацией компании *Allaire*, протокол WDDX предназначен для организации обмена сложными объектами данных между элементами сети без изменения основных подходов к разработке Web-приложений.

WDDX отличается от XML-RPC и SOAP двумя основными характеристиками. Во-первых, его можно охарактеризовать как базирующийся на структурах, а не на событиях. Кроме того, WDDX не базируется на семантике удаленных вызовов процедур RPC. Основными характеристиками WDDX являются схема WDDX DTD и модуль сериализации, который преобразует собственные структуры данных определенного языка в XML и наоборот. Пакет WDDX создается в тот момент, когда приложение преобразует структуры данных к WDDX. Эти пакеты проверяются с использованием схем WDDX DTD. В листинге 20.5 приведен простой документ WDDX, представляющий собой один пакет, который, подобно приведенному раньше примеру SOAP, отправляется для получения сведений о стоимости фонда.

Листинг 20.5. Пакет WDDX XML для получения сведений о стоимости WDDX_stock.xml

```
1: <?xml version='1.0'?>
2:  <!-- Листинг 20.5, WDDX_stock.xml -->
3:
4:   <!DOCTYPE wddxPacket SYSTEM 'Stock_check.dtd'>
5:   <wddxPacket version='0.9'>
6:       <data>
7:           <struct>
8:               <var name='getQuote'>
9:                   <value>SSEFX</value>
10:               </var>
11:           </struct>
12:       </data>
13: </wddxPacket>
```

АНАЛИЗ Корневой элемент `wddxPacket` (строки 5–13) содержит дочерний элемент `data`. Элемент `data` (строки 6–12), в свою очередь, содержит элемент `struct` (строки 7–11). Элемент `struct` включает элемент `var` (строки 8–10), который передает параметры приложению, которое выполняется на сервере.

Компания *Allaire* выпустила спецификацию WDDX для ознакомления с ней всего сообщества разработчиков. WDDX реализован с использованием ColdFusion, а его SDK поддерживает такие языки, как JavaScript, Java, PERL и COM/ASP.

ebXML

ebXML (аббревиатура от electronic business XML — XML для электронного бизнеса) — это инициатива организаций *UN/CEFACT* (United Nations body for Trade Facilitation and Electronic Business — комитет Организации Объединенных Наций по содействию торговле и электронному бизнесу) и *OASIS* (Organization for the Advancement of Structured Information Standards — Организация по продвижению стандартов структурированной информации). Основное назначение этой инициативы — предоставление открытой XML-инфраструктуры, обеспечивающей безопасное глобальное использование информации электронного бизнеса (источник: `http://www.ebxml.org/`). В настоящее время ebXML еще не получил какой-либо серьезной поддержки со стороны основных производителей программного обеспечения.

Над разработкой ebXML работает несколько групп, отвечающих за различные аспекты электронного бизнеса и разрабатывающих те или иные технологии XML. Команда, отвечающая за передачу и маршрутизацию данных, разработала рекомендации, касающиеся спецификации конверта сообщений, общих требований к документам, а также спецификации заголовков сообщений.

Конверт сообщения — это документ XML, который состоит из заголовка конверта и его содержания. Заголовок конверта содержит заголовок сообщения, а также сведения о маршрутизации. Пример заголовка конверта приведен в листинге 20.6.

Листинг 20.6. Конверт заголовка ebXML ebXML_header.xml

```
1: <?xml version="1.0" encoding="UTF-8"?>
2:  <!-- Листинг  20.6, ebXML_header.xml -->
3:
4:  <ebXMLMessageHeader>
5:     <Version>1.0</Version>
6:     <MessageType>Request</MessageType>
7:     <ServiceType>Stock Quote</ServiceType>
```

```
8:     <Intent>Last Trading Price</Intent>
9:  </ebXMLMessageHeader>
```

АНАЛИЗ

Корневой элемент `ebXMLMessageHeader` (строки 4–9) содержит дочерние элементы `Version`, `MessageType`, `ServiceType` и `Intent`. Эти элементы содержат все необходимые сведения маршрутизации, инструкции, а также цифровые подписи, необходимые получающим приложениям.

Основная часть конверта может содержать один или несколько сегментов. Пример простого тела сообщения приведен в листинге 20.7. Этот документ неполон, но его достаточно для иллюстрации основных моментов.

Листинг 20.7. Пример документа ebXML `ebXML_payload.xml`

```
1: <!-- Листинг  20.7, ebXML_payload.xml -->
2:
3:   <Control>
4:    <Session Identity="Quote">
5:     <Value>SSEFX</Value>
6:    </Session>
7:   </Control>
```

АНАЛИЗ

Этот фрагмент кода содержит пример тела сообщения. Элемент `Control` (строки 3–7) содержит элементы `Session` (строки 4–6) и `Value` (строка 5), которые передают сведения приложению, возвращающему данные о котировках акций.

На момент написания настоящей книги стандарту ebXML уже почти два года, хотя первая спецификация датирована маем 2001 года. Создается впечатление, что многие средства ebXML разработаны для компенсации ряда недостатков SOAP, поскольку спецификация SOAP 1.0 не обеспечивает обработку двоичных данных, что было одной из целей, поставленных перед рабочими группами, занятыми разработкой ebXML. ebXML создавался для обеспечения глобального использования сведений электронного бизнеса, совершенно безопасного для всех участников.

Несмотря на то, что у ebXML и SOAP очень много общего, комитеты, отвечающие за разработку ebXML, полагают, что SOAP является недостаточно открытым стандартом. Таким образом, ebXML не может реализовать все функции SOAP. Несмотря ни на что, некоторую схожесть отрицать просто нельзя. Оба подхóда обеспечивают безопасную и надежную передачу деловых документов. Оба построены на стандартных технологиях MIME и используют заголовки XML и средства маршрутизации. Два этих стандарта настолько похожи, что у разработчиков часто возникает вопрос, а почему они оба существуют. В связи с этим комитеты, отвечающие за разработку ebXML, согласились заимствовать часть спецификации SOAP, чтобы избежать реализации в своих продуктах двух различных стандартов. Это означает, что со временем ebXML превратится в дополняющую, а не конкурирующую технологию. Будет обеспечена поддержка сообщений SOAP при обмене документами. Я надеюсь, что в обозримом будущем компании-производители все-таки предложат полноценные реализации ebXML.

Резюме

Сегодня вы познакомились с различными технологиями XML, используемыми в электронном бизнесе. Многие из подобных подходов касаются организации обмена коммерческими транзакциями через Internet по протоколу HTTP. Вы рассмотрели не-

сколько сценариев приложений для рынков B2B и B2C, познакомились с использованием каждого из них. Предприятия делают достаточно крупные капиталовложения для использования технологий XML в электронной коммерции в будущем.

Вопросы и ответы

Каким образом можно использовать XML на узлах сбора данных?

XML обеспечивает возможностями контроля и интеллектуального хранения данных. XML идеально подходит для сред, в которых используются различные форматы исходных данных, которые нужно объединить в одном результирующем документе.

Каким образом управление звеньями поставок на основе XML может улучшить бизнес-модель предприятия?

Электронные средства управления звеньями поставок позволяют значительно повысить скорость выполнения различных деловых операций. Контроль над потоком бизнес-транзакций можно полностью автоматизировать.

Где можно найти подробные сведения о реализациях SOAP для обработки транзакций B2B?

Вы можете найти очень много полезной информации в Web, посетив, например, следующие узлы.

- `http://www.soapware.org`. Этот узел предлагает сведения о спецификации, каталоги, учебники, примеры реализации, статьи и новости из мира SOAP.
- `http://soap.weblogs.com`. На этом узле вы найдете сведения, полученные из различных групп новостей, посвященных SOAP, интервью со специалистами и документацию.
- `http://www.xmlrpc.com`. На этом узле содержатся различные сведения и новости о простых распределенных и платформно-независимых вычислениях.
- `http://blogspace.com/rss/`. На этом узле вам предлагается приложение Web-сервера, которое можно запустить на вашем компьютере. Это мощное средство для сбора новостей, поддерживающее протоколы XML-RPC и SOAP 1.1.

На каких стандартах в настоящее время базируется протокол SOAP?

SOAP базируется на протоколе HTTP версии 1.0 или выше и использует ряд преимуществ этого протокола (`http://www.w3.org/Protocols/HTTP/ietf-http-ext`). SOAP также полагается на рекомендацию XML консорциума *W3C* (`http://www.w3.org/TR/1998/REC-xml-19980210`). SOAP поддерживает спецификацию пространства имен XML консорциума *W3C* (`http://www.w3.org/TR/REC-xml-names`). Сообщения SOAP должны быть правильными документами XML, но их проверка с помощью схем (DTD или других) не нужна. На момент написания настоящей книги рассматривалось использование языка описания схем XML Schema (`http://www.w3.org/TR/xmlschema-1/`) для описания конечных точек SOAP, но в спецификации SOAP 1.0 это не отражено.

Почему разработчики отдают предпочтение SOAP перед решением на основе XML/HTTP?

SOAP предлагает значительные преимущества по сравнению с собственными словарями XML. Растущая поддержка SOAP со стороны компаний — производителей программного обеспечения, предлагающих законченные решения, обеспечит улучшенное качество реализации этих технологий. Клиентские и серверные приложения очень быстро становятся доступными для основных сфер деятельности.

Какие ограничения SOAP существуют?

SOAP не обеспечивает двунаправленные взаимодействия, хотя решение подобных задач возможно на основе уже существующих реализаций SOAP. Текущая спецификация SOAP описывает передачу данных по протоколу HTTP и не обеспечивает использование других протоколов.

Поскольку данные SOAP передаются по протоколу HTTP, они могут проходить через основную часть существующих брандмауэров; каким образом это сказывается на безопасности обмена данными?

Данные SOAP передаются по протоколу HTTP, а значит, они не менее безопасны, чем любые другие данные, которые передаются по этому протоколу. Кроме того, разработчики SOAP не включили в протокол инструкции, которые приводят к потенциальным проблемам безопасности. Поскольку пакеты SOAP содержат данные заголовков HTTP, в будущем брандмауэры могут обзавестить возможностями фильтрации данных на основе данных заголовков. Как и в случае передачи других данных по протоколу HTTP, SOAP поддерживает SSL (Secure Sockets Layer) и ряд других схем проверки подлинности.

Каким образом транзакции B2B улучшают ведение деловых операций?

Во многих случаях модель транзакций B2B полностью заменяет любые действия с бумагами. Замена этих операций электронными процессами, которые выполняются быстрее, намного точнее и гибче, позволяют всем участникам рынка получить определенные преимущества.

День 21-й

Разработка Web-приложения с помощью XML

XML очень хорошо подходит для использования на Web-узлах, особенно в тех ситуациях, когда преобразования XSL, сценарии и другие формы обработки выполняются на стороне сервера, а не на стороне клиента. Объединив значительную часть технологий, описанных в этой книге, вы сможете создать небольшое Web-приложение уже сейчас.

Сегодня рассматриваются такие темы.

- Создание Web-приложения на основе файла XML.
- Выполнение преобразований XSLT на стороне сервера.
- Использование объектной модели документа XML для создания узлов дерева XML.

Проектирование Web-приложения

В настоящей главе вы создадите приложение Web-узла, которое позволит любителям вина вводить замечания обо всех перепробованных ими винах. Собранные данные включают в себя такие сведения: название вина, производитель, регион происхождения, тип вина, выдержка, а также личные замечания о том или ином вине. Записи сохраняются в базах данных и извлекаются оттуда точно так же, как вы обходились с записями сообщений в уроке дня 19-го. Однако вы будете использовать именно XML для сохранения данных. В частности, данные содержатся в файле XML на сервере, а вы инициализируете объект модели XML DOM для создания новых узлов в сохраненном дереве документа. Сценарии на стороне сервера, выполненные в виде Active Server Pages, используются для преобразований XSLT над исходным документом XML для приведения его к формату HTML, когда надо представить все сохраненные данные в виде таблицы.

Приложение представляет собой набор связанных документов. В упражнениях, которые вы найдете дальше, вы создадите каждый из этих документов, после чего разместите их на Web-узле для выполнения. Для создания приложения указанным образом вам необходимы несколько инструментов из арсенала компании *Microsoft*. Вы можете очень легко создать подобное приложения с помощью других приложений, если необходимо.

Структура приложения будет включать в себя начальную страницу (`index.html`) со ссылками на форму для ввода данных (`wine_notes.html`), сценарий на стороне

сервера (view_notes.asp), который перечисляет все записи из набора данных (wine_notes.xml). Необязательная каскадная таблица стилей (wine_notes.css) позволяет придать общий стиль всему приложению, применив одинаковое форматирование ко всем HTML-страницам. После того как пользователь заполнит форму (wine_notes.html), сценарий на стороне сервера (add_notes.asp) обработает введенные данные и передаст их в документ wine_notes.xml. Еще один сценарий на стороне сервера используется для просмотра пользователями полного набора сведений (view_notes.asp). Описанная нами архитектура приложения приведена на рис. 21.1.

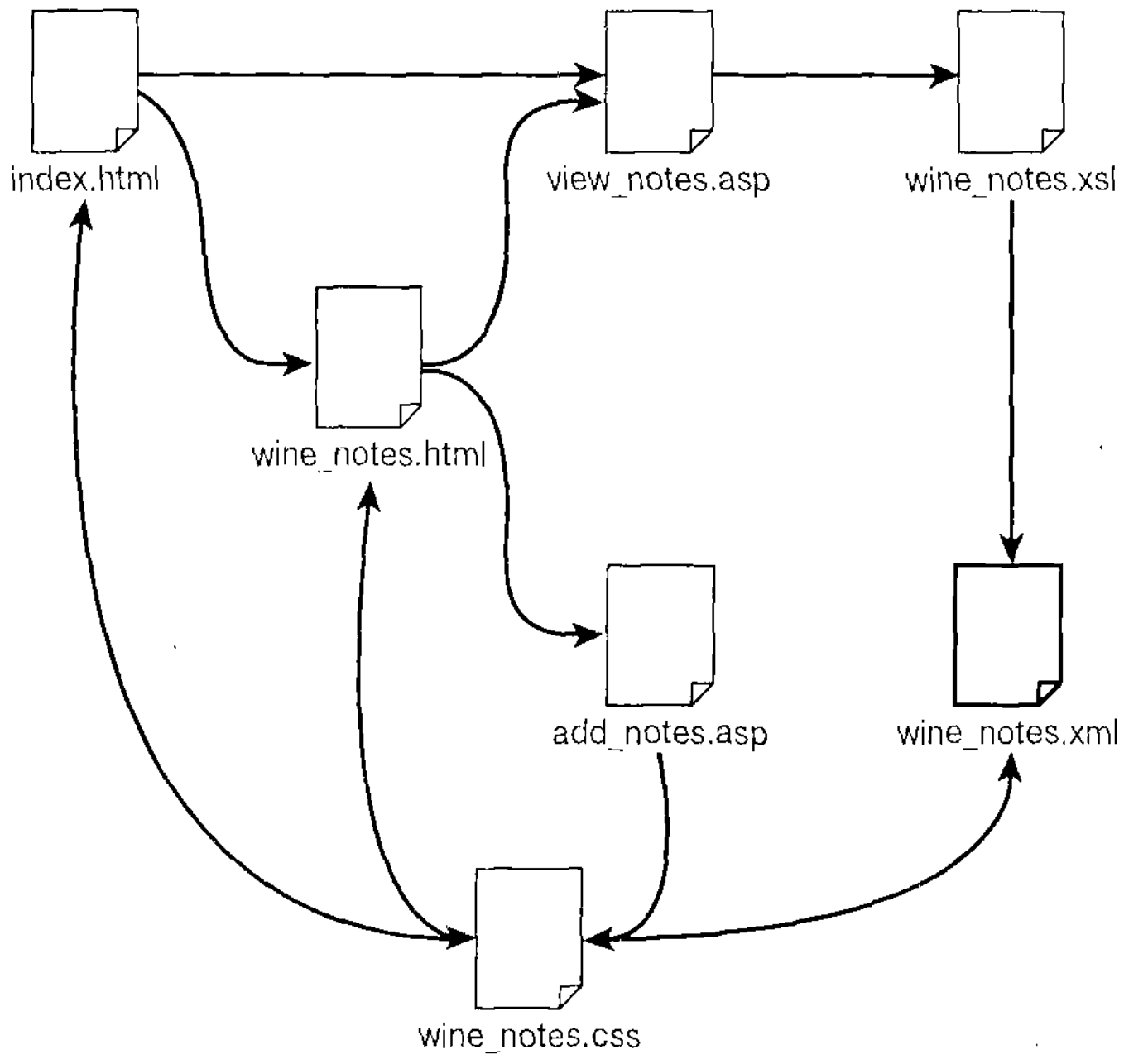


Рис. 21.1. Архитектура типичного Web-приложения

Сохранение записей в документе wine_notes.xml

Данные для приложения будут сохраняться в файле XML в рамках корневого элемента wine_notes. Каждый элемент wine представляет собой аналог записи в иерархической структуре базы данных. Элемент wine содержит такие дочерние элементы, как house, region, type, vintage и notes. Элемент notes указывается как раздел CDATA, что гарантирует, что все введенные пользователем данные сохранятся, но не будут обрабатываться синтаксическим анализатором. Атрибут date элемента wine содержит дату и временную метку, которые показывают, когда именно запись добавлена в набор данных. В листинге 21.1 приведен пример файла wine_notes.xml, содержащего несколько записей. Вам необходимо ввести как минимум одну запись вручную при создании этого документа. Создайте подобный документ и сохраните его под именем wine_notes.xml.

Листинг 21.1. Набор данных Wine Notes — `wine_notes.xml`

```
1: <?xml version="1.0"?>
2: <!-- Листинг 21.1 - wine_notes.xml -->
3:
4: <wine_notes>
5:  <wine date="6/9/2001 10:40:01">
6:      <house>Chateau de La Chaize</house>
7:      <region>Brouilly, France</region>
8:      <type>Beaujolais</type>
9:      <vintage>1998</vintage>
10:     <notes><![CDATA[Full of  berries and firm enough to hold its own
11: with a variety of foods, particularly aged hard cheeses]]></notes>
12: </wine>
13: <wine date="6/9/2001 11:20:08">
14:     <house>Chateau d'Epire</house>
15:     <region>Savennieres, France</region>
16:     <type>Cuvee Speciale</type>
17:     <vintage>1998</vintage>
18:     <notes><![CDATA[Firm, flinty acid, with a late spiciness that
19: is reminiscent of the soft, lactic tingle of a Berlinerweisse]]></notes>
20: </wine>
21: <wine date="6/9/2001 4:48:18 PM">
22:     <house>Estancia Estates</house>
23:     <region>Ruterford, California</region>
24:     <type>Pinot Noir</type>
25:     <vintage>1997</vintage>
26:     <notes><![CDATA[Fat and juicy, if not overly lingering, with
27: a hint of cherry flavor]]></notes>
28: </wine><wine date="6/9/2001 5:47:39 PM"><house>Domaine Bru
29:  Bache</house><region>Pau, France</region><type>Jurancon
30:  Sec</type><vintage>1997</vintage><notes><![CDATA[Sweet in an intensely-
31: grapey way, with the
32: flintiness of a sauvignon blanc, but leaner body]]></notes></wine>
33: </wine_notes>
```

Корневой элемент `wine_notes` (строки 4–33) содержит дочерние элементы `wine` с атрибутами `date`. Каждый из элементов `wine` содержит дочерние элементы `house`, `region`, `type`, `vintage` и `notes`. Элементы `note` задаются как разделы `CDATA`.

Создание Web-формы wine_notes.html

`wine_notes.html` — это HTML-страница, которая содержит форму для ввода сведений об определенной марке вина. Форма, которую вы создадите в настоящем уроке, содержит поля, перечисленные в табл. 21.1. Поля формы соответствуют определенным элементам в файле XML (`wine_notes.xml`).

Таблица 21.1. Поля формы и соответствующие им элементы XML

Поле HTML	*Элемент XML*	*Описание*
House/Label	`house`	Название производителя вина или его марки
Region/Location	`region`	Регион или страна происхождения вина
Type/Varietal	`type`	Вид вина, сорт винограда
Vintage	`vintage`	Год урожая
Tasting Notes	`notes`	Любые замечания, введенные пользователем. Данные помещаются в документ XML как раздел `CDATA`

Каждое из этих полей, за исключением поля Tasting Notes, создается с помощью стандартного дочернего элемента HTML `INPUT` по отношению к элементу `FORM`. Каждый из подобных элементов `INPUT` задается в следующей форме:

```
<INPUT SIZE="ЗНАЧЕНИЕ" TYPE=TEXT NAME="ССЫЛКА" MAXLENGTH="ЗНАЧЕНИЕ">
```

Поле для ввода замечаний указывается с помощью элемента `TEXTAREA`, как показано ниже:

```
<TEXTAREA NAME="ССЫЛКА" COLS="ЗНАЧЕНИЕ" ROWS="ЗНАЧЕНИЕ">
```

Значительная часть HTML-страниц этого приложения будет использовать внешнюю каскадную таблицу стилей CSS для придания общего вида всему узлу. Файл CSS называется `wine_notes.css` и связан с текущей страницей с помощью элемента `LINK`, как показано ниже.

```
<LINK rel = "stylesheet" type = "text/css"
          href = "wine_notes.css"/>
```

Форма, созданная на HTML-странице с помощью элемента `FORM`, использует метод `POST` для передачи полученной от пользователя информации сценарию `add_notes.asp`, который добавит данные в документ `wine_notes.xml`, созданный вами раньше. Полный код HTML-страницы приведен в листинге 21.2. Создайте документ и сохраните его под именем `wine_notes.html`.

Листинг 21.2. HTML-форма для сбора сведений о дегустации вин `wine_notes.html`

```
1: <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
2:  "http://www.w3.org/TR/Html4/loose.dtd">
3: <!-- Листинг 21.2 - wine_notes.html -->
4:
5: <HTML>
6:   <HEAD>
7:     <TITLE>Online Wine Tasting Notes</TITLE>
8:     <LINK rel = "stylesheet" type = "text/css"
9:           href = "wine_notes.css"/>
10:  </HEAD>
11:  <BODY>
12:    <H1>Online Wine Tasting Notes</H1>
13:    <H3>Add Your Wine Tasting Notes:</H3>
14:    <FORM METHOD=POST ACTION="add_notes.asp">
15:      <TABLE border ="1">
16:        <TR>
```

```
17:            <TD>
18:               <TABLE cellspacing="3" cellpadding="3">
19:                 <TR>
20:          <TD ALIGN="RIGHT">House/Label:</TD>
21:          <TD ><INPUT SIZE="40" TYPE=TEXT NAME="HOUSE" MAXLENGTH="60">
22:          </TD>
23:          </TD>
24:        </TR>
25:        <TR>
26:          <TD ALIGN=RIGHT>Region/Location:</TD>
27:          <TD><INPUT SIZE="40" TYPE=TEXT NAME="REGION" MAXLENGTH="60">
28:          </TD>
29:        </TR>
30:        <TR>
31:          <TD ALIGN=RIGHT>Type/Varietal:</TD>
32:          <TD><INPUT SIZE="40" TYPE=TEXT NAME="TYPE" MAXLENGTH="60">
33:                
34:               <INPUT TYPE="RESET" VALUE="Clear the Form">
35:          </TD>
36:        </TR>
37:        <TR>
38:          <TD ALIGN=RIGHT>Vintage:</TD>
39:          <TD><INPUT SIZE="40" TYPE=TEXT NAME="VINTAGE" MAXLENGTH="20">
40:                
41:              <INPUT TYPE="SUBMIT" VALUE="Add Notes To Collection">
42:          </TD>
43:        </TR>
44:        <TR>
45:          <TD ALIGN=RIGHT VALIGN=TOP>Tasting Notes:</TD>
46:          <TD><TEXTAREA NAME="NOTES" COLS="60" ROWS="5"></TEXTAREA>
47:        </TR>
48:
49:      </TABLE>
50:
51:      </TD>
52:     </TR>
53:    </TABLE>
54:      <UL>
55:        <LI><A HREF="index.html">Return to HomePage</A></LI>
56:      </UL>
57:  </FORM>
58:
59:  </BODY>
60:
61:  </HTML>
```



В строках 14–57 содержится элемент FORM, предназначенный для получения данных, введенных пользователем. Элементы house (строка 21), region (строка 27), type (строка 32) и vintage (строка 39) соответствуют полям для ввода текстовых данных. Элемент note указан с помощью дескриптора TEXTAREA (строка 46). Элемент LINK, указывающий на каскадную таблицу стилей, содержится в строках 8 и 9. Кнопка сброса (<INPUT TYPE="RESET" VALUE="Clear the Form">) задана в строке 34, позволяя пользователям очищать поля формы. Кнопка отправки (<INPUT TYPE="SUBMIT" VALUE="Add Notes To Collection">) задана в строке 41, позволяя пользователям отправить заполненную форму. Форма обрабатывается с помощью метода PASS (<FORM METHOD=POST ACTION="add_notes.asp">), который передает собранные данные в файл add_notes.asp. Дескриптор <A> в строке 55 содержит ссылку на главную страницу Web-приложения.

Применение стилей к Web-узлу с помощью таблицы стилей wine_notes.css

Использование каскадной таблицы стилей, строго говоря, необязательно. Действительно, если вы опустите файл CSS, ваше приложение не будет исключено броузерами, которые не полностью поддерживают каскадные таблицы стилей. Если вы опустили файл CSS, обязательно удалите дескрипторы <LINK> из HTML-страниц приложения, хотя броузеры, которые не полностью поддерживают каскадные таблицы стилей, просто проигнорируют эти дескрипторы.

Преимущество использования файла CSS состоит в том, что он позволяет придать единый вид всем страницам приложения. Именно для этого мы и будем использовать файл CSS в нашем примере. В данном случае каскадная таблица стилей CSS — это только одна из моделей, которые вы можете использовать для применения стилей к приложению. Создайте свои стили или используйте стили, приведенные в листинге 21.3.

Листинг 21.3. Необязательные стили, примененные к приложению, wine_notes.css

```
1:  <!-- Листинг 21.3 - wine_notes.css -->
2:
3:      BODY        {display:block; background-color:#999966;
4:                  background-image:URL(http://www.samspublishing.com/images/
5:                   samslogo.gif);
6:                   background-repeat:no-repeat}
7:
8:      H1          {text-align:center;padding:6px;font-weight:bold;
9:                   color:#99cc99;font-style:bold;
10:                  font-family:verdana,arial,sans-serif;}
11:
12:     H3          {padding:6px;font-size:12pt;
13:                  font-family:verdana,arial,sans-serif;}
14:
15:     P           {font-family:verdana,arial,sans-serif;}
16:
17:     TD          {font-family:verdana,arial,sans-serif;
```

```
18:                 background-color:#99cc99;}
19:
20:     A:link    {color:#99cc99;font-size:14pt;text-decoration:none;}
21:
22:     A:visited {color:#99cc66;font-size:14pt;text-decoration:none;}
23:
24:     A:hover   {text-decoration:underline; color:red;
                  font-weight:bold;}
25:
26:     .head     {background-color:99cc99;
27:                 font-family:verdana,arial,sans-serif;
                   font-size:14pt;}
28:
29:     SPAN      {font-family:verdana,arial,sans-serif;
                   font-size:12pt;}
30:
31:    TR        {font-family:verdana,arial,sans-serif;}
32:
33:    LI        {font-family:verdana,arial,sans-serif;}
```



Каждый из элементов разметки, к которому применяется определенный стиль, указан в этом файле CSS вместе с соответствующим стилем. В строках 4 и 5 содержатся ссылки на графические изображения, которые хранятся на Web-сервере. При необходимости вы можете указать графические изображения, которые содержатся на жестком диске вашего компьютера.

Конструирование начальной страницы index.html

Создайте начальную страницу для приложения и назовите ее `index.html`. Здесь должны быть представлены всего две ссылки и некоторый текст. Одна ссылка указывает на страницу `wine_notes.html`, которую вы создали раньше. Другая ссылка должна указывать на страницу `view_notes.asp`, которая собирает сведения из документа `wine_notes.xml`.

Возможный вариант страницы `index.html` приведен в листинге 21.4. Если вы решили использовать файл CSS, укажите его с помощью дескриптора `<LINK>`.

Листинг 21.4. Начальная страница Web-приложения Wine Tasting Notes `index.html`

```
1: <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
2:  "http://www.w3.org/TR/Html4/loose.dtd">
3: <!-- Листинг 21.4 - index.html -->
4:
5: <HTML>
6:   <HEAD>
7:    <TITLE>Wine Tasting Notes</TITLE>
8:    <LINK rel = "stylesheet" type = "text/css" href =
     "wine_notes.css">
9:   </HEAD>
10:   <BODY>
11:    <H1>Online Wine Tasting Notes</H1>
12:    <HR/>
```

```
13:     <H3>Navigation</H3>
14:     <H3>Please select one of the following:</H3>
15:     <UL>
16:       <LI><A HREF="view_notes.asp">View all of the Wine Note Rec-
          ords</A></LI>
17:       <LI><A HREF="wine_notes.html">Add a new Wine Tasting
          Note</A></LI>
18:      </UL>
19:    </BODY>
20: </HTML>
```



Начальная страница содержит ссылки на документ `wine_notes.html` (строка 17) и документ `view_notes.asp` (строка 16). Дескриптор `<LINK>`, указывающий на таблицу стилей `wine_notes.css`, записан в строке 8.

Управление данными на сервере с помощью сценария add_notes.asp

Сценарий ASP (Active Server Pages) используется для добавления сведений, полученных от пользователей, в документ `wines_notes.xml`. Для этого предусмотрены объекты DOM, подобные тем, что вы использовали в главе 12. Вы можете начать с определения переменных, которые будут содержать данные, полученные с помощью формы, для каждого из узлов элементов в наборе данных. Сюда относятся такие элементы, как `wHouse`, `wRegion`, `wType`, `wVintage` и `wNotes`. Таким образом, переменные задаются следующим образом:

```
Dim wHouse
Dim wRegion
  Dim wType
Dim wVintage
Dim wNotes
```

Вам также потребуется определить переменные, которые будут содержать отдельные данные, переданные с помощью DOM узлам в экземпляре документа XML. Вы создадите переменные для узла документа, корневого узла, а также узлов `notes`. Вам также потребуется атрибут `notes`, который будет содержать такие сведения, как дата и время. Таким образом, переменные задаются следующим образом:

```
Dim oWineNotesDOM
Dim oWineNotesRootNode
Dim oDateStamp
Dim oWineNotesNodes
```

Поскольку вы передаете данные непосредственно из полей формы HTML, вы можете "перехватывать" соответствующие значения HTTP, как показано ниже:

```
wHouse=Request("HOUSE")
wRegion=Request("REGION")
wType=Request("TYPE")
wVintage=Request("VINTAGE")
wNotes=Request("NOTES")
```

Для того чтобы инициализировать объект DOM, вы можете использовать соответствующий синтаксис, с которым вы познакомились в предыдущих уроках настоящей книги. В данном случае он выглядит следующим образом:

```
Set oWineNotesDOM = Server.CreateObject("Microsoft.XMLDOM")
oWineNotesDOM.async = false
```

Загрузить документ XML можно с помощью следующей команды:

```
oWineNotesDOM.load server.mappath("wine_notes.xml")
```

Вам необходимо "выявить" все ошибки обработки, например, ссылки на несуществующие документы XML или ошибки ввода-вывода. Для этого вы можете использовать следующую команду:

```
If oWineNotesDOM.parseError.ErrorCode <> 0 Then
```

В случае возникновения ошибки сценарий создаст пустой документ XML с помощью следующей команды:

```
oWineNotesDOM.loadXML "<wine_notes/>"
End If
```

Сценарий будет задавать метку даты и времени в момент сохранения записи, добавляя код, который содержит атрибут даты, для чего используется метод `now()`:

```
Set oDateStamp = oWineNotes-
DOM.documentElement.AppendChild(oWineNotesDOM.createElement("wine"))
oDateStamp.setAttribute "date", now()
```

Для каждого из узлов документов (*elementNameXML*) в документе XML вы передаете данные формы новому узлу элемента (*oMyObjectName*) в результирующей древовидной структуре. В каждом случае синтаксис имеет следующий вид:

```
Set oMyObjectName = oMyDateStampObject-
Name.appendChild(oMyObjectName.createElement("elementNameXML"))
oMyObjectName.Text = MyElementName
```

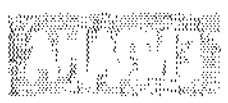
Вам необходимо повторить этот фрагмент кода для каждого из узлов, которые должны быть созданы. В нашем случае это `house`, `region`, `type`, `vintage` и `notes`. В случае элемента `notes` вам следует добавить метод `createCDATASection()`. Полный сценарий приведен в листинге 21.5.

Листинг 21.5. Сценарий ASP для добавления записей, который выполняется на сервере, `add_notes.asp`

```
1:  <HTML>
2:    <link rel = "stylesheet" type = "text/css"
3:     href = "wine_notes.css">
4:  <%
5:  ' Листинг 21.5 - add_notes.asp
6:
7:    Dim wHouse
8:    Dim wRegion
9:    Dim wType
10:    Dim wVintage
11:    Dim wNotes
12:
13:    Dim oWineNotesDOM
```

```
14:    Dim oWineNotesRootNode
15:    Dim oDateStamp
16:    Dim oWineNotesNodes
17:
18:    wHouse=Request("HOUSE")
19:    wRegion=Request("REGION")
20:    wType=Request("TYPE")
21:    wVintage=Request("VINTAGE")
22:    wNotes=Request("NOTES")
23:
24:    Set oWineNotesDOM = Server.CreateObject("Microsoft.XMLDOM")
25:    oWineNotesDOM.async = false
26:
27:    oWineNotesDOM.load server.mappath("wine_notes.xml")
28:
29:    If oWineNotesDOM.parseError.ErrorCode <> 0 Then
30:
31:       oWineNotesDOM.loadXML "<wine_notes/>"
32:    End If
33:
34:    Set oDateStamp = oWineNotesDOM.documentElement.AppendChild
35:        (oWineNotesDOM.createElement("wine"))
36:    oDateStamp.setAttribute "date", now()
37:
38:
39:    Set oWineNotesNodes = oDateStamp.appendChild
40: (oWineNotesDOM.createElement("house"))
41:    oWineNotesNodes.Text = wHouse
42:
43:
44:    Set oWineNotesNodes = oDateStamp.appendChild
45: (oWineNotesDOM.createElement("region"))
46:    oWineNotesNodes.Text = wRegion
47:
48:
49:    Set oWineNotesNodes = oDateStamp.appendChild
50: (oWineNotesDOM.createElement("type"))
51:    oWineNotesNodes.Text = wType
52:
53:
54:    Set oWineNotesNodes = oDateStamp.appendChild
55: (oWineNotesDOM.createElement("vintage"))
56:    oWineNotesNodes.Text = wVintage
57:
58:
59:
60:    Set oWineNotesNodes = oDateStamp.appendChild
61: (oWineNotesDOM.createElement("notes"))
```

```
62:    oWineNotesNodes.appendChild oWineNotes
       DOM.createCDATASection(wNotes)
63:
64:
65:
66:    oWineNotesDOM.Save Server.MapPath("wine_notes.xml")
67: %>
68: <BR/><BR/><BR/>
69: <H1>Thank You</H1>
70:
71: <P>
72: <B>Your Wine Notes about : <%= sWine %></B><BR/>
73: </P>
74: <PRE>
75: <%= wNotes %>
76: </PRE>
77:
78: <P><B>Have been entered</P>
79:
80: <P>Click <A href="view_notes.asp">Here</A> To View Wine Notes</P>
81:
82: </HTML>
```



В строках 7–16 определяются переменные, необходимые при работе сценария. Переменные заголовков HTTP передаются в результате запросов в строках 18–22. Объект DOM (`oWineNotesDOM`) создается в строке 24, а документ XML (`wine_notes.xml`) передается этому объекту в строке 27. В случае успешного выполнения сценария задается метка времени и даты. Эта метка задается с помощью атрибута `date` элемента `wine` (строки 34–36). Каждый из новых узлов элементов (`house`, `region`, `type`, `vintage` и `notes`) создается в строках 38, 43, 48, 53 и 58 соответственно. Элемент `notes` дальше определяется с помощью метода `createCDATASection (wNotes)` в строке 62. Свойство `save` объекта DOM выполняется в строке 66 для сохранения измененного документа `wine_notes.xml`. Строки 69–82 содержат минимальные сведения, необходимые для отображения на экране сообщения об успешном выполнении сценария.

Отображение сведений с помощью сценария view_notes.asp

Для получения списка всех отзывов, содержащихся в документе `wine_notes.xml`, нужно использовать сценарий, который будет выполнять на сервере преобразования XSL (`wine_notes.xsl`). Вы можете сделать это, инициализировав два объекта XML DOM — один для документа XML и другой для документа XSL. После этого вы завершите создание всей древовидной структуры результатов, используя команду, подобную приведенной ниже:

```
Response.Write oWineNotesXML.TransformNode(oWineNotesXSL)
```

Полный вариант сценария приведен в листинге 21.6.

Листинг 21.6. Выполнение преобразований XSLT на стороне сервера `view_notes.asp`

```
1:  <%
2:  ' Листинг 21.6 - view_notes.asp
3:
4:    Dim oWineNotesXML       ' XML document
5:    Dim oWineNotesXSL       ' XSL document
6:
7:  ' Instantiate the XML DOM Object for the wine_notes.xml file
8:    Set oWineNotesXML = Server.CreateObject("Microsoft.XMLDOM")
9:    oWineNotesXML.async = False
10:
11: ' Load the wine_notes.xml file into the DOM object
12:   oWineNotesXML.Load Server.MapPath("wine_notes.xml")
13:
14: ' Create the XML DOM Object for the wine_notes.xsl Stylesheet
15:   Set oWineNotesXSL = Server.CreateObject("Microsoft.XMLDOM")
16:   oWineNotesXSL.async = False
17:
18: ' Load the wine_notes.xsl file into the Style DOM object
19:   oWineNotesXSL.load Server.MapPath("wine_notes.xsl")
20:
21: ' Write the transformed document out to the browser
22:   Response.Write oWineNotesXML.TransformNode(oWineNotesXSL)
23: %>
```



Необходимые новые переменные объектов объявляются в строках 4 и 5. Первый объект DOM (`oWineNotesXML`) создается на сервере в строке 8. Этот объект, который содержит документ `wine_notes.xml`, загружается в строке 12. Эти же действия повторяются и для файла XSL в строках 15 и 19. В строке 22 полученный результат отображается на экране.

Преобразование содержания с помощью документа wine_notes.xsl

Преобразования, о которых мы упоминали при рассмотрении сценария ASP, определяются с помощью документа `wine_notes.xsl`. В большинстве ситуаций преобразования XSLT заключают содержание элементов и атрибутов XML в соответствующие дескрипторы HTML, чтобы представить их в окне броузера.

Если вы решили использовать каскадную таблицу стилей CSS, воспользуйтесь дескриптором `<LINK>`:

```
<LINK rel = "stylesheet" type = "text/css"
        href = "wine_notes.css"/>
```

В преобразовании XSLT используется один шаблон для корневого элемента документа XML, после чего размещает соответствующие дескрипторы HTML в результирующем документе. Соответствующий шаблон может выглядеть следующим образом:

```
<xsl:template match ="/">
```

Создайте HTML-таблицу, используя заголовок, занимающий ширину всего одной ячейки (<TD>).

Использование элемента xsl:for-each для выбора каждого элемента-контейнера wine в экземпляре XML приводит к созданию еще одной строки в таблице. Вы также можете использовать атрибут order-by для сортировки записей. В данном случае отсортируйте по метке даты и времени, созданной выполненным сценарием. Соответствующая инструкция может выглядеть следующим образом:

```
<xsl:for-each select="wine_notes/wine" order-by="-wine[@date]">
```

Полный вариант документа XSLT приведен в листинге 21.7. Создайте документ и сохраните его под именем wine_notes.xsl.

Листинг 21.7. Преобразования XSLT приложения Wine Notes — wine_notes.xsl

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

<!-- Листинг 21.7 - wine_notes.xsl -->

<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">

  <xsl:template match="/">

  <HTML>
    <HEAD>
      <TITLE>Online Wine Testing Notes</TITLE>
      <LINK rel = "stylesheet" type = "text/css"
            href = "wine_notes.css"/>
    </HEAD>
    <BODY>
      <H1>Online Wine Tasting Notes</H1>
     <TABLE border = "1" width="100%">
      <TR class="head">
        <TD class="head">Tasting Notes</TD>
      </TR>
      <TR>
        <TD>
          <TABLE width="100%">

            <xsl:for-each select="wine_notes/wine" order-by="-
            wine[@date]">
              <TR>
                <TD width = "100%">
                  <B>Entered: </B> <xsl:value-of se-
                  lect="@date"/><BR/>
                  <B>House:   </B> <xsl:value-of se
                 lect="house"/><BR/>
                  <B>Region:  </B> <xsl:value-of
                  select="region"/><BR/>
                  <B>Type:    </B> <xsl:value-of se-
                  lect="type"/><BR/>
```

```
32:                     <B>Vintage: </B> <xsl:value-of
                        select="vintage"/><BR/>
33:                     <B>Notes:   </B> <PRE><SPAN>
34:                     <xsl:value-of select="notes"/></SPAN></PRE><BR/>
35:                      <HR/>
36:                   </TD>
37:                 </TR>
38:               </xsl:for-each>
39:
40:             </TABLE>
41:           </TD>
42:         </TR>
43:       </TABLE>
44:     </BODY>
45:   </HTML>
46:
47:   </xsl:template>
48:
49: </xsl:stylesheet>
```



В строке 7 определяется шаблон, совпадающий с корневым узлом (`<xsl:-template match="/">`). Как правило, инструкции HTML помещаются в результирующее дерево (строки 9—23). Таблица стилей CSS, связанная с документом HTML, создается после преобразований XSLT в строках 12 и 13. Каждая из записей выбирается и сортируется по дате и времени (строка 25) (`<xsl:for-each select="wine_notes/wine" order-by="-wine[@date]">`). Содержание каждого узла атрибута или элемента в исходном документе передается в результирующий документ. Содержание элемента `notes` задается с помощью дескрипторов HTML `<PRE>`.

Создание Web-узла

Создав все необходимые файлы, сохраните их в соответствующей папке на Web-сервере. Если вы используете IIS, можно создать виртуальный общий Web-ресурс на основе папки, в которой содержатся файлы, как вы уже поступали в предыдущих уроках. Выполните страницу `index.html` на Web-сервере. В случае IIS команда выглядит следующим образом.

`http://localhost/`*`виртуальный_каталог`*`/index.html`

HTML-форма для ввода данных должна выглядеть приблизительно так, как показано на рис. 21.2, а список узлов —на рис. 21.3.

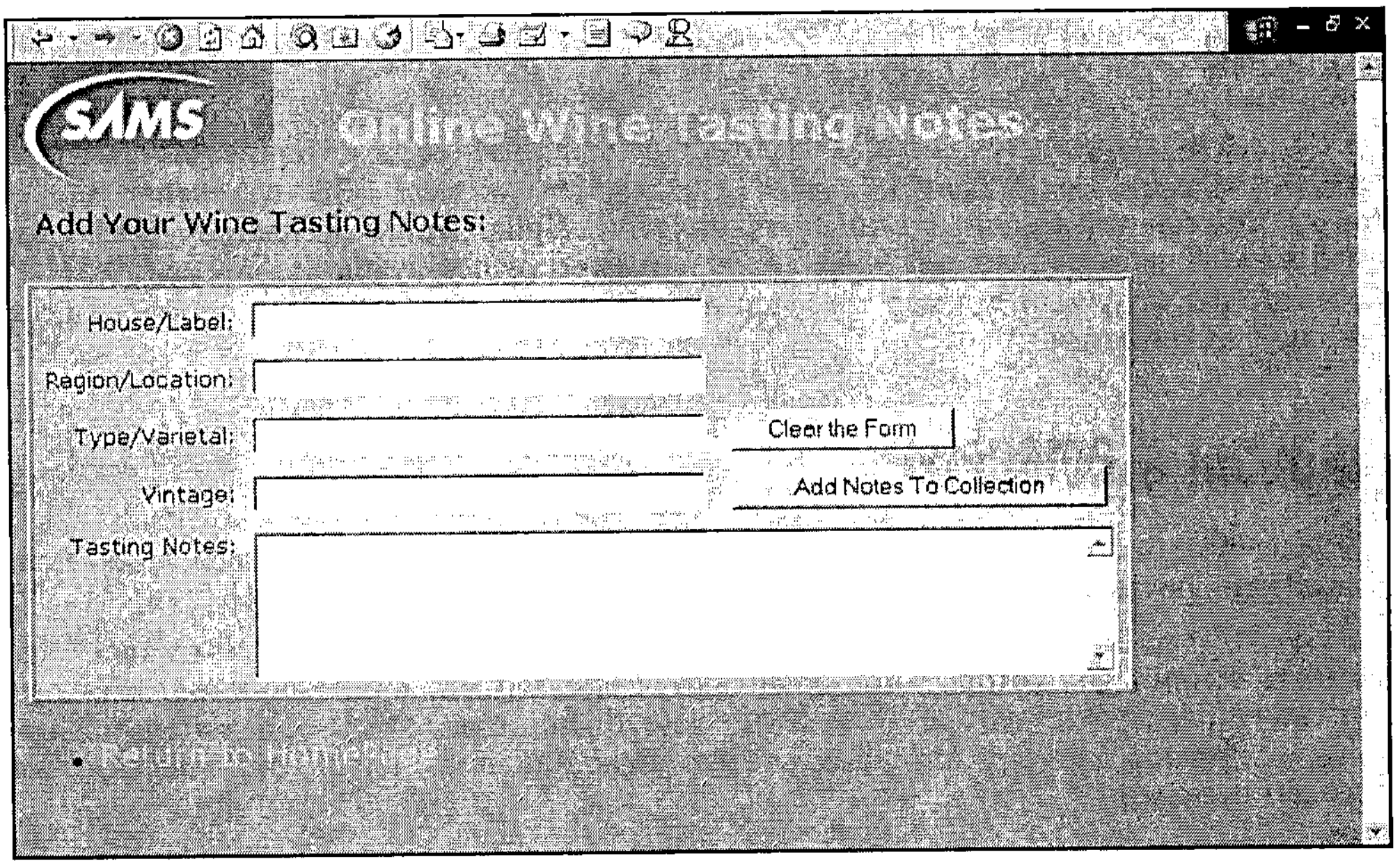


Рис. 21.2. Интерактивная форма для ввода данных

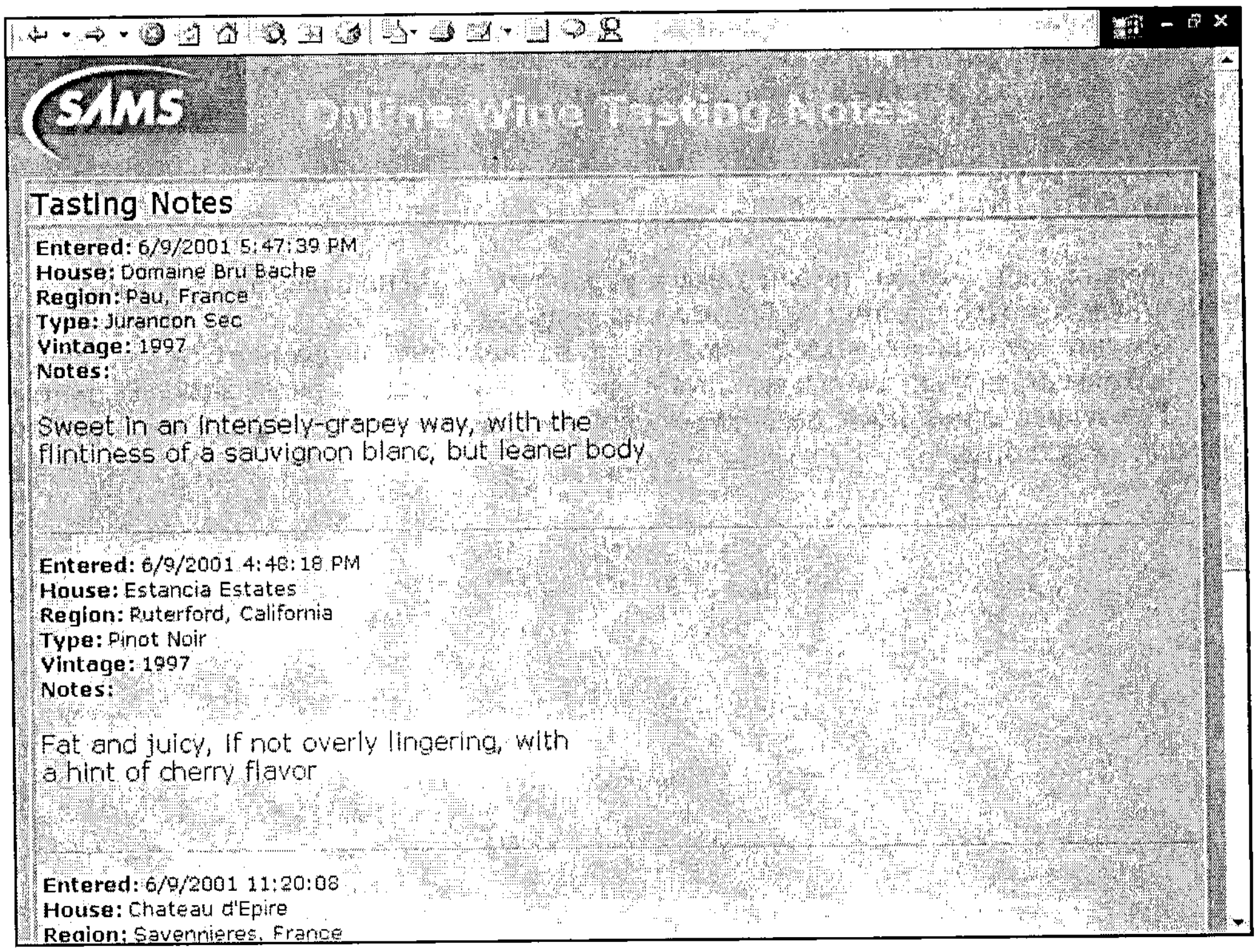


Рис. 21.3. Список всех отзывов, содержащихся в наборе данных XML

Резюме

Сегодня вы буквально с нуля создали целое приложение, в котором используются технологии XML, преобразования XSLT, каскадная таблица CSS, объектная модель XML DOM, а также смежные с ними технологии. Приложение предлагает форму, которую можно использовать для добавления данных в существующий документ XML. В результате вы получили пример того, каким образом могут совместно использоваться несколько технологий.

Вопросы и ответы

Что необходимо сделать для того, чтобы созданное приложение не зависело от используемого броузера?

Вам следует изменить каскадную таблицу стилей таким образом, чтобы в ней были только те стили CSS, которые поддерживаются броузерами, с помощью которых осуществляется доступ к узлу. XML и XSL будут функционировать при использовании любых броузеров, поддерживающих соответствующие технологии, без внесения каких-либо изменений?

Назовите несколько других типов приложений, которые можно создать с использованием подобной архитектуры.

С помощью преобразований и объектов, рассмотренных сегодня можно создать целый ряд приложений для сбора данных. Например, вы можете создать приложение, которое позволит ученым обмениваться своими соображениями и гипотезами относительно решения тех или иных научных проблем.

Упражнение

Проверьте свои знания, полученные в результате изучения настоящего урока. Ответ вы найдете в Приложении А, "Ответы на вопросы".

Расширьте функциональные возможности приложения, созданного в настоящем уроке, обеспечив поиск, используя JavaScript или VBScript, а также острова данных XML. Проведите поиск имен по полю House/Label, после чего выведите на экран полученные результаты.

Часть IV

SAMS Освой самостоятельно

Приложения

А. Ответы на вопросы

Б. Ресурсы

Приложение А

Ответы на вопросы

Возможное решение для упражнения дня 1-го

```
<exercise day="1">
    <name>Devan Shepherd</name>
    <address>123 any street</address>
    <email>me@whatever.com</email>
    <dob month="August" year="1958" day="11">
</exercise>
```

Возможное решение для упражнения дня 2-го

Экземпляр, приведенный в листинге 2.4, обладает целым рядом недостатков. К ним относятся следующие.

- Зарезервированное слово XML в строке объявления должно указываться символами только нижнего регистра.
- Значение 1.0 версии атрибута в инструкции должно заключаться в кавычки.
- В документе отсутствует уникальный корневой элемент.
- Несколько примеров значений атрибутов не указаны должным образом.
- Первый элемент `<cd>` не содержит закрывающего дескриптора.
- Элементы `<tracks...>` являются пустыми и должны быть завершены.
- К некоторым дескрипторам необходимо добавить угловые скобки.
- В некоторых парах открывающего и закрывающего дескрипторов используются символы различного регистра.

После исправления всех перечисленных ошибок документ становится правильным и выглядит так, как показано в листинге А.1.

Листинг А.1. Решение для упражнения дня 2-го

```
1:  <?xml version="1.0"?>
2:  <music>
3:   <cd number="432">
4:   <title>The Best of Van Morrison</title>
5:   <artist>Van Morison</artist>
6:   <tracks total="20" />
7:   </cd>
8:   <cd number="97">
9:   <title>HeartBreaker</title>
10:  <subtitle>Sixteen Classic Performances</subtitle>
11:  <artist>Pat Benatar</artist>
12:  <tracks total="16" />
13:  </cd>
14:  </music>
```

Возможное решение для упражнения дня 3-го

Возможный вариант древовидной диаграммы для языка разметки MCML показан на рис. А.1.

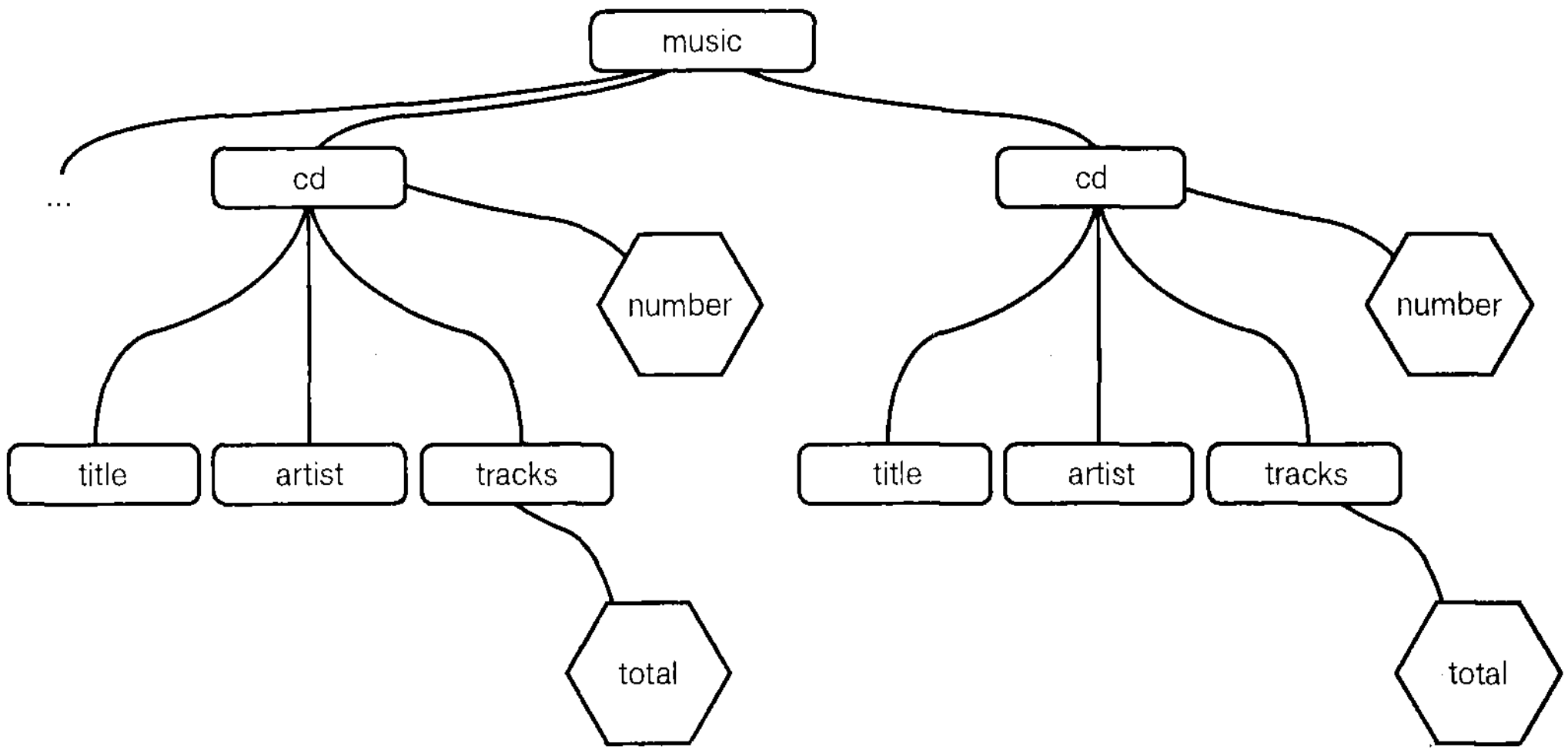


Рис. А.1. Возможное решение упражнения из урока 3: древовидная диаграмма для языка разметки MCML

К документам MCML применяются следующие правила.

- Корневой элемент называется `music` и содержит один или несколько элементов `cd`.
- Элемент `cd` содержит по одному элементу `title`, `artist` и `track`, причем в строго указанном порядке.
- Элемент `cd` содержит обязательный элемент `number`, значение которого должно быть целым числом.
- Элемент `tracks` содержит обязательный элемент `total`, значение которого должно быть целым числом.

Возможное решение для упражнения дня 4-го

В листинге А.2 приведена схема DTD для проверки экземпляров MCML.

Листинг А.2. Решение для упражнения дня 4-го

```
<!ELEMENT music ( cd+ ) >
<!ELEMENT cd ( artist | subtitle | title | tracks )* >
<!ELEMENT artist ( #PCDATA ) >
<!ELEMENT subtitle ( #PCDATA ) >
<!ELEMENT title ( #PCDATA ) >
<!ELEMENT tracks EMPTY >
<!ATTLIST cd
        number NMTOKEN #REQUIRED >
<!ATTLIST tracks
        total NMTOKEN #REQUIRED >
```

Возможное решение для упражнения дня 5-го

В листинге А.3 приведена схема XDR для проверки экземпляров MCML.

Листинг А.3. Схема XDR для языка MCML

```
1:  <?xml version="1.0" encoding="UTF-8"?>
2:  <Schema
3:   name="Untitled-schema"
4:   xmlns="urn:schemas-microsoft-com:xml-data"
5:   xmlns:dt="urn:schemas-microsoft-com:datatypes">
6:   <ElementType name="music" model="closed" content="eltOnly" order="seq">
7:       <element type="cd" minOccurs="1" maxOccurs="*"/>
8:   </ElementType>
9:   <ElementType name="cd" model="closed" content="eltOnly" order="one">
10:      <AttributeType name="number" dt:type="string" required="yes"/>
11:      <attribute type="number"/>
12:      <group minOccurs="0" maxOccurs="*" order="one">
13:          <element type="artist" minOccurs="1" maxOccurs="1"/>
14:          <element type="subtitle" minOccurs="1" maxOccurs="1"/>
15:          <element type="title" minOccurs="1" maxOccurs="1"/>
16:          <element type="tracks" minOccurs="1" maxOccurs="1"/>
17:      </group>
18: </ElementType>
19: <ElementType name="artist" model="closed" content="textOnly"
     dt:type="string"/>
20: <ElementType name="subtitle" model="closed" content="textOnly"
     dt:type="string"/>
21: <ElementType name="title" model="closed" content="textOnly"
     dt:type="string"/>
22: <ElementType name="tracks" model="closed" content="empty">
```

```
23:      <AttributeType name="total" dt:type="number" required="yes"/>
24:      <attribute type="total"/>
25: </ElementType>
26: </Schema>
```

Возможное решение для упражнения дня 6-го

В листинге А.4 приведена схема XSD для проверки экземпляров MCML.

Листинг А.4. Схема XSD для языка MCML

```
1: <?xml version="1.0" encoding="UTF-8"?>
2: <xsd:schema xmlns:xsd="http://www.w3.org/2000/10/XMLSchema">
3:  <xsd:element name="artist" type="xsd:string"/>
4:  <xsd:complexType name="cdType">
5:      <xsd:sequence>
6:          <xsd:element ref="title"/>
7:          <xsd:element ref="subtitle" minOccurs="0"/>
8:          <xsd:element ref="artist"/>
9:          <xsd:element name="tracks" type="tracksType"/>
10:     </xsd:sequence>
11:     <xsd:attribute name="number" use="required">
12:         <xsd:simpleType>
13:             <xsd:restriction base="xsd:NMTOKEN">
14:                 <xsd:enumeration value="432"/>
15:                 <xsd:enumeration value="97"/>
16:             </xsd:restriction>
17:         </xsd:simpleType>
18:     </xsd:attribute>
19: </xsd:complexType>
20: <xsd:element name="music">
21:     <xsd:complexType>
22:         <xsd:sequence>
23:             <xsd:element name="cd" type="cdType" maxOccurs=
                 "unbounded"/>
24:         </xsd:sequence>
25:     </xsd:complexType>
26: </xsd:element>
27: <xsd:element name="subtitle" type="xsd:string"/>
28: <xsd:element name="title" type="xsd:string"/>
29: <xsd:complexType name="tracksType">
30:     <xsd:attribute name="total" use="required">
31:         <xsd:simpleType>
32:             <xsd:restriction base="xsd:NMTOKEN">
33:                 <xsd:enumeration value="16"/>
34:                 <xsd:enumeration value="20"/>
35:             </xsd:restriction>
```

```
36:          </xsd:simpleType>
37:      </xsd:attribute>
38: </xsd:complexType>
39: </xsd:schema>
```

Возможное решение для упражнения дня 7-го

В листинге A.5 приведен документ XML, созданный на MCML, в каждый из элементов `cd` внесены необходимые изменения. В листинге A.6 приведена схема DTD, используемая для проверки документа XML, в ней также объявляются и определяются подстановочные строки.

Листинг A.5. Экземпляр MCML с подстановкой элементов `cd07.xml`

```
1: <?xml version="1.0"?>
2: <!-- листинг 7.3 - cd07.xml-->
3:
4: <!DOCTYPE music SYSTEM "cd.dtd">
5: <music>
6:  <cd number="432">
7:      <title>The Best of Van Morrison</title>
8:      <artist>Van Morison</artist>
9:      <tracks total="20"/>
10:     <style>&style;</style>
11: </cd>
12: <cd number="97">
13:     <title>HeartBreaker</title>
14:     <subtitle>Sixteen Classic Performances</subtitle>
15:     <artist>Pat Benatar</artist>
16:     <tracks total="16"/>
17:     <style>&style;</style>
18: </cd>
19: </music>
```

Листинг A.6. Схема DTD для экземпляра MCML с подстановкой элементов `cd.dtd`

```
1: <!-- листинг 7.4 - cd.dtd-->
2:
3: <!ELEMENT music (cd+)>
4: <!ELEMENT cd (artist | subtitle | title | tracks | style)*>
5: <!ELEMENT artist (#PCDATA)>
6: <!ELEMENT subtitle (#PCDATA)>
7: <!ELEMENT title (#PCDATA)>
8: <!ELEMENT tracks EMPTY>
9: <!ELEMENT style (#PCDATA)>
10: <!ATTLIST cd
11: number NMTOKEN #REQUIRED
12: >
```

```
13: <!ATTLIST tracks
14: total NMTOKEN #REQUIRED
15: >
16: <!ENTITY style "This CD is a compilation
17:       of the artist's top hits and is characteristic
18:       of the style we have come to enjoy."
19: >
```

Возможное решение для упражнения дня 8-го

Возможное решение упражнения приведено в листинге A.7. Пространство имен объявлено в строке 5 с URL, равным `urn:myoldies:LPs`. В строках 17–21 показан новый класс данных, включая префикс пространства имен `vinyl`.

Листинг A.7. Экземпляр MCML с пространством имен `vinyl` — `cd08.xml`

```
1: <?xml version="1.0"?>
2: <!-- Листинг 8.7 - cd08.xml -->
3:
4: <music
5:     xmlns:vinyl="urn:myoldies:LPs">
6:   <cd number="432">
7:       <title>The Best of Van Morrison</title>
8:       <artist>Van Morison</artist>
9:       <tracks total="20"/>
10:   </cd>
11:   <cd number="97">
12:       <title>HeartBreaker</title>
13:       <subtitle>Sixteen Classic Performances</subtitle>
14:       <artist>Pat Benatar</artist>
15:       <tracks total="16"/>
16:   </cd>
17:   <vinyl:record number="105">
18:       <vinyl:title>King of Blue</vinyl:title>
19:       <vinyl:artist>Miles Davis</vinyl:artist>
20:       <vinyl:tracks total="11"/>
21:   </vinyl:record>
22: </music>
```

Возможное решение для упражнения дня 9-го

Возможные решения упражнений этого дня приведены в таблице ниже.

Проблема	*Решение*
Выбрать все дочерние элементы корневого узла	`//*`
Выбрать все элементы, атрибут `tracks` которых равен `16`	`//tracks[@total="16"]`

Проблема	Решение
Выбрать все элементы, в имени которых содержится буква i	`//*contains(name(), "i")`
Выбрать все элементы, в имени которых содержится больше 11 знаков	`//*[string-length(name()>11)]`
Выбрать все элементы одного уровня с первым элементом cd	`//music/cd/following-sibling::*`

Возможное решение для упражнения дня 10-го

Возможное решение упражнения приведено в листинге А.8. Вам потребуется документ `shelf.xml` для подтверждения простой ссылки.

Листинг А.8. Возможное решение для `cd10.xml`

```
1: <?xml version="1.0"?>
2: <!-- Листинг 10.9 - cd10.xml -->
3:
4: <music
5:     xmlns:vinyl="urn:myoldies:LPs"
6:     xmlns:xlink="http://www.w3.org/1999/xlink">
7:   <cd number="432" xlink:type="simple"
8:                                    xlink:href="shelf.xml"
9:                                    xlink:show="replace"
10:                                    xlink:actuate="onRequest">
11:       <title>The Best of Van Morrison</title>
12:       <artist>Van Morison</artist>
13:       <tracks total="20"/>
14:   </cd>
15:   <cd number="97">
16:       <title>HeartBreaker</title>
17:       <subtitle>Sixteen Classic Performances</subtitle>
18:       <artist>Pat Benatar</artist>
19:       <tracks total="16"/>
20:   </cd>
21:   <vinyl:record number="105">
22:       <vinyl:title>King of Blue</vinyl:title>
23:       <vinyl:artist>Miles Davis</vinyl:artist>
24:       <vinyl:tracks total="11"/>
25:   </vinyl:record>
26: </music>
```

В листинге А.9 приведен документ, полученный после активизации простой ссылки XLink.

Листинг А.9. Документ XLink `shelf.xml`

```
1: <?xml version="1.0"?>
2: <!-- Листинг 10.10 - shelf.xml -->
3:
4: <shelf>
5: This CD is located on the third shelf
6: </shelf>
```

Возможное решение для упражнения дня 12-го

В листинге А.10 приведен код для возвращения элементов `title` и `artist` для первого компакт-диска в коллекции.

Листинг А.10. Сценарий DOM для языка MCML — `DOM_12a.xml`

```
1:  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
2:  <!-- Day 12 Exercise 1 - DOM_12a.html -->
3:
4:  <html>
5:  <head>
6:  <title>DOM Scripting</title>
7:  </head>
8:  <body>
9:  <script language="javascript">
10:  <!--
11:
12:     var oMystuff = new ActiveXObject("Microsoft.XMLDOM")
13:     oMystuff.async="false"
14:     oMystuff.load("cd12.xml")
15:
16:
17:   document.write
18:   ("Title:")
19:     document.write
20:   (oMystuff.getElementsByTagName("title").item(0).text)
21:
22:   document.write
23:   ("<br/>Artist:")
24:     document.write
25:   (oMystuff.getElementsByTagName("artist").item(0).text)
26:
27:
28:  -->
29:  </script>
30:
31:  </body>
32:  </html>
```

Сценарий, начиная со строки 9, содержится в разделе `<body>` (строки 8–31) HTML-страницы. В строке 9 объявляется используемый язык сценариев (`language="javascript"`). В строке 10 начинается комментарий, который скрывает сценарий от ранних версий броузеров, которые не поддерживают JavaScript. Строки 12–14 соответствуют созданию объекта документа и загрузке XML. Строки 17–18 выводят на экран содержание элемента `title`. Строка 20 выбирает текстовое данные (`text`) первого (`item(0)`) элемента `title` (`getElementsByTagName("title")`) в объекте документа (`oMystuff`). Точно так же, строка 25 выбирает текстовые данные (`text`) первого (`item(0)`) элемента `artist` в документе XML. Строки 28–30 завершают разделы комментария — `<script>` и `<body>` соответственно.

Упражнение – часть 2

Для выбора вторых элементов `title` и `artist` в документе XML вам необходимо изменить число в методах `item()` в строках 20 и 25 на `1`. Полный код приведен ниже.

Листинг A.11. Сценарий DOM для языка MCML — `DOM_12b.xml`

```
1:  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
2:  <!-- Day 12 Exercise 2 - DOM_12b.html -->
3:
4:  <html>
5:  <head>
6:  <title>DOM Scripting</title>
7:  </head>
8:  <body>
9:  <script language="javascript">
10: <!--
11:
12:    var oMystuff = new ActiveXObject("Microsoft.XMLDOM")
13:    oMystuff.async="false"
14:    oMystuff.load("cd12.xml")
15:
16:
17:  document.write
18:  ("Title:")
19:    document.write
20:  (oMystuff.getElementsByTagName("title").item(1).text)
21:
22:  document.write
23:  ("<br/>Artist:")
24:    document.write
25:  (oMystuff.getElementsByTagName("artist").item(1).text)
26:
27:
28: -->
29: </script>
30:
31: </body>
32: </html>
```

Значения, передаваемые методам `item()` в строках 20 и 25, изменены на `1` для выбора второго экземпляра каждого из элементов. На экране отображается текстовое содержание каждого из элементов.

Возможное решение для упражнения дня 13-го

Вам необходимо изменить значение для проверки в строке 27 из листинга 13.2 на `true`, после чего перекомпилировать результат.

```
27:          saxFactory.setValidating( true);
```

Поскольку вы изменили имя класса в программе `EList2.java`, вам также необходимо изменить ссылки на имя класса в строках 14, 19 и 32, чтобы откомпилировать программу.

Для обнаружения ошибок проверки, если таковые существуют, вам необходимо заменить метод `error` класса `HandlerBase` и заставить обработчик прервать работу и сообщить об ошибке, вместо того чтобы продолжать обработку документа.

Ниже приведены два новых метода:

```
27: public void error(SAXParseException minor)
28: throws SAXParseException
29: {
30: throw minor;
31: }
32:
33: public void warning(SAXParseException warn)
34: throws SAXParseException
35: {
36: System.err.println( "Warning: " + warn.getMessage() );
```



Первый новый метод (строки 1–5) отлавливает любые ошибки обработки, возникающие при выполнении метода `main`. Второй метод (строки 6–10) отображает предостережения, полученные от синтаксического анализатора SAX.

Когда вы откомпилируете программу, в которую внесены все перечисленные изменения, и измените документ `cd.xml`, добавив пустой элемент в конце экземпляра, вы получите следующий результат:

```
Name of Document: file:C:/SAX/cd13_bad.xml
SAX Event - Start of Document
SAX Event - Element Start: music
SAX Event - Element Start: cd
SAX Event - Attribute: number="432"
SAX Event - Element Start: title
SAX Event - Characters: The Best of Van Morrison
SAX Event - Element End: title
SAX Event - Element Start: artist
SAX Event - Characters: Van Morrison
SAX Event - Element End: artist
SAX Event - Element Start: tracks
SAX Event - Attribute: total="20"
SAX Event - Element End: tracks
SAX Event - Element End: cd
SAX Event - Element Start: cd
SAX Event - Attribute: number="97"
SAX Event - Element Start: title
SAX Event - Characters: HeartBreaker
```

```
SAX Event - Element End: title
SAX Event - Element Start: subtitle
SAX Event - Characters: Sixteen Classic Performances
SAX Event - Element End: subtitle
SAX Event - Element Start: artist
SAX Event - Characters: Pet Benatar
SAX Event - Element End: artist
SAX Event - Element Start: tracks
SAX Event - Attribute: total="20"
SAX Event - Element End: tracks
SAX Event - Element End: cd
Parser Error: Element "music" does not allow "empty" here.
```

Ошибка (новый пустой элемент, наличие которого не допускается в соответствии со схемой DTD) указана для последней строки, после чего работа синтаксического анализатора прекращается.

Обратите внимание, что обработчик не отлавливает сведения о лишних символах. Это связано с тем, что для определения данных используется схема DTD, которая не налагает ограничения на количество пробелов.

Возможное решение для упражнения дня 14-го

Вы можете использовать несколько методов для достижения эффектов, приведенных на рис. 14.3. Возможный вариант документа CSS приведен в листинге А.12.

Листинг А.12. Возможное решение для упражнения дня 14-го

```
1:  * Challenge, Day 14, cd14.css */
2:
3:  music
4:        {display:block;
5:         font-size:14pt;
6:         padding:60px;
7:         border-style:solid;
8:            background-color:#999966;
9:            background-image:URL(http://www.samspublishing.com/images/
                                   samslogo.gif);
10:           background-repeat:no-repeat}
11:
12: title, artist
13:       {display:block}
14:
15: cd
16:       {display:block;
17:        padding:20px;
18:           border-style:double;
19:           border-width:thick;
20:           width:75%;
21:           margin:20px;
22:           background-color:#99cc99}
```

АНАЛИЗ Строки 3–10 определяют набор правил для корневого элемента `music`. Содержание корневого элемента `music` отображается в виде блоков (строка 4). Размер текста равен 14 пунктам (строка 5). Расстояние между внешней границей и внешними границами внутренних окон равно 60 пикселям (строка 6). Внешняя граница отображается сплошной линией (строка 7). Цвет фона задан равным HEX999966, что соответствует цветовому заполнению логотипа издательства *Sams Publishing* (строка 8). Доступ к изображению осуществляется по указанному адресу URL (строка 9). Свойство `background-repeat` задано равным `no-repeat` (строка 10), что позволяет избежать отображения графического изображения поверх текста. Строки 12 и 13 обеспечивают перевод строки для разделения элементов `title` и `artist`. Для содержания элемента `cd` определен набор правил в строках 15–22. Для задания пространства между словами и каждым из внутренних прямоугольников, предусмотрен отступ, равный 20 пикселям (строка 17). Внутренние прямоугольники отображены тонкой двойной линией (строки 18 и 19). В строке 20 задается 75%-ное отношение по горизонтали между размерами внешнего и внутренних прямоугольников. Цвет фона для внутренних прямоугольников задан равным HEX99cc99 (строка 22).

Возможное решение для упражнения дня 15-го

Простейший способ создания маркированного списка — использование элемента `fo:list-block` для содержания всех элементов списка. Каждый элемент списка задается парой `label/body`. Элемент `label` определяет маркер, а `body` — текстовые данные, которые отображаются после него. Один из способов получения необходимых результатов приведен в листинге A.13.

Листинг A.13. Возможное решение для упражнения дня 15-го

```
1: <?xml version="1.0"?>
2: <!-- Day 15 Challenge - cd15.fo -->
3:
4: <fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">
5:
6:   <fo:layout-master-set>
7:     <fo:simple-page-master master-name="CD">
8:       <fo:region-body margin-left="1.5in"
9:              margin-right="1.5in" margin-top="1in"/>
10:     </fo:simple-page-master>
11:   </fo:layout-master-set>
12:
13:   <fo:page-sequence master-name="CD">
14:     <fo:flow flow-name="xsl-region-body">
15:
16:       <fo:block
17:          text-align="center"
18:          font-size="28pt"
19:          font-family="sans-serif"
20:          font-weight="bold"
21:          color="blue">CD Titles
```

```
22:         </fo:block>
23:
24:       <fo:list-block>
25:
26:         <fo:list-item>
27:
28:           <fo:list-item-label>
29:             <fo:block>*</fo:block>
30:           </fo:list-item-label>
31:
32:           <fo:list-item-body>
33:             <fo:block start-indent=".25in">
34:                 The Best of Van Morrison</fo:block>
35:           </fo:list-item-body>
36:
37:         </fo:list-item>
38:         <fo:list-item>
39:
40:           <fo:list-item-label>
41:             <fo:block>*</fo:block>
42:           </fo:list-item-label>
43:
44:           <fo:list-item-body>
45:             <fo:block start-indent=".25in">
46:                 HeartBreaker</fo:block>
47:           </fo:list-item-body>
48:
49:         </fo:list-item>
50:
51:       </fo:list-block>
52:
53:     </fo:flow>
54:   </fo:page-sequence>
55:
56: </fo:root>
```

Элемент `fo:list-block` начинается со строки 24. Каждый элемент списка (`fo:list-item`) содержит два дочерних элемента — `fo:list-item-label` и `fo:list-item-body`. Элементы `fo:list-item-label` задают используемые в списке маркеры (звездочки). Элементы `fo:list-item-body` содержат названия компакт-дисков. В элементах `fo:list-item-body` присутствует атрибут `start-indent`, который задает расстояние между маркером и данными элемента списка.

Возможное решение для упражнения дня 16-го

Один из способов решения упражнения состоит в создании двух простых шаблонов. Первый шаблон касается обычных дескрипторов HTML. Этот шаблон совпадает с корневым элементом (`music`) корневого узла. Второй шаблон совпадает с элементом `cd` и использует элемент `xsl:number` со значением `xsl:apply-templates`. Полный вариант решения приведен в листинге А.14.

Листинг А.14. Возможное решение для cd16.html

```
1:  <?xml version="1.0"?>
2:  <!-- Day 16 - Challenge - cd16.xsl -->
3:
4:  <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
5:                  version="1.0">
6:
7:      <xsl:template match="music">
8:          <HTML>
9:          <HEAD>
10:          <TITLE>CD Collection</TITLE>
11:          </HEAD>
12:          <BODY>
13:              <xsl:apply-templates/>
14:          </BODY>
15:          </HTML>
16:      </xsl:template>
17:
18:      <xsl:template match="cd">
19:              <xsl:number/>. <xsl:apply-templates select=
                 "title"/><br/>
20:      </xsl:template>
21:
22:
23: </xsl:stylesheet>
```

Строки 18–20 содержат шаблон, который выполняет подсчет элементов. Этот шаблон применяется к элементу `cd` (строка 18). Значение элемента `xsl:number` увеличивается при каждом выполнении шаблона. Поскольку шаблон выполняется для каждого элемента `cd`, мы подсчитываем количество компакт-дисков в коллекции. Благодаря использованию атрибута `select` в результирующий документ будут включены только названия компакт-дисков, т. е. элементы `title` (строка 19).

Возможное решение для упражнения дня 17-го

Одно из возможных решений заключается в использовании DSO-аплета. Полный вариант решения приведен в листинге А.15.

Листинг А.15. Возможное решение для cd17.html

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
 "http://www.w3.org/TR/html4/loose.dtd">
<!-- листинг 17.7 cd17.html -->

<HTML>
<HEAD>
<TITLE>XML Data Island</TITLE>
</HEAD>

 <BODY>
 <applet code="com.ms.xml.dso.XMLDSO.class"
         width="100%" height="50" id="CDs">
    <PARAM NAME="url" VALUE="cd17.xml">
 </applet>
 <H1>My Messages</H1>
  <TABLE id="table" border="6" width="100%"
          datasrc="#CDs" summary="messages">
        <THEAD style="background-color: aqua">
              <TH>Title</TH>
              <TH>Artist</TH>
        </THEAD>
        <TR valign="top" align="center">
             <TD><SPAN datafld="title"/></TD>
             <TD><SPAN datafld="artist"/></TD>
      </TR>
   </TABLE>
</BODY>
</HTML>
```

Возможное решение для упражнения дня 21-го

Начните с создания острова данных XML для документа `wine_notes.xml`. Из урока 17 вы помните, что при этом используется следующий синтаксис:

```
<XML ID="oMyDSOObjectName" SRC="mySource.xml"/>
```

Запросите необходимые сведения у пользователей. Сохраните введенную строку текста в качестве значения переменной, которую вы передадите функции поиска. Используйте VBScript или JavaScript для поиска в содержании элемента `house` в документе `wine_notes.xml`, а также для присвоения записи новой переменной при обнаружении совпадения. Отформатируйте данные, которые присвоены переменной, должным образом. Пример функции поиска, написанной на JavaScrpipt, приведен ниже:

```
function fSearchNotes ()
         {
         strHouse = SearchText.value.toUpperCase();
         if (strHouse == "")
            {
```

```
    strSearchResult.innerHTML = "&ltPlease enter a search "
                              + "string into "
                              + "'Produce/House Name' box.&gt";
    return;
    }

oWineNotesDSO.recordset.moveFirst();

RSSearchResultOutput = "";
while (!oWineNotesDSO.recordset.EOF)
    {
    RSstrHouse = oWineNotesDSO.recordset("house").value;

    if (RSstrHouse.toUpperCase().indexOf(strHouse)
       >=0)
       RSSearchResultOutput += "<P><B>"
                    + oWineNotesDSO.recordset("house")
                    + "</B>, "
                    + oWineNotesDSO.recordset("region")
                    + ", <I>"
                    + oWineNotesDSO.recordset("type")
                    + "</I>, "
                    + oWineNotesDSO.recordset("vintage")
                    + ", <BR/>"
                    + oWineNotesDSO.recordset("notes")
                    + "</P>";

       oWineNotesDSO.recordset.moveNext();
       }

    if (RSSearchResultOutput == "")
       strSearchResult.innerHTML = "&ltSorry, no Wine Tasting"
                         + "Notes were found&gt";
    else
       strSearchResult.innerHTML = RSSearchResultOutput;
    }
```

Полный код страницы приведен в листинге А.16.

Листинг А.16. Функция поиска, использующая XML Data Island — `wine_search.html`

```
1: <!DOCTYPE XHTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
2:  "http://www.w3.org/TR/html4/loose.dtd">
3:
4: <!-- Day 21 Exercise - wine_search.html -->
5:
6: <HTML>
7:
8: <HEAD>
9:
```

```
10:      <TITLE>Wine Notes</TITLE>
11:      <LINK rel = "stylesheet" type = "text/css"
12:            href = "wine_notes.css"/>
13:
14: </HEAD>
15:
16: <BODY>
17:
18:     <XML ID="oWineNotesDSO" SRC="wine_notes.xml"/>
19:
20:     <H1>Online Wine Tasting Notes</H1>
21:     <H3>Search For Notes By Producer's House/Label</H3>
22:
23:     Enter characters from the Producer/House Name: <BR/>
24:     <INPUT TYPE="TEXT" ID="SearchText" SIZE="50"> 
25:     <BUTTON ONCLICK='fSearchNotes()'>Search</BUTTON>
26:     <HR>
27:     Results:<P>
28:     <DIV ID=strSearchResult></DIV>
29:
30:     <SCRIPT LANGUAGE="JavaScript">
31:        function fSearchNotes ()
32:           {
33:           strHouse = SearchText.value.toUpperCase();
34:           if (strHouse == "")
35:              {
36:              strSearchResult.innerHTML = "&ltPlease enter a search "
37:                                        + "string into "
38:                                        + "'Produce/House Name' box.&gt";
39:              return;
40:              }
41:
42:           oWineNotesDSO.recordset.moveFirst();
43:
44:           RSSearchResultOutput = "";
45:           while (!oWineNotesDSO.recordset.EOF)
46:              {
47:              RSstrHouse = oWineNotesDSO.recordset("house").value;
48:
49:              if (RSstrHouse.toUpperCase().indexOf(strHouse)
50:                 >=0)
51:                 RSSearchResultOutput += "<P><B>"
52:                                      + oWineNotesDSO.recordset("house")
53:                                      + "</B>, "
54:                                      + oWineNotesDSO.recordset("region")
55:                                      + ", <I>"
56:                                      + oWineNotesDSO.recordset("type")
57:                                      + "</I>, "
```

```
58:                                 + oWineNotesDSO.recordset("vintage")
59:                                 + ", <BR/>"
60:                                 + oWineNotesDSO.recordset("notes")
61:                                 + "</P>";
62:
63:             oWineNotesDSO.recordset.moveNext();
64:             }
65:
66:          if (RSSearchResultOutput == "")
67:             strSearchResult.innerHTML = "&ltSorry, no Wine
Tasting"
68:                                       + "Notes were found&gt";
69:             else
70:                strSearchResult.innerHTML = RSSearchResultOutput;
71:             }
72:     </SCRIPT>
73:     <UL>
74:       <LI><A HREF="index.html">Return to HomePage</A></LI>
75:     </UL>
76:
77: </BODY>
78:
79: </HTML>
```



В строке 18 определяется остров данных в документе HTML, связанном с документом `wine_notes.xml`. В строках 23–25 создается поле для ввода данных, в котором пользователь может ввести строку поиска. Строка, которая вводится, называется `SearchText`. После того как пользователь введет строку поиска, вызывается функция `fSearchNotes()` (строка 31). Функция поиска просматривает содержание элемента `house` (строка 47), после чего возвращает целую запись в виде набора записей (строка 44) при обнаружении совпадения со строкой поиска `SearchText`. Полученная строка форматируется (строки 51–61) и отправляется броузеру для отображения. Если пользователь забыл ввести данные перед тем, как отправить форму, строки 34–40 выведут сообщение об ошибке. Если в результате поиска также не найдено никаких данных, строки 66–68 также выведут сообщение об ошибке. Строка 74 "возвращает" пользователя к главной странице приложения.

Приложение Б

Ресурсы

Несмотря на то, что настоящая книга была выпущена в достаточно сжатые сроки, некоторые стандарты наверняка претерпели изменения даже на протяжении того времени, пока вы читали книгу, а другие стандарты разработали. Поэтому приведенный ниже список Web-узлов окажется для вас ценным источником дополнительных сведений об изложенном в книге *Освой самостоятельно XML за 21 день, 2-е издание.*

Некоторые рекомендации W3C

Консорциум *W3C* (World Wide Web Consortium) постоянно рассматривает и разрабатывает новые технологии. Сведения обо всех рассматриваемых в настоящее время рекомендациях вы найдете по адресу `http://www.w3.org/Cosortium/Process-20010208/tr`. Каждая новая технология начинает свое существование как нота (note), которая отправляется в консорциум *W3C* одним из его членов. После этого она проходит через такие этапы развития.

- Рабочий черновик (Working Draft). Рабочий черновик содержит подробный план действий рабочего комитета по расширению описания новой технологии.
- Last Call Working Draft (Последний вариант рабочего черновика). На этом этапе все члены консорциума работают с рабочим черновиком и добавляют к нему свои замечания.
- Candidate Recommendation (Кандидат в рекомендации). На этом этапе разработчики и пользователи могут реализовать новую технологию в соответствии с рабочим черновиком.
- Proposed Recommendation (Предварительная рекомендация). На этом этапе признаются успешная реализация и выполнение всех требований, сформулированных рабочей группой.
- W3C Recommendation (Рекомендация W3C). Рекомендация — это результат интенсивных исследований, чаще всего компромиссный вариант, одобренный всеми участниками консорциума.

Рекомендация XML 1.0

Это официальная рекомендация XML версии 1.0 консорциума *W3C.* Она относительно коротка и достаточно понятна при прочтении.
`http://www.w3.org/TR/REC-xml`

Рекомендация XML Schema: Part 0, Primer

В настоящее время язык описания схем XML Schema консорциума *W3C* получил статус рекомендации. В такой части документа, как Part 0, Primer, вы найдете обзор спецификации, из которого узнаете, каким образом можно добавлять ограничения для экземпляров XML для обеспечения их структурной целостности.

`http://www.w3.org/TR/xmlschema-0/`

Рекомендация XML Schema: Part 1, Structures

В документации XML Schema: Part 1, Structures вы найдете описание концепции и синтаксиса языка описания схем XML Schema.

`http://www.w3.org/TR/xmlschema-1/`

Рекомендация XML Schema: Part 2, Datatypes

В документации XML Schema: Part 2, Datatypes вы найдете определение типов данных XML, используемых в выражениях XML Schema и других словарях XML.

`http://www.w3.org/TR/xmlschema-2/`

Рекомендация XSL Transformations (XSLT) Version 1.0

Синтаксис и семантика XSLT подробно описаны именно в этой рекомендации. Из нее вы узнаете, каким образом использовать XSLT для преобразования документов, созданных с помощью одного языка разметки, к новой структуре на том же или на другом языке.

`http://www.w3.org/TR/xslt`

Рекомендация XML Path Language (XPath) Version 1.0

XPath — это язык, который используется для определения месторасположения нужных фрагментов в документе XML. Документация касается использования XPath в выражениях XSLT и XPointer.

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

Объектная модель документа DOM

Приведенные ниже ссылки укажут вам на целый ряд Web-узлов, которые касаются различных аспектов использования модели DOM по отношению к документам XML.

- Спецификация Document Object Model (DOM) Level 1:
  `http://www.w3.org/TR/REC-DOM-Level-1/`
- Спецификация Document Object Model (DOM) Level 2 Core:
  `http://www.w3.org/TR/REC-DOM-Level-2-Core/`
- Спецификация Document Object Model (DOM) Level 2 Views:
  `http://www.w3.org/TR/REC-DOM-Level-2-Views/`
- Спецификация Document Object Model (DOM) Level 2 Events:
  `http://www.w3.org/TR/REC-DOM-Level-2-Events/`
- Спецификация Document Object Model (DOM) Level 2 Style:
  `http://www.w3.org/TR/REC-DOM-Level-2-Style/`
- Спецификация Document Object Model (DOM) Level 2 Transversal and Range:
  `http://www.w3.org/TR/REC-DOM-Level-2-Transversal-Range/`

Пространства имен XML

Спецификацию, касающуюся использования пространств имен в XML, вы найдете по следующему адресу:

`http://www.w3.org/TR/REC-xml-names`

Рекомендация XML Base

Рекомендация по поводу использования относительных адресов URL содержится по следующему адресу:

`http://www.w3.org/TR/xmlbase/`

Рекомендация XML Linking Language (XLink) Version 1.0

Язык XLink обеспечивает возможность использования гипертекста в документах XML. Рекомендацию по поводу этого языка вы найдете по следующему адресу:

`http://www.w3.org/TR/xlink/`

Рекомендация XML Information Set

Если в какой-то спецификации необходимо сослаться на рекомендацию XML 1.0 в той или иной форме, набор всех допустимых определений вы найдете в этом документе. Этот документ вообще можно считать словарем, содержащим сведения о наиболее важных концепциях XML и соответствующую терминологию.

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

Рекомендация Stylesheet Language (XSL) Version 1.0

В этой рекомендации вы найдете описание того, каким образом можно использовать язык XSL для создания таблиц стилей, определяющих словарь форматирования, предназначенный для представления целого класса документов XML.

`http://www.w3.org/TR/xsl`

Некоторые рабочие черновики консорциума W3C

XML Inclusions (XInclude) Version 1.0

В этом черновике описан способ включения целых документов XML или их частей в составные документы XML.

`http://www.w3.org/TR/xinclude`

XML Pointer Language (XPointer) Version 1.0

Объединяя идентификаторы URI с выражениями XPath, вы сможете обнаружить определенные данные в документе, используя язык XPointer, описанный в данном черновике.

`http://www.w3.org/TR/xptr`

XPath Requirements Version 2.0

Здесь можно найти дополнительную информацию об XPath.

`http://www.w3.org/TR/xpath20req`

XSLT Requirements Version 2.0

Здесь можно найти дополнительную информацию о следующей версии XSLT.

`http://www.w3.org/TR/xslt20req`

Спецификация Document Object Model (DOM) Level 3 Core

Спецификации DOM Level 1 и DOM Level 2 уже полностью закончены. В настоящем черновике вы найдете предложения, касающиеся версии Level 3 объектной модели DOM. Эта версия расширяет функциональные возможности двух предыдущих.

`http://www.w3.org/TR/DOM-Level-3-Core/`

Simple Object Access Protocol (SOAP) 1.2

SOAP — этот протокол, предназначенный для организации обмена информацией на основе транзакций в децентрализованной распределенной среде посредством вызовов методов удаленных объектов.

`http://www.w3.org/TR/2001/WD-SOAP12-20010709`

Некоторые ноты консорциума W3C

Web Services Description Language (WSDL) 1.1

В настоящем документе WSDL определяется как формат XML для поиска сетевых служб. WSDL можно использовать совместно с SOAP, HTTP и другими коммуникационными протоколами.

`http://www.w3.org/TR/wsdl`

Рекомендуемые Web-узлы, содержащие сведения об XML

Tim Bray's Annotated Version of XML 1.0 Recommendation

Тим Брей — один из соавторов рекомендации XML. Он предлагает собственные комментарии и замечания по поводу этой рекомендации.

`http://www.xml.com/axml/axml.html`

The XML Cover Pages (by Robin Cover)

Это один из лучших Web-узлов в Internet. Здесь вы найдете ссылки на программное обеспечение и текущие версии спецификаций различных технологий XML. Содержание узла регулярно обновляется и достаточно полно охватывает тематику.

`http://oasis-open.org/cover/xml.html`

Apache XML Project

Организация *Apache XML Project* предлагает высококачественное программное обеспечение для разработки приложений XML. Многие из предложенных решений

перенесены с Java на C++, PERL или COM. Здесь вы найдете синтаксический анализатор Xerces; обработчик таблиц стилей XSLT Xalan; средство публикации в Web на основе XML Cocoon; средство XSL-FO FOP; средство быстрой разработки динамических страниц Xang; средства поддержки XML SOAP; инструмент для создания объектов SVG Batik, а также полнофункциональный синтаксический анализатор Crimson.

`http://xml.apache.org`

IBM XML Developer Works

Web-узел IBM XML Developer Works предлагает исчерпывающие сведения о программных продуктах и технологии XML. На нем вы найдете учебные пособия, бесплатное программное обеспечение и многое другое.

`http://www-106.ibm.com/developerworks/xml/`

Microsoft Developers Network (MSDN)

Web-узел MSDN предоставляет доступ к сведениям о работе со всеми средствами разработки компании *Microsoft*. В разделе, посвященном XML, вы найдете бесплатные синтаксические анализаторы, инструменты для работы с SOAP, разработки Web-служб, различные наборы инструментальных средств SDK, а также руководства.

`http://msdn.microsoft.com/library/default.asp?url=/nhp/Default.asp?contentid=28000438`

SAX 2.0: The Simple API for XML

SAX — это стандартный интерфейс для обработки экземпляров документов XML на основе событий.

`http://www.megginson.com/SAX/`

Полезные редакторы XML

В этом разделе приводятся сведения о некоторых программных продуктах, предназначенных для разработки приложений XML.

XRay XML Editor

Редактор XRay XML Editor, разработанный компанией *Architag International*, представляет мощную среду разработки для создания документов XML, схем, а также документов XSL, XSLT, HTML и DHTML. Есть средство проверки ошибок, которое выполняет свою работу после каждого нажатия клавиши. Данный редактор можно совершенно бесплатно загрузить со следующего Web-узла:

`http://www.architag.com/xray/`

XML Spy

Редактор XML Spy — это интегрированная среда разработки IDE для XML. Поддерживаются такие возможности, как редактирование, проверка на действительность, а также выполнение преобразований. Кроме того, есть мощные инструменты для создания схем DTD, XSD, XDR и ряда других на основе экземпляра документа XML. Схему, созданную с помощью одного языка описания, с помощью XML Spy очень легко преобразовать к другому языку.

`http://www.xmlspy.com/`

Turbo XML

Turbo XML версии 2.0 — это полнофункциональный коммерческий редактор. Он представляет собой интегрированную среду разработки IDE, поддерживающую язык описания схем XSD консорциума *W3C*, а также предлагает средства для создания, преобразования и проверки экземпляров схем. Ознакомительную версию этого редактора вы можно загрузить с Web-узла компании *Tibco Extensibility*.

Другие программные средства для работы с XML

Практически невозможно составить полный список всего программного обеспечения для разработки приложений XML. Достаточно полный и постоянно изменяющийся список программных средств, доступных для разработчиков, вы найдете на Web-узле *XML Software*. Здесь вы найдете списки различных категорий программного обеспечения, таких как инструменты для сериализации объектов, инструменты преобразования, интерфейсы для работы с базами данных, системы управления содержанием, редакторы схем, прикладные интерфейсы API, инструменты, утилиты, программы для создания Web-служб, броузеры, редакторы, синтаксические анализаторы и многое-многое другое.

`http://www.xmlsoftware.com/`

Предметный указатель

A

Apache FOP, *267*

B

B2C-приложения, *362*

D

DOM, *30*
  вызов, *211*
  для XML, *207*
  отношения, *210*
  сообщения об ошибках, *221*
DTD, *29*

E

ebXML, *374*

H

HTML, *16*; *21*
HTTP, *21*

S

SAX, *30*
  методы, *234*
  обработка ошибок, *241*
  синтаксические анализаторы, *226*
SAX, *225*
SOAP, *369*

U

URL, *21*

W

WDDX, *373*
Web, *20*
Web-приложение
  проектирование, *379*

X

XBase, *30*
XDR, *29*
XInclude, *30*
XLink, *29*
  ограничения, *187*
  описательные атрибуты, *191*
XML
  декларация, *45*
  добавление комментариев, *47*
  правильный экземпляр, *51*
  разработка Web-приложений, *379*
  синтаксические правила, *38*
  ссылки, *137*
XMLDSO, *328*
XML-RPC, *369*
XPath, *29*
  взаимосвязи узлов, *159*
  узлы документа, *158*
XPointer, *30*
  выражения, *201*; *335*
  идентификационные ссылки, *202*
  синтаксис, *201*
XSD, *29*
XSL, *30*
  использование имен, *265*
XSL-FO, *30*
XSLT
  сортировка, *304*
XSLT, *30*
  использование, *290*
  итерационная обработка, *296*
  параметры обработки, *287*

А

Абсолютная адресация, *203*
Агент пользователя, *57*
Анализ документов
  основные положения, *338*
Анализ экземпляров XML, *84*

Атрибут
xml
base, *331*
маркированных типов, *75*
шрифта, *278*

Б

Бизнес-приложения, *341*

В

Вложение элементов, *44*
Выбор
атрибутов, *177*
всех элементов по имени, *176*
всех элементов по их предкам, *176*
дочернего элемента корневого элемента, *175*
корневого элемента, *174*
узла по его расположению, *218*
узлов по имени с помощью косой, *174*
числовой, *204*
элементов XML по имени, *219*
элементов по известному направлению, *176*
Выражения отношений, *203*

Г

Гипертекст, *183*

Д

Данные
структурированные, *24*
Декларация XML, *45*
Документ XML, *37*

И

Индикатор вхождения, *82*
в схемах XDR, *113*
Использование
атрибутов ID, *76*
атрибутов IDREF, *76*
атрибутов NMTOKEN, *79*
ключевого слова ANY схемы DTD, *70*
подстановочных символов, *176*
элементов ENTITY, *78*

К

Каскадные таблицы стилей
поддержка броузерами, *247*
свойства, *252*
Каскадные таблицы стилей, *246*
Класс
импортирование, *231*
Консорциум W3C, *22*
Конфликт имен, *147*

М

Модель
объектная, *209*
смешанного содержания, *71*

Н

Настройка
Web-сервера, *351*

О

Объект
форматирования, *269*
Объявление
ElementType, *98*
атрибутов строкового типа, *73*
атрибутов типа #FIXED, *74*
атрибутов типа #IMPLIED, *75*
внешних схем DTD, *88*
пустого элемента, *69*
элементов и атрибутов XDR, *98*
Ограничение вхождений
в схемах XSD, *131*
Определение
групп содержания, *116*
Отношение
дочерний узел, *165*
потомок самого себя, *169*
Предок самого себя, *167*

П

Парсер, *35; 38; 59*
Правила, *66*
Преобразование
XML в HTML, *293*

Применение XML в бизнесе, *361*
Проверка
атрибутов с помощью схемы XDR, *105*
документов с помощью SAX, *242*
Пространство имен, *29*
XSD, *125*

Р

Роль электронного бизнеса, *32*

С

Свойства
отображения, *257*
рамки, *255*
стилей текста, *252*
цвета и фона, *254*
шрифтов, *253*
Связывание
документа XML с HTML-страницей, *318*
элементов HTML с элементами XML, *320*
Содержание, *26*; *245*
Создание
HTML-страницы с помощью XSLT, *301*
HTML-таблиц, содержащих данные XML, *321*
Web-узла, *392*
документа XML с помощью сценария, *355*
Ссылка
HTML, *184*
общая, *138*
параметрическая, *141*
представление, *191*
приведение в действие, *192*
простая, *188*
расширенная, *188*
Стиль, *245*
определение, *250*
элементов, *280*
Структура
добавление форматирования, *264*
документа XML, *316*
Структура данных, *56*
Схема
XML-Data Reduced, *95*
языка DDML, *96*
Схема
DOD, *96*
DTD, *63*; *95*
SOX, *96*
XML-Data, *95*
DTD
объявление атрибутов, *73*
XDR
преобразование, *120*
расширяемость, *120*
XSD
атрибуты, *133*
основы, *125*

Т

Таблица
XSL-FO, *281*
Технология
XML Base, *331*
XML Include, *333*
Тип данных
XSD, *128*
в схемах XDR, *108*
примеры, *109*
проверка, *54*
Тип элемента
простой, *126*

У

Узел
self, *164*
последующий, *169*
потомок, *169*
предыдущий, *170*
родительский, *165*
сбора данных, *365*
тип, *204*
Уровень
данных, *346*
клиента, *345*
промежуточный, *345*

Ф

Фрагмент
именованный, *199*

Э

Экземпляр, *37*
Элемент, *26*
  arc, *196*
  xsl
    choose, *310*
    for-each, *309*
    if, *309*
    otherwise, *310*
    when, *310*
  вложенный, *26*; *104*
  локатор, *195*
  связующий, *187*
  содержание, *26*
  содержащий другой элемент, *68*
  создание, *37*

Я

Язык
  XHTML, *31*
  XLink, *183*
  XPath, *157*
  XPointer, *199*
  XSLT, *285*
  разметки, *21*